Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2024 SCPE. Volume 25, Issues 5, pp. 3651–3673, DOI 10.12694/scpe.v25i5.3030

REVIEW OF AUTOMATED TEST CASE GENERATION, OPTIMIZATION, AND PRIORITIZATION USING UML DIAGRAMS: TRENDS, LIMITATIONS, AND FUTURE DIRECTIONS

SRINIVASA RAO KONGARANA [*], A ANANDA RAO [†] AND P RADHIKA RAJU [‡]

Abstract. This systematic literature review examines the effectiveness of automated test case generation, optimization, and prioritization methods based on Unified Modeling Language (UML) diagrams. The review summarizes the methods, main contributions, and limitations, and suggests areas for future research. This paper examines various optimization algorithms, model-based testing methods, and UML diagram validation methods to determine how well they perform. The review highlights some issues with the current situation, such as the fact that it only examines a few types of UML diagrams and does not go into great detail about how they work or compare to other diagrams. However, it also suggests ways in which these issues could be addressed in future research. Some of the suggested directions include researching different modeling languages and devising solutions to handle the complexity of system models. Model-based testing should also be combined with optimization and prioritization methods to increase the flexibility and usefulness of research in this field. This article makes no direct comparisons to UML diagrams, but it does provide a thorough discussion of the current state of the art and a list of strategic priorities to advance the field of automated test case generation, optimization, and prioritization. These reviews are useful for both researchers and practitioners because they demonstrate how things are currently done and how they should be done in the future.

Key words: UML, Optimization, Prioritization, Test Case Generation, Sequence Diagram, SLR

1. Introduction.

Software testing finds and fixes system bugs, which is a critical part of the software development lifecycle. Creating inputs and expected outputs to assess the software's functionality and behavior is known as test case generation, and it is an essential part of software testing [1]. The ability of automated test case generation techniques to increase software testing effectiveness, decrease human labor, and increase efficiency has drawn attention in recent years [1]. Test case generation using Unified Modeling Language (UML) diagrams shows promise for automated testing. Software systems can be visually represented with UML diagrams, which depict interactions, behavior, and structure [2]. These diagrams are appropriate for automated test case generation because they standardize and simplify software system modeling. From 2010 to 2022, a thorough overview of research on creating test case diagrams from UML diagrams is intended to be provided by this systematic literature review. This systematic review outlined the field's present methods, results, research trends, shortcomings, and potential future directions. The use of UML diagrams in test case generation has several advantages. Between test cases and system requirements or design specifications, UML diagrams provide a clear mapping [3]. Making it simpler to track test coverage and ensure system behavior matches design, traceability enhances understanding and documentation of the testing process.

Because UML diagrams automatically generate test cases and model system behavior, they also enhance model-based testing [4]. Automated test case generation is minimized, dynamic behavior is managed, and system complexity is captured through model-based capturing. Efficient system functionality coverage is another advantage of UML diagrams. A comprehensive view of the system is provided by UML diagrams, which enable testers to pinpoint crucial paths, significant interactions, and test scenarios [5]. With the aid of this data, automated test case generation techniques are able to produce an exhaustive collection of test cases that encompass every facet of the system, guaranteeing a comprehensive assessment of its functionality.]

UML diagrams have limitations when it comes to test case generation. One drawback is the emphasis on state machine, activity, and sequence UML diagrams. These diagrams show important system behavior,

[*]CSE Department, JNTUA College of Engineering (Corresponding author, srinivas.cst4@gmail.com)
[†]CSE Department, JNTUA College of Engineering (akepogu@gmail.com).
[‡]CSE Department, JNTUA College Of Engineering (radhikaraju.p@gmail.com).

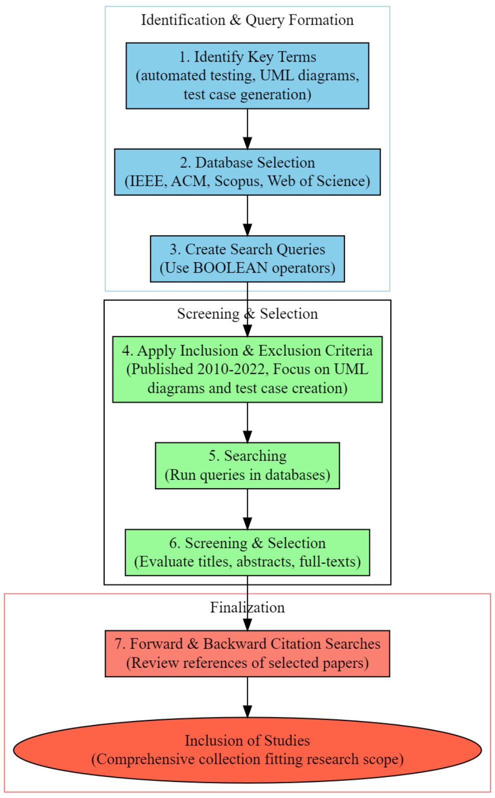
Identification & Query Formation
1. Identify Key Terms (automated testing, UML diagrams, test case generation)
2. Database Selection (IEEE, ACM, Scopus, Web of Science)
3. Create Search Queries (Use BOOLEAN operators)
Screening & Selection
4. Apply Inclusion & Exclusion Criteria (Published 2010-2022, Focus on UML diagrams and test case creation)
5. Searching (Run queries in databases)
6. Screening & Selection (Evaluate titles, abstracts, full-texts)
Finalization
7. Forward & Backward Citation Searches (Review references of selected papers)
Inclusion of Studies (Comprehensive collection fitting research scope)

Fig. 2.1: Architecture of the systematic literature review (SLR)

but they may not cover all software details, resulting in test coverage gaps. Evaluate whether UML-based test case generation is applicable to complex systems as well. UML diagrams might not adequately depict the complex behavior, interdependencies, and edge cases of complex software systems. Assessing the effectiveness and scalability of these techniques is essential before applying them to large, complex systems.

Empirical evaluations and comparative analyses of the suggested approaches are crucial to assess their effectiveness, efficiency, and practicality. The lack of thorough empirical evaluations in many of the current studies limits their understanding of adoption and performance in practice. Consolidating knowledge and identifying gaps and limitations in automated test case generation using UML diagrams are the goals of this systematic literature review. Insights into new trends will be provided by this review, which will also help to clarify the current research landscape. By addressing limitations, investigating new avenues for research and development, and promoting the adoption of successful and efficient test case generation methods in software testing practices, the objective is to advance the field of automated test case generation.

2. Search Strategy. Figure 2.1 depicts the systematic literature review (SLR), which used a thorough and methodical search strategy to uncover relevant literature on creating test cases from UML diagrams. The method was carefully planned to ensure that a comprehensive set of research papers were assembled that met the specific criteria for inclusion and covered the entire scope of the research question. With this focused and organized approach, the review sought to compile a body of work that demonstrates the current state and recent progress in creating test cases from UML diagrams.

The following steps were used to create an efficient search strategy:

1. Finding the key terms and concepts associated with the research topic: ”automated testing,” ”UML diagrams,” ”test case generation,” and other related terms.

2. Database selection: Digital libraries and pertinent scholarly databases were looked through. In computer science, software engineering, and testing, common databases include IEEE Xplore, ACM Digital Library, Scopus, and Web of Science.
3. Creating search queries: The key terms and concepts were combined to create search queries. The search space was widened and terms were connected using BOOLEAN operators (AND, OR). UML diagrams were used in the search queries to locate studies on test case generation.
4. Application of inclusion and exclusion criteria: Clearly defined inclusion and exclusion criteria were created in order to weed out studies that did not fit the intended research scope. Published in English between 2010 and 2022, with a focus on UML diagrams and test case creation.
5. Searching: Queries were run in a few databases to get the desired results. Consistency and accuracy were sought after by several researchers working independently.
6. Methods of screening and selection: The titles and abstracts of the retrieved studies were evaluated for their applicability to the research question. The full-text eligibility of the chosen studies was subsequently assessed for systematic literature reviews.
7. Forward and backward citation searches: To increase the search's rigor, we looked through the reference lists of a few chosen papers and carried out forward and backward citation searches to locate more pertinent studies that the first search had overlooked.

In order to reduce bias, incorporate pertinent studies, and offer a thorough and representative sample of UML diagram test case literature, this systematic search approach was employed. The method found and chose studies that supported the systematic literature review and the research goals.

2.1. Search Queries. Compatible search queries are terms or keywords that align with the objectives and topic of a study. In databases and other sources, these queries discover relevant and compatible literature.

Creating compatible search queries requires researchers to take into account the key concepts, terms, and relationships related to their research topic. While excluding irrelevant studies, the queries should cover a wide range of relevant research. Additionally, they should consider variations in terminology or synonyms used in the literature.

Here are some search queries related to perform suggested review:

1. Automated test case generation techniques using UML diagrams
2. Trends in test case generation, optimization, and prioritization with UML diagrams
3. Limitations of automated test case generation with UML diagrams
4. Future directions in test case generation, optimization, and prioritization using UML diagrams
5. Model-based testing and UML diagrams for automated test case generation
6. Optimization algorithms for test case generation with UML diagrams
7. Validation techniques for automated test case generation using UML diagrams
8. Comparative analysis of automated test case generation approaches with UML diagrams
9. Handling complexities in system models for test case generation with UML diagrams
10. Integration of optimization methods, prioritization techniques, and model-based testing for test case generation with UML diagrams

These search queries can assist researchers and practitioners in exploring pertinent literature, trends, limitations, and future directions in the field of automated test case generation, optimization, and prioritization using UML diagrams.

2.2. Search Statistics. Table 2.1, Figure 2.2 categorizes scholarly articles based on the frequency of keywords related to test case development techniques using Unified Modeling Language (UML) diagrams. It shows that "UML" is the most prevalent keyword, appearing in 36 articles, which constitutes 69% of the literature. This is followed by "Test Case Generation" with 30 articles (58%), suggesting a strong academic focus on these areas. Less prevalent, though still significant, are articles on "Sequence Diagram" and "Activity Diagram," with 8 (15%) and 14 (27%) articles respectively, indicating a moderate research interest. "Test Case Optimization" and "Test Case Prioritization" are covered in 11 (21%) and 9 (17%) articles, pointing towards an interest in enhancing the efficacy of testing procedures.
Finally, "UML Diagrams" as a collective category feature in 23 articles, making up 44% of the distribution, highlighting the central importance of UML in the discourse of test case methodologies. This tabulation

Table 2.1: Distribution of Articles According to Keywords

S.No	Number of Articles	Keywords	Ratio of Articles
1	36	UML	69.00%
2	30	Test Case Generation	58.00%
3	8	Sequence Diagram	15.00%
4	14	Activity Diagram	27.00%
5	11	Test case optimization	21.00%
6	9	Test Case Prioritization	17.00%
7	23	UML Diagrams	44.00%

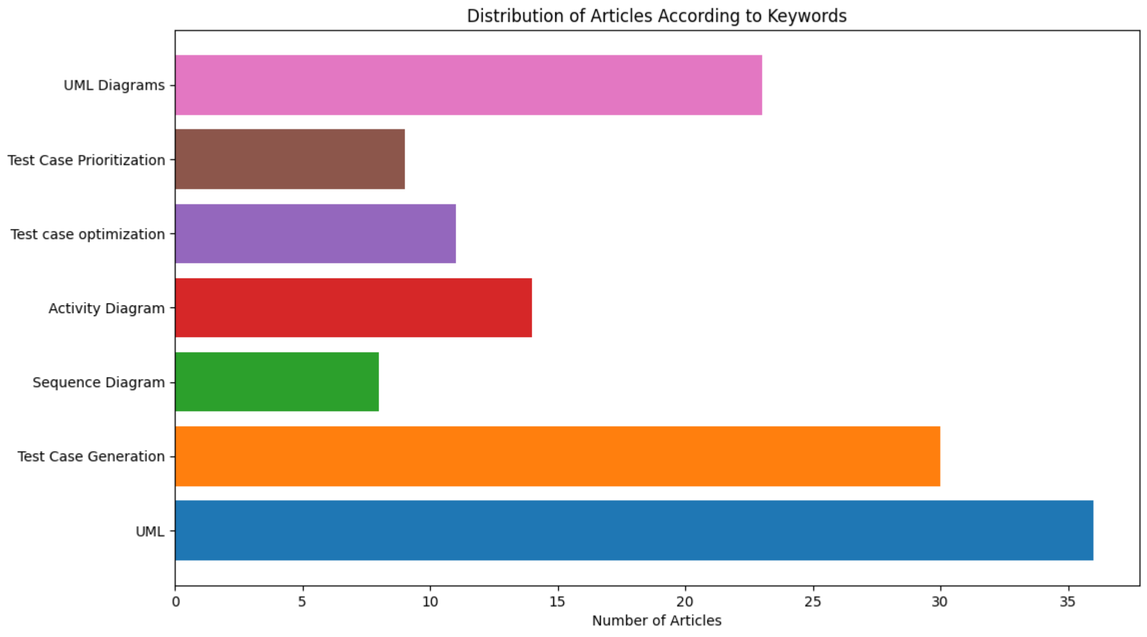

Fig. 2.2: The distribution of articles according to keywords

underscores the varied levels of academic engagement across different facets of test case methodologies within the realm of UML.

A filter was also applied to the publication type of the articles, resulting in the removal of 5 articles. The selected articles' type distribution is displayed in Table 2.2, Figure 2.3. The publisher reviewed the 52 remaining articles, discarding none. By publisher, Table 2.2 lists the articles.

The Figure 2.4, Table 2.3 resulting in the removal of two articles, articles were filtered in the final stage based on their year of publication. After applying qualitative synthesis factors to the remaining 52 articles, two more were discarded. Table 2.3 displays the articles' publication years.

2.3. Research Questions.

Research Question 1: What is the most effective and efficient approach for automated test case generation in object-oriented software development, considering the use of UML behavioral models and diagrams?

Research Question 2: What are the most effective and efficient approaches for automated test case generation in software development, considering the utilization of optimization techniques and UML diagrams?

Research Question 3: How can automated test case generation techniques using UML diagrams be effectively

Table 2.2: Distribution of Articles by Publisher

S.No	Publisher	Ratio of articles
1	Elsevier	8%
2	Springer	6%
3	IEEE	2%
4	Conference	23%
5	Other Journals	62%

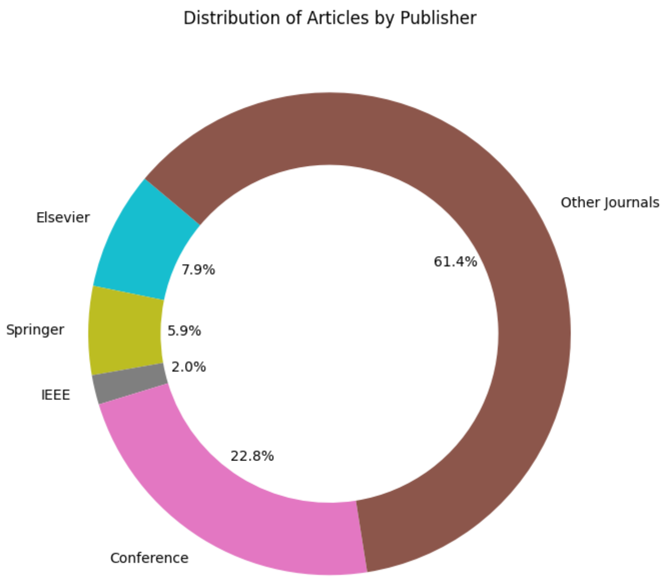

Fig. 2.3: The distribution of articles by publisher

utilized to improve software development processes, considering optimization methods, soft computing techniques, and the automation of test case generation?

Research Question 4: How can automated test case generation techniques utilizing UML diagrams be improved to address limitations such as applicability to specific diagram types, complexity limitations, and lack of comparative analysis, while considering factors like model-based testing, optimization, and validation approaches?

Research Question 5: How can the limitations of existing approaches for test case generation and optimization in model-driven software development using UML diagrams be overcome to improve their applicability, effectiveness, and efficiency?

Research Question 6: How can test case prioritization techniques be enhanced to address the limitations identified in the reviewed papers?

3. Literature Review. This review on ”Test Case Generation and Optimization” presented in section 3.1 provides valuable insights into the state of software testing today by examining the intricate processes and state-of-the-art methodologies. The architecture diagram shown in figure 3.1 that serves as a visual guide for the plethora of studies, methodologies, and findings that make up this field is used to describe our systematic literature review strategy in detail. These images not only demonstrate the meticulous manner in which data was gathered and examined, but they also demonstrate the connections between various research topics,

Table 2.3: Fraction of articles by year of publication

Article Count	**Publish Year**
15	2019
7	2020
12	2021
5	2022
13	Others

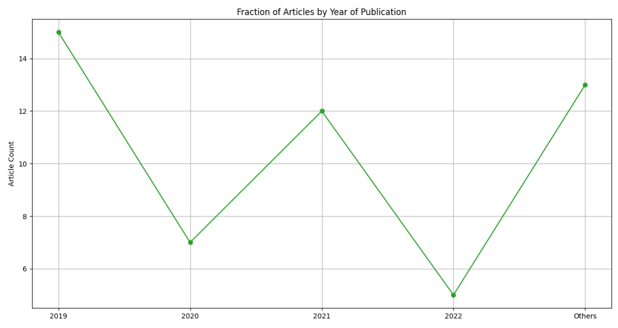

Fig. 2.4: Fraction of articles by year of publications

ranging from the most recent optimization algorithms to the use of UML diagrams to create test cases. By assembling an extensive array of contemporary models and methodologies, this review seeks to provide readers with a comprehensive understanding of test case generation and optimization. It also emphasizes how crucial systematic reviews are to expanding the realm of software testing possibilities.

The review presented in section 3.2 underscores the significance of prioritizing test cases to boost the effectiveness of software testing procedures. The central component of our study is a meticulously constructed architecture diagram that shown in figure 3.2. It demonstrates the sequential and iterative processes involved in prioritizing test cases, from locating them and grouping them according to their importance to applying various prioritization criteria and algorithms and ultimately executing the tests in an orderly fashion. The iterative refinement inherent in test case prioritization criteria is highlighted by the feedback loop for modifying test case prioritization based on empirical results. We aim to provide you with a comprehensive understanding of the approaches, issues, and advancements in test case prioritization by based our review on this architectural framework. Researchers and practitioners who are attempting to enhance software testing outcomes will benefit from this.

3.1. Test case Generation and Optimization. The use of UML diagrams to streamline test cases in software development was covered by Tiwari, R. G., et al. [1]. Contribution includes recommending the usage of UML diagrams (activity, state, and sequence) and outlining their benefits for streamlining test cases. The technique lacks any actual data or experiments and is based solely on a survey of the literature. The goal is to educate software engineers about the usage of UML diagrams for test case simplification. The absence of empirical data, unresolved difficulties or constraints, and the narrow focus on just three categories of UML diagrams are all drawbacks. To establish efficacy and resolve any difficulties, more study is required.

J. Cvetkovi and M. Cvetkovi., [3] provided a research on the creation of test cases with UML diagrams that concentrated on modelling depression brought on by internet addiction. By recommending a technique for creating test cases using UML diagrams and offering a case study to illustrate its use, the paper makes a contribution to the subject. The process comprises categorising test case creation based on unit diagrams or combinations of UML diagrams and employing a variety of UML diagrams. The objective is to advance the area of software testing by offering a fresh method for creating test cases. The case study's specificity

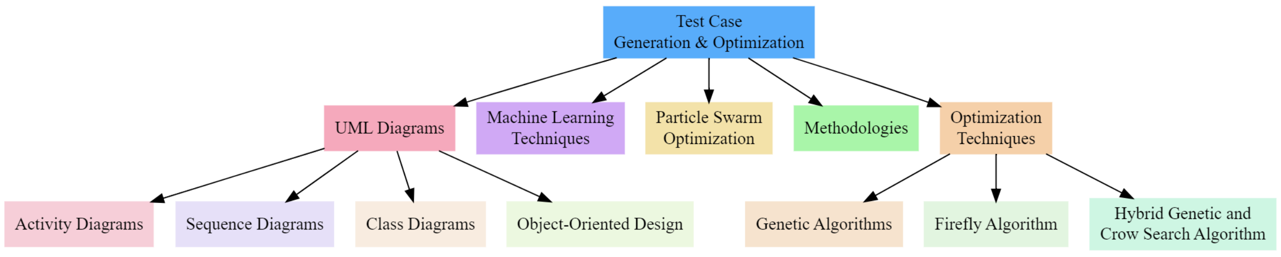

Fig. 3.1: The process of Test Case Generation and Optimization

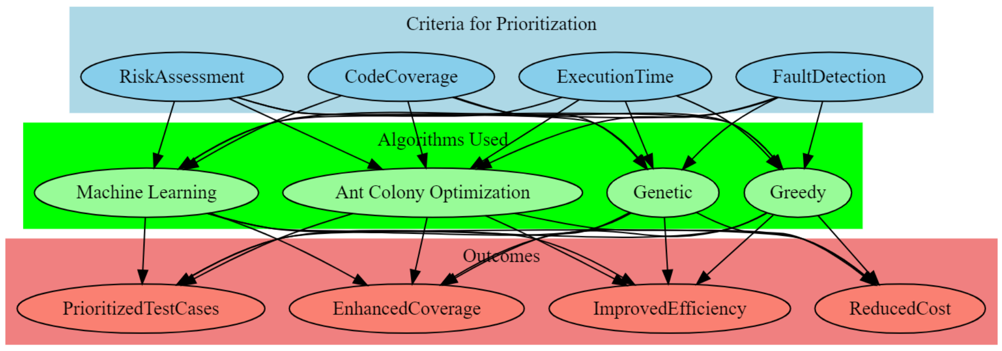

Fig. 3.2: Process model of the Test case Prioritization

and the absence of comparison with other approaches are also drawbacks. An approach for creating test cases from UML sequence diagrams combining MDE (“model-driven engineering”) and MBT (“model-based testing”) methodologies was given by Rocha, M., et al. in [5]. Modifying transformation rules, using real software models in a case study, and using the ModelJUnit and JUnit libraries are all part of the process. The contribution consists of a methodical process for automatically creating test cases from UML models, which raises the productivity and calibre of software development. The inability to use nested combined fragments in sequence diagrams, the manual creation of stubs, and the limited usage of one case study for demonstration are some limitations.

B. N. Biswal. Using UML behavioural models, [6] suggested a technique for creating and refining test cases of object-oriented software. It offers a technique for creating test cases that uses UML activity diagrams, sequence diagrams, as well as class diagrams. An error minimization method for test case optimisation is also presented in the study. Its application to particular categories of object-oriented software as well as reliance on the excellence of UML behavioural models are limitations.

Meena, D.K. [7] A technique for creating test cases using UML diagrams, notably Interaction Overview diagrams as well as Sequence diagrams, was presented by [7]. The difficulty of test case selection in object-oriented programmes is what this aims to address. Building UML diagrams, creating intermediate graphs, and creating test cases that reflect various scenarios are all part of the technique. The paper’s contribution is the message route coverage. Limitations include the fact that it only applies to object-oriented programmes and the absence of actual data contrasting the suggested strategy with other ones.

Jiang, L., and Li, Y. A technique for creating test cases using UML sequence diagrams was covered in [8]. By examining the sequence diagram as well as assessing the message sequence, it gives instructions for creating test cases. The idea is to provide a useful method for testing object-oriented applications. The study does not, however, present a prototype system for autonomous test case generation, and it does not discuss the drawbacks or viability of the suggested approach in practical contexts.

A. Herrmann, M. Felderer, and M. [9] looked at mistakes occurring while manually deriving test cases from state machines and activity diagrams using UML. The goal is to offer instructions for methodically generating test cases while avoiding mistakes. Participants in the methodology's controlled student experiment create test cases from state machines and UML activity diagrams. The study offers a taxonomy of faults, evidence that activity diagrams are more error-prone than state machines, and recommendations for minimising errors. The utilisation of student participation and the emphasis on UML activity diagrams as well as state machines are limitations.

An approach for creating test cases using UML sequence diagrams as well as interaction overview diagrams was proposed by Jena, A. K., et al. in [10]. As part of the technique, produced graphs from various UML components are combined to create test cases and find errors early in the design process. The goal of the study is to identify various error types and raise software quality. The emphasis on a single case study and the lack of a comparison with alternative techniques for creating test cases are both drawbacks.

A. Herrmann, M. Felderer, and M. [11] findings of a controlled experiment [11] examining mistakes caused during manual test case derivation from state machines and UML activity diagrams were provided. The study offers a taxonomy of faults, evidence that activity diagrams are more error-prone than state machines, and recommendations for minimizing errors. The concentration on certain diagram kinds without comparison to other types or automated test case derivation techniques, as well as the sample size's relative smallness, are limitations.

A technique for creating test cases using state chart diagrams and UML use case diagrams was put out by Jagtap, S., et al. [12]. Utilizing testing criteria that address the diagrams' state and transition is part of the technique. The goal is to aid software engineers in the early phases of software development in the creation of efficient test cases. Limitations include the emphasis on use case diagrams as well as state chart diagrams in UML 2.0 and the potential need for manual testing to ensure thorough coverage.

An automated test case creation method for unstructured SysML "Activity Diagrams" (Ads) was provided by Yin, Y., et al. [13]. The process entails converting the ADs into an IBM ("Intermediate Black Box Model"), which is used to generate test cases. The objective is to offer industry practitioners employing SysML ADs an efficient testing methodology. The concentration on the AD model alone, without taking into account other SysML models, is the restriction.

In order to create test cases from UML state chart diagrams, Salman, Y. D., et al. [14] concentrated on finding the most efficient combination of coverage requirements, notably in handling loops. The study suggests appropriate coverage criteria and offers formulas to calculate coverage percentages. The goal is to increase test data generation's efficiency during software testing. The emphasis on UML state chart diagrams and the requirement for context adaption are limitations.

A method for automated test case development utilising UML Statechart as well as Sequence diagrams was proposed by Efendi, N. B. M., et al. [15]. The approach includes using UML diagrams to create test cases automatically. The aim is to keep the price of software development as low as possible. The drawback is the emphasis on UML Statechart and Sequence diagrams at the expense of other UML diagram types.

PSO ("Particle Swarm Optimization") was described by Prakash, V. C., et al. [16] as a tool for automated test case development in combinatorial testing. In order to address the optimisation issue in test case generation, the study offers a comprehensive assessment of PSO and its variations. The goal is to demonstrate how PSO-based algorithms may be used to reduce test suite sizes and boost programme dependability. The emphasis on PSO and its comparison to other optimisation methods is one limitation.

Using a dynamic programming technique and tester specification, Kamonsantiroj, S., et al. [17] suggested a memorising strategy for producing test cases in concurrent UML activity diagrams. The route explosion challenge in testing concurrent systems is the key contribution. The goal is to prevent the development of all feasible concurrent activity routes in order to reduce the explosion of test cases. The application to other diagram types or concurrent systems, as well as the emphasis on concurrency testing in activity diagrams, are limitations.

Mokhati, F.; Dehimi, N. E. H. A novel method for evaluating multi-agent systems using AUML sequence diagrams was put out by [18]. The method addresses interactions between actors and potential scenarios in a parallel, exclusive, or inclusive manner. To make sure that faults found in one interaction or scenario are not connected to others that are occurring simultaneously or in parallel, it provides plugs utilising OCL restrictions. The goal is to offer a testing strategy that guarantees mistake independence. Only applying to holonic agents and the absence of empirical proof of the efficacy of the suggested strategy are drawbacks.

Dawood, Y. S., and Hashim, N. L. [19] evaluated the prior research on the creation of test cases using UML statechart diagrams. It investigates the domain's algorithms, coverage standards, and assessment techniques. The goal is to highlight discrepancies and give a better knowledge of the test case generating procedures. A content analysis of 24 books in the topic is part of the technique. The emphasis on UML statechart diagrams and the little treatment of prior works are limitations.

A method to produce test cases from software requirements at the use case description level was put out by Alrawashed, T. A., et al. [20]. The method fine-tunes use cases, transforms them into control flow diagrams, and then use a tool to produce test cases. Regression testing is made more effective while test complexity is reduced and test coverage is improved. Limitations include the absence of comparison with current methods and a thorough evaluation of the case study findings.

Using machine learning, Khalifa, E. M., et al. [21] demonstrated a method for creating test cases from use case diagrams. The technique tries to automate the difficult and laborious process of test case generation. In order to increase precision and effectiveness, it employs a metaheuristic method. Information extraction, preprocessing, the use of the metaheuristic approach, and performance evaluation make up the methodology. The application to certain software systems and the reliance on the calibre of use case diagrams are limitations.

A method for creating optimised test cases from UML sequence diagrams using the Firefly algorithm was put forth by Runal, G., et al. [22]. The process of creating test cases will be automated, and the quality and dependability of software systems will be increased. Model-based testing and the structural test selection concept are used in the technique. The emphasis on UML sequence diagrams and possible drawbacks in big and complicated systems are constraints.

A technique for automatically creating executable test scripts for an IoT system using UML state machine diagrams was provided by Swain, R., et al. [23]. The goal is to make testing simpler and require less manual labour. The process entails the creation of mappings between actions and assertions as well as between transition events and functions. The algorithm creates transition pathways together with the corresponding assertions and actions. The contribution is a unique strategy that is shown by a case study on a diabetic monitoring and control system. Limitations include the absence of transition guard evaluation and the potential for method improvement through symbolic evaluation.

ALI, H. M. B. M. [24] addressed looping and iteration issues in order to improve the production of test cases from UML sequence diagrams. The contribution is an enhanced method that gives software testers a quick and easy approach to create test cases. A case study is used in the approach to show how the suggested strategy is used and how it compares to other strategies. The objective is to address the problem of producing test cases from UML sequence diagrams. The emphasis on looping and iteration issues as well as the need for knowledge of UML sequence diagrams are limitations. In their discussion of the automation of software development processes in telecom carrier networks, Kikuma, K., et al. [25] emphasised cost savings while upholding dependability and safety. The method's creation, which uses mathematical principles to boost learning effectiveness in software testing, is the contribution. The methodology makes use of the preparation of test cases and the use of mathematical techniques by qualified engineers. The report emphasises the significance of using the preparation procedure to its full potential. Data creation from already-existing design documentation is one limitation.

A methodical process for creating test cases from a UML model, especially from UML Sequence Diagrams, was provided by Rocha, M., et al. [26]. The contributions include the usage of the ModelJUnit and JUnit libraries for automated test case generation as well as the design of transformation rules using ATL. The goal is to provide UML Sequence Diagrams a defined meaning and make them appropriate for automated testing. ModelJUnit and JUnit libraries are used in the process, which entails translating UML Sequence Diagrams into Extended Finite State Machines. The emphasis on UML Sequence Diagrams and the efficacy relying on the calibre of the UML model are limitations.

Tiwari, H. Swathi, B. [27] investigates soft computing approaches like genetic algorithms and artificial bee colonies while focusing on test case creation in software testing. The contributions also explore soft computing approaches, provide an assessment criterion, and examine the value of code coverage in addition to underlining the significance of test case production. The methodology uses soft computing techniques to generate test cases and analyse code coverage to determine efficacy. To give a thorough grasp of test case generation and its importance is the goal. The emphasis on soft computing methods and the empirical study's constrained scope are both drawbacks.

Soni, D. Jain, P. [28] gave a survey on several methods for prioritising and generating test cases using UML diagrams. A thorough analysis of relevant work, a comparison of methods, and the identification of knowledge gaps are all included in the contributions. The process entails gathering data and comparing algorithms according to their efficacy. The goal is to outline test case creation techniques using UML diagrams and identify potential directions for further study. The emphasis on UML diagrams as well as the absence of a thorough analysis and useful implementation are limitations.

Using UML interaction diagrams, Minhas, N. M., et al. [29] developed a methodical mapping of test case generating approaches. The objective is to contrast various methods based on their strengths and weaknesses. The study evaluates the original studies' reporting quality and identifies any potential errors. An overview of the possibilities and constraints of test case generating methods utilising UML interaction diagrams is provided in this work. The report demonstrates that the examined studies lacked empirical evaluation in industry contexts and conformity to research principles. It implies that the industry needs stronger tool support for test case creation methods based on UML interaction diagrams.

Using class and activity diagrams, Mburu, J. M., et al. [30] suggested an improved multiview test case generating approach for object-oriented software. The creation and use of the MUTCAS Generator technology constitute the contribution. Utilising UML models, the technique creates test cases automatically that include both structural and behavioural aspects of the system. The aim is to allow early identification of software flaws and to solve the issues of time, money, and effort necessary for manual test case development. Limitations include the fact that they only apply to UML models, the possibility of problems in complicated systems, and reliance on the calibre of the UML models.

Dash and Panda, M. [31] An overview of the model-based and search-based testing strategies used to create test cases and test data for object-oriented programmes can be found in [31]. One aspect of the contribution is the framework for search-based testing that uses hybrid metaheuristics algorithms. The process includes a literature research to compile data on testing procedures and approaches. A framework for search-based testing of object-oriented programmes is proposed together with insights into the various testing methodologies. The suggested paradigm has limitations, such as the lack of a comparative analysis and empirical backing.

S. B. Tatale, V. C. Prakash, & Co. [32] concentrated on leveraging UML Sequence and Activity diagrams to automatically generate test cases. The contribution is a feasibility study on producing test cases focused on combinatorial logic using UML diagrams. The process entails employing dynamic slicing techniques and converting UML diagrams into tree or graph representations. The goal is to present a fresh method for creating test cases automatically from a system's design specification. The only emphasis on UML Sequence and Activity diagrams without taking into account other UML diagrams is one limitation.

K. Jin, K. Lano, and K. A systematic literature review (SLR) on creating test cases from UML diagrams was published in [33]. The contribution consists of identifying methods, results, research trends, gaps, and suggestions for future study. The approach entails running an SLR and using predetermined criteria to choose pertinent documents. The goal is to give a summary of the research on creating test cases from UML diagrams. The concentration on UML diagrams alone without taking other modelling languages into account is the restriction.

For the purpose of load testing mobile apps, Ali, A., et al. [34] suggested an autonomous model-based test case creation technique. An technique that minimises testing time while validating requirements, including both performance and functional elements, is the contribution. The process includes model-based testing, UML model creation, test case generation, and performance evaluation. The goal is to solve time-to-market restrictions and load testing of mobile applications. The absence of a thorough review and the choice of workload are limitations.

A technique for automatically creating test cases for flight control systems using UML state diagrams was developed by Fan, C., and Zou, P. [35]. A real-time extension strategy, a time domain equivalence partition method, and a feasibility check of the approach are the contributions. Modelling flight control systems, expanding UML state diagrams, creating equivalence classes, creating test pathways, and automating test case creation are all part of the technique. The goal is to increase the efficacy and efficiency of flight control system testing. Only being applicable to UML state diagrams, having a cap on complexity, and not taking component interactions into account are some of the restrictions.

Using a Basic Genetic Algorithm (BGA), Sahoo, R. K., et al. [36] suggested a method for improving test data generated from UML activity and state chart diagrams. To produce test data that validates system requirements is the goal. The automated creation of specification-centered test data from UML models is the contribution. The process entails integrating diagrams into intermediate graphical representations, then optimising them using BGA. The paper examines findings and offers a case study. The application to particular diagram kinds, reliance on input quality and BGA settings, and absence of comparison analysis are some limitations.

J. Mburu, J. G. Ndia, & Co. [37] provided a comprehensive mapping research on approaches for creating and optimising test cases based on UML models. Finding trends and gaps in the study is the goal. A comprehensive literature review of publications published between 2010 and 2022 is part of the process. Contributions include an overview of trends and gaps, a list of often used search strategies, a need for greater study on combinational UML diagrams and optimisation, and a list of frequently used validation strategies. The focus on test case optimisation, automation, and combinational UML diagrams is restricted, and the assessment of research is also limited.

Prakash, V. C., Tatale, S. A method for automatically creating combinatorial test cases from UML Activity Diagrams was proposed in [38]. The key contribution is the creation of a programme that takes input parameters from UML Activity Diagrams and use the Particle Swarm Optimisation technique to produce the optimal number of test cases. To show how well the method works in practise, a case study of the Indian Railway Reservation System is presented. The technique uses the Particle Swarm Optimisation algorithm for test case generation, constraint detection, and parameter extraction. To reduce time and effort, test case generation is automated. Application to UML Activity Diagrams only, possible efficacy restrictions in large systems, and appropriateness for certain testing situations are some limits.

A technique for producing scenario-based test cases from UML-ADs ("UML activity diagrams") was put out by Hettab, A., et al. in [39]. The primary contribution enables early testing in the software development life cycle by automatically generating test cases from UML-ADs. According to the technique, test scenarios are derived from EADG models by creating an EADG ("extended activity dependency graph") from UML-ADs. By employing graphical simulation to apply the test scenarios to UML-ADs, testers may verify the test scenarios. Through mutation analysis, the method's capacity to discover faults is assessed. Application to UML-ADs alone, possible drawbacks in big and complicated systems, and reliance on the correctness of UML-ADs are some constraints.

At the beginning of software development, Tamizharasi, A., and Ezhumalai, P. [40] presented a technique for producing optimised test data from UML models using the Hybrid GBCSA ("Genetic and Crow Search Algorithm"). The contribution consists of using GBCSA to streamline the test suite by eliminating unnecessary test data and focusing the search on global optima. Comparing experimental results to conventional crow search and genetic optimisation methods, they show 100% route coverage and time efficiency. To assess the suggested technique, UML models are used, along with experimentation. The goal is to show the method's efficacy and tackle the problem of producing test data for intricate software systems. The absence of a thorough examination of constraints, comparison with alternative approaches, and scalability for big systems are among the drawbacks.

Based on various coverage requirements, Pradhan, S., et al. [41] suggested methods for creating test cases from a state chart diagram. By addressing an object's states and transitions, the aim is to spot state-based defects. In order to build test cases based on various coverage requirements, the process entails converting the state chart diagram into a SCIG ("State Chart Intermediate Graph"). In this study, efficient state-based criterion algorithms like RTP ("Round Trip Path") and ATP ("All Transition Pair") are introduced. There includes discussion of the two case studies, stack operation and vending machine automation system. The algorithmic suggestions for creating test cases based on coverage requirements constitute the contribution. The restriction is ATP's inability to provide complete transition coverage.

Hammad, M., and Hamza, Z. A. [42] gave a case study on the creation of test sequences, concentrating on the usage of a previously suggested method based on use case model analysis. The process entails dissecting the UML use case diagram, transforming it into activity diagrams, simplification, and information extraction. The purpose of the study is to show how well this method works at producing test sequences for software testing. The method's shortcomings, however, include its restricted application to UML use case models, reliance on a single case study, and lack of a comparison with other methodologies already in use.

Sumalatha, V., & Raju, G. S. V. P. [43] used UML activity diagrams to solve the problem of test case creation in model-driven software development. For the purposes of test case creation, reduction, and prioritisation, the authors suggest using Evolutionary and Greedy Heuristic algorithms. The authors of the study compare their methodology to current practises and use UML activity diagrams in an effort to increase the efficacy of software testing. The examination of a single case study, the absence of in-depth comparisons with other methodologies, and algorithm constraints are some of the shortcomings, though.

An innovative method for creating and refining test cases from UML design diagrams was put forth by Khurana, N., et al. [44]. The SYTG ("system graph"), which is explored using a Genetic Algorithm, is created by combining use case, activity, and sequence diagrams. Its limitations include the fact that it only applies to UML design diagrams, how dependent it is on the quality of the input diagrams, and how well suited it is for big and complicated systems.

Using a hybrid bee colony approach, Sahoo, R. K., et al. [45] described a method for creating and refining test cases from combinational UML diagrams. The contribution is an automated test case generation method that aims to make software testing more effective and affordable. Although the technique is model-driven testing-based, it has several drawbacks, such as its emphasis on combinatorial UML diagrams and possible customisation needs for particular software systems.

Chandra and Meiliana, L. C. D. [46] A approach for automatically creating and improving test cases in software testing was put out by [46]. The goal is to use resources more efficiently, especially in the field of mobile technology. Utilising a genetic algorithm, the process entails creating test cases from combinational UML diagrams. Despite offering a beneficial technique, the study has certain drawbacks, such as a small population size and few genetic algorithm operators.

Tiwari, H. Swathi, B. [47] A method for creating test cases for web applications utilising input values and data dependencies was put forth in [47]. Pairwise testing, a genetic algorithm, and a system graph are all components of the process. The contribution of the research is the suggested approach for online applications, which addresses the challenging task of test case creation. The emphasis on functional testing and the absence of other testing kinds, such as security testing, are drawbacks, though.

S. S. Panigrahi. [48] suggested a technique for automatically creating test cases that makes use of a hybrid firefly algorithm and UML Activity diagrams. The strategy focuses on choosing the best test cases in terms of cost and coverage. The case study of an ATM withdrawal strategy is presented in the paper to illustrate the viability of the suggested approach. The objective is to lessen the amount of effort and time needed for software testing. However, drawbacks include the lack of a comparison with alternative approaches and the evaluation's constrained scope, which was only the ATM withdrawal system case study. The contribution. Methodology, merits and limits of these contemporary models have been listed in table 3.1 for quick view.

In table 3.1 there are some research gaps in the use of UML diagrams for test case generation, optimization, and prioritization across multiple domains. One recurring theme is that people tend to focus on specific UML diagrams, such as sequence and activity diagrams, rather than exploring other types of diagrams. An important issue is that there are few empirical studies comparing the proposed methods to existing methods. This lack of comparative data makes it more difficult to demonstrate the effectiveness of new methods and determine how they can be applied in various software development scenarios.

In future research, it may be beneficial to look beyond simple UML diagrams and include modeling languages that are more diverse and complex. This could result in deeper insights and stronger testing frameworks. Furthermore, future research should focus on developing real-world studies that not only compare different test case generation methods, but also assess their effectiveness and scalability. Filling in these gaps can aid in the development of more complex and useful test case strategies, resulting in improvements in automated testing that can keep up with changing software system requirements.

Table 3.1: Summary of contemporary models on test case generation, optimization, and prioritization.

Author	**Contribution**	**Methodology**	**Merits**	**Limits**
B. N. Biswal [6]	Technique for creating and refining test cases of object-oriented software	UML activity diagrams, sequence diagrams, class diagrams	Error minimization method for test case optimization	Application limited to particular categories of object-oriented software, reliance on the excellence of UML behavioural models
Meena, D.K. [7]	Technique for creating test cases using UML diagrams	Interaction Overview diagrams, Sequence diagrams	Message route coverage	Limited to object-oriented programmes, absence of actual data contrasting the suggested strategy with other ones
Jiang, L., and Li, Y. [8]	Technique for creating test cases using UML sequence diagrams	Examination of sequence diagram, message sequence assessment	Useful method for testing object-oriented applications	No prototype system for autonomous test case generation, no discussion of drawbacks or viability of the suggested approach in practical contexts
A. Herrmann, M. Felderer, and M. [9]	Instructions for methodically generating test cases from state machines and activity diagrams	Student experiment, controlled methodology	Taxonomy of faults, evidence of activity diagrams being more error-prone, error maminimization	Utilization of student participation, emphasis on UML activity diagrams and state machines
Jena, A. K., et al. [10]	Approach for creating test cases using UML sequence diagrams and interaction overview diagrams	Combination of produced graphs from UML components, error identification	Early error detection, software quality improvement	Emphasis on a single case study, lack of comparison with alternative techniques for creating test cases
A. Herrmann, M. Felderer, and M. [11]	Controlled experiment examining mistakes in manual test case derivation	Student experiment, taxonomy of faults, evidence of activity diagrams being error-prone	Error minimization, recommendations for minimizing errors	Concentration on certain diagram kinds without comparison to other types or automated test case derivation techniques, relative smallness of sample size
Jagtap, S., et al. [12]	Technique for creating test cases using state chart diagrams and use case diagrams	Testing criteria addressing state and transition in the diagrams	Creation of efficient test cases during early phases of software development	Emphasis on use case diagrams and state chart diagrams in UML 2.0, potential need for manual testing to ensure thorough coverage
Yin, Y., et al. [13]	Automated test case creation method using unstructured SysML ADs	Conversion of ADs into Intermediate Black Box Model, test case generation	Efficient testing methodology for industry practitioners employing SysML Ads	Concentration on the AD model alone, without considering other SysML models
Salman, Y. D., et al. [14]	Efficient combination of coverage requirements in test case generation	Emphasis on UML state chart diagrams, handling loops	Increased efficiency in test data generation during software testing	Emphasis on UML state chart diagrams, requirement for context adaption
Efendi, N. B. M., et al. [15]	Automated test case development using UML Statechart and Sequence diagrams	Utilization of UML diagrams for automatic test case creation	Cost reduction in software development	Emphasis on UML Statechart and Sequence diagrams at the expense of other
Prakash, V. C., et al. [16]	Tool for automated test case development using Particle Swarm Optimization (PSO)	Comprehensive assessment of PSO and its variations	Reduction of test suite sizes, boost in program dependability	Emphasis on PSO and its comparison to other optimization methods

Continued on next page...

Table 3.1 – continued from previous page.

Author	Contribution	Methodology	Merits	Limits
Kamonsantiroj, S., et al. [17]	Memorizing strategy for producing test cases in concurrent UML activity diagrams	Dynamic programming technique, tester specification	Reduction of explosion of test cases in concurrent systems	Application limited to concurrent UML activity diagrams, emphasis on concurrency testing, limitations on other diagram types or concurrent systems
Mokhati, F.; Dehimi, N. E. H. [18]	Evaluation method for multi-agent systems using AUML sequence diagrams	Parallel, exclusive, or inclusive interactions, OCL restrictions	Testing strategy ensuring mistake independence	Only applies to holonic agents, absence of empirical proof of the efficacy of the suggested strategy
Dawood, Y. S., and Hashim, N. L. [19]	Evaluation of test case creation using UML statechart diagrams	Content analysis of 24 books, investigation of algorithms and coverage	Identification of discrepancies, improved understanding of test case generation procedures	Emphasis on UML statechart diagrams, limited treatment of prior works
Alrawashed, T. A., et al. [20]	Test case generation from software requirements at the use case description level	Fine-tuning of use cases, transformation into control flow diagrams	More effective regression testing, reduced test complexity and improved coverage	Absence of comparison with current methods, lack of thorough evaluation of case study findings
Khalifa, E. M., et al. [21]	Test case generation from use case diagrams using machine learning	Metaheuristic approach, information extraction, preprocessing	Automation of laborious test case generation process, increased precision and effectiveness	Application limited to certain software systems, reliance on the quality of use case diagrams
Runal, G., et al. [22]	Creation of optimized test cases from UML sequence diagrams using the Firefly algorithm	Model-based testing, structural test selection concept	Automation of test case creation, improved quality and dependability of software systems	Emphasis on UML sequence diagrams, potential limitations in big and complicated systems
Swain, R., et al. [23]	Automated creation of executable test scripts for an IoT system using UML state machine diagrams	Creation of mappings, algorithm for transition pathway creation	Simplification of testing process, reduced manual labor	Absence of transition guard evaluation, potential for method improvement through symbolic evaluation
J. Cvetkovi and M. Cvetkovi. [3]	Creation of test cases with UML diagrams focusing on modelling depression caused by internet addiction	Categorization of test case creation, utilization of various UML diagrams	Advancement in software testing, fresh method for creating test cases	Specificity of the case study, absence of comparison with other approaches
ALI, H. M. B. M. [24]	Improvement of test case generation from UML sequence diagrams	Enhanced method for quick and easy test case creation	Quick and easy approach to create test cases	Emphasis on looping and iteration issues, need for knowledge of UML sequence diagrams
Kikuma, K., et al. [25]	Automation of software development processes in telecom carrier networks	Mathematical principles, preparation of test cases, use of mathematics	Cost savings, dependability, safety	Data creation limited to already-existing design documentation
Rocha, M., et al. [26]	Methodical process for creating test cases from UML model, specifically UML Sequence Diagrams	ModelJUnit, JUnit libraries, transformation rules	Defined meaning for UML Sequence Diagrams, automation of testing process	Emphasis on UML Sequence Diagrams, efficacy relies on the calibre of the UML model
Tiwari, H. Swathi, B. [27]	Soft computing approaches for test case creation in software testing	Genetic algorithms, artificial bee colonies	Assessment criterion, exploration of soft computing approaches	Emphasis on soft computing methods, constrained scope of empirical study

Continued on next page...

Table 3.1 – continued from previous page.

Author	**Contribution**	**Methodology**	**Merits**	**Limits**
Soni, D. Jain, P. [28]	Survey on methods for prioritizing and generating test cases using UML diagrams	Analysis of relevant work, comparison of methods	Identification of knowledge gaps, overview of test case creation techniques using UML diagrams	Emphasis on UML diagrams, lack of thorough analysis and useful implementation
Minhas, N. M., et al. [29]	Mapping of test case generating approaches using UML interaction diagrams	Evaluation of reporting quality, identification of errors	Comparison of methods based on strengths and weaknesses	Lack of empirical evaluation in industry contexts, lack of conformity to research principles
Mburu, J. M., et al. [30]	Improved multiview test case generating approach for object-oriented software using UML diagrams	MUTCASGenerator technology	Automation of test case creation, inclusion of structural and behavioral aspects of the system	Limited to UML models, potential issues in complicated systems, reliance on the calibre of UML models
Dash and Panda, M. [31]	Overview of model-based and search-based testing strategies for object-oriented programs	Framework for search-based testing, hybrid metaheuristics algorithms	Compilation of data on testing procedures and approaches, insights into various testing methodologies	Lack of comparative analysis, lack of empirical backing
S. B. Tatale, V. C. Prakash, & Co. [32]	Feasibility study on producing test cases focused on combinatorial logic using UML diagrams	Dynamic slicing techniques, conversion of UML diagrams	Automatic generation of test cases from UML Sequence and Activity diagrams	Emphasis only on UML Sequence and Activity diagrams, lack of consideration for other UML diagrams
K. Jin, K. Lano, and K. [33]	Systematic literature review on creating test cases from UML diagrams	Systematic literature review	Identification of methods, results, research trends, gaps, and suggestions for future study	Focus only on UML diagrams without considering other modelling languages
Ali, A., et al. [34]	Autonomous model-based test case creation technique for load testing mobile apps	Model-based testing, UML model creation, performance evaluation	Minimization of testing time, validation of requirements for both performance and functional elements	Lack of thorough review, choice of workload
Fan, C., and Zou, P. [35]	Technique for creating test cases for flight control systems using UML state diagrams	Real-time extension strategy, time domain equivalence partition method	Inclusion of real-time aspects, feasibility check of the approach	Applicable only to UML state diagrams, potential issues in complicated systems, no consideration of component interactions
Sahoo, R. K., et al. [36]	Method for improving test data generated from UML activity and state chart diagrams	BGA (“Basic Genetic Algorithm”)	Test data generation that validates system requirements	Application to particular diagram kinds, reliance on input quality and BGA settings, absence of comparison analysis
J. Mburu, J. G. Ndia, & Co. [37]	Mapping research on approaches for creating and optimizing test cases based on UML models	Comprehensive literature review	Overview of trends and gaps, identification of frequently used strategies	Focus on test case optimization, automation, and combinational UML diagrams, limited assessment of research
Prakash, V. C., Tatale, S. [38]	Automatic creation of combinatorial test cases from UML Activity Diagrams	PSO (“Particle Swarm Optimization”) technique, case study	Quick and efficient test case generation from UML Activity Diagrams	Application limited to UML Activity Diagrams, possible efficacy restrictions in large systems, appropriateness for certain testing situations

Continued on next page...

Table 3.1 – continued from previous page.

Author	**Contribution**	**Methodology**	**Merits**	**Limits**
Hettab, A., et al. [39]	Technique for producing scenario-based test cases from UML activity diagrams	EADG, graphical simulation	Early testing in software development, automatic generation of test cases from UML activity diagrams	Application limited to UML activity diagrams, possible drawbacks in big and complicated systems, reliance on the correctness of UML activity diagrams
Tamizharasi, A., and Ezhumalai, P. [40]	Producing optimised test data from UML models using the Hybrid ("GBCSA")	Hybrid GBCSA ("Genetic and Crow Search Algorithm")	Streamlining the test suite, elimination of unnecessary test data	Absence of thorough examination of constraints, comparison with alternative approaches, scalability for big systems
Pradhan, S., et al. [41]	Methods for creating test cases from a state chart diagram	Conversion of state chart diagram into SCIG ("State Chart Intermediate Graph")	Spotting state-based defects, algorithmic suggestions for test case creation	Inability of ATP to provide complete transition coverage
Hammad, M., and Hamza, Z. A. [42]	Case study on the creation of test sequences using use case model analysis	UML use case diagram analysis, transformation into activity diagrams	Production of test sequences for software testing	Restricted application to UML use case models, reliance on a single case study, lack of comparison with other methodologies
Sumalatha, V., & Raju, G. S. V. P. [43]	Test case creation, reduction, and prioritisation using UML activity diagrams	Evolutionary and Greedy Heuristic algorithms	Increased efficacy of software testing using UML activity diagrams	Examination limited to a single case study, absence of in-depth comparisons with other methodologies, algorithm constraints
Khurana, N., et al. [44]	Creating and refining test cases from UML design diagrams	Genetic Algorithm, creation of ("SYTG")	Combination of different UML design diagrams, refinement of test cases	Application limited to UML design diagrams, dependence on input diagram quality, suitability for big and complex systems
Sahoo, R. K., et al. [45]	Creating and refining test cases from combinational UML diagrams	Hybrid bee colony approach, model-driven testing	Automated test case generation, improved software testing	Emphasis on combinational UML diagrams, customization needs for specific software systems
Chandra and Meiliana, L. C. D. [46]	Automatic creation and improvement of test cases in software testing	Genetic Algorithm, creation of test cases from combinational UML diagrams	Efficient resource utilization, application in the mobile technology field	Small population size, limited genetic algorithm operators
Tiwari, H. Swathi, B. [47]	Creation of test cases for web applications using input values and data dependencies	Pairwise testing, genetic algorithm, system graph	Approach for online applications, addressing the challenge of test case creation	Emphasis on functional testing, absence of other testing types
S. S. Panigrahi [48]	Automatic creation of test cases using a hybrid firefly algorithm and UML Activity diagrams	Hybrid firefly algorithm, UML Activity diagrams	Reduction in effort and time for software testing	Lack of comparison with alternative approaches, evaluation limited to a specific case study

3.2. Test Case Prioritization. Regression testing test case prioritisation method employing sequence diagrams and labelled transition systems was suggested by As' Sahra, N. F., & Komputeran, F. [49]. The method use Bayesian Networks to incorporate source code modifications, software fault-proneness, as well as test coverage data into a single model. However, the research makes an assumption about test case independence that could not hold true in actual circumstances.

A novel method of test case prioritisation for model-based mutation testing in the automotive sector was introduced by Shin, K. W., and Lim, D. J. [50]. They use the UML statechart to create a software model, use mutation operators to generate mutations, and suggest a TCP technique based on the (”alternating variable method”) AVM. The study covers actual research in the automobile sector, however its application outside the sector and the quantity of investigated problems are also limits.

The use of machine learning approaches in ”test case prioritization” (TCP) for regression testing was examined by Meçe, E. K., Paci, and Binjaku [51]. They review current research that employs machine learning in TCP and provide details on methods, measurements, data, benefits, and drawbacks. There are drawbacks, such as the potential rejection of important test cases via selection approaches and the requirement for a significant quantity of data for efficient machine learning methods.

An autonomous test route generating method and prioritisation model for software testing were suggested by Fan, L., Wang, Y., and Liu [52]. They build a priority model to order the test pathways, design the activity flow graph, and specify the mapping rules from UML activity diagram to the graph. The article does not, however, compare new algorithms to old ones or evaluate large-scale software systems. It also presupposes that UML activity diagrams are accessible.

For user interface testing, Nguyen, V., & Le, B. [53] introduced a unique test prioritisation technique called RLTCP. The technique uses the coverage graph and reinforcement learning (RL) to prioritise test cases. With an experimental assessment contrasting it with other approaches, the research builds on a past work on prioritising UI automated test cases using RL. The drawback is that it ignores alternative testing methods in favour of a narrow emphasis on user interface testing.

A test route prioritisation approach based on the testers’ interests, as well as the altered area in UML activity diagrams, was proposed by Sornkliang, W., and Phetkaew, T. [54]. They use various techniques to give weights to symbols and then order test pathways accordingly. The technique tries to aid testers in swiftly identifying critical issues. The dependence on testers’ preferences and the work involved in weight assignment are limitations.

The contribution of the Methodology, merits and limits of these contemporary models have been listed as a table for quick view table 3.2.

Table 3.2 a thorough examination of the most recent models for test case prioritization, several research gaps were identified that could be addressed in future studies. Sahra, N. F., and Komputeran, F. [49] use Bayesian Networks for prioritization, assuming test case independence, which may not be true in practice. Future research could look into models that account for interdependence among test cases. Shin, K. W., and Lim, D. J. [50] use model-based mutation testing exclusively in the automotive industry. This demonstrates the need for mutation testing to be applied in more areas and studied in greater depth. The machine learning approach proposed by Mece, E. K., Paci, and Binjaku [51] shows promise in terms of efficiency, but it may miss important test cases and need a large amount of data. This suggests that we need more reliable machine learning models that do not require as much data.

Fan, L., Wang, Y., and Liu [52] propose an autonomous test route generation method that needs to be compared to traditional algorithms and tested in large-scale systems. This opens the door for future research to confirm its usefulness and potential for expansion. The RL-based method developed by Nguyen, V., and Le, B. [53] for testing user interfaces is novel and intriguing, but it only tests a few things. This demonstrates that reinforcement learning could be applied in a broader range of testing scenarios. Finally, Sornkliang, W., and Phetkaew, T. [54] interest-based route prioritization heavily relies on tester input to assign weights. This demonstrates the need for more automated, objective prioritization frameworks with less human bias and effort. Filling these identified gaps would significantly improve the development of test case prioritization methods, making regression testing more useful.

4. Observations. The review of the articles [6–15] highlights various techniques and approaches for automated test case generation using UML behavioral models and diagrams in object-oriented software development. Each article focuses on specific UML models or diagrams and has some limitations. The most effective and efficient automated test case generation method that makes use of UML behavioral models and diagrams needs to be determined in order to address these limitations and provide guidance to practitioners. The identification of best practices for automated test case generation in object-oriented software development will be made possible by this research question, which will also allow for a thorough examination of current approaches, comparison of their effectiveness and efficiency.

Table 3.2: Summary of contemporary models on test case prioritization

Author	**Contribution**	**Methodology**	**Merits**	**Limits**
As' Sahra, N. F., & Komputeran, F. [49]	Regression testing test case prioritisation method employing sequence diagrams and labelled transition systems	Bayesian Networks	Incorporation of source code modifications, fault-proneness, and test coverage	Assumption of test case independence that may not hold true in actual circumstances
Shin, K. W., and Lim, D. J. [50]	Test case prioritisation for model-based mutation testing in the automotive sector	UML statechart, mutation operators, TCP technique based on AVM	Application in the automotive sector, software model creation	Limited application outside the automotive sector, limited investigation of problems
Meçe, E. K., Paci, and Binjaku [51]	Use of machine learning approaches in test case prioritization for regression testing	Review of existing research, analysis of methods, measurements, data	Potential for efficient machine learning methods, insights into benefits	Potential rejection of important test cases, requirement for a significant amount of data
Fan, L., Wang, Y., and Liu [52]	Autonomous test route generating method and prioritisation model for software testing	Priority model, activity flow graph, mapping rules from UML activity diagram	Autonomous test route generation, prioritisation of test pathways	Lack of comparison with old algorithms, lack of evaluation on large-scale software systems, assumption of accessibility of UML activity diagrams
Nguyen, V., & Le, B. [53]	Test prioritisation technique called RLTCP for user interface testing	Coverage graph, RL ("reinforcement learning")	Prioritisation of test cases in user interface testing	Narrow emphasis on user interface testing, ignoring alternative testing methods
Sornkliang, W., and Phetkaew, T. [54]	Test route prioritisation approach based on testers' interests and altered area in UML activity diagrams	Weight assignment techniques, ordering of test pathways based on weights	Aid in quickly identifying critical issues, prioritisation based on interests	Dependence on testers' preferences, effort required for weight assignment

The review of the articles [16–24] highlights different approaches for automated test case generation that use optimization techniques and UML diagrams. The article focuses on specific optimization techniques or UML diagram types and has some limitations. In order to address these limitations and offer guidance for practitioners, identify the most effective and efficient approaches for automated test case generation that combine optimization techniques with different kinds of UML diagrams. This research question will enable a thorough examination of current approaches, comparison of their effectiveness, and identification of best practices for automated test case generation in software development.

The review of the articles [25–32], [1], and [5] highlights different approaches and methodologies for automated test case generation using UML diagrams. Each article focuses on specific optimization techniques, soft computing, or UML diagrams and has its own limitations. In order to address these limitations and offer thorough recommendations for software development, it is important to investigate how various methodologies and techniques can be effectively combined to improve test case generation using UML diagrams. In order to enhance the effectiveness and efficiency of test case generation and software development, this research question will enable an exploration of optimization methods, soft computing techniques, and automation approaches.

Reasoning the review of the articles [33–41] reveals limitations and opportunities for automated test case generation using UML diagrams. Limitations include a lack of comparative analysis or thorough evaluation, a focus on specific diagram types, and applicability to complex systems. To overcome these limitations and enhance the effectiveness of test case generation, look into how automated techniques can be improved to address the identified challenges. This research question will examine the exploration of factors such as model-based testing, optimization techniques, and validation approaches to enhance the applicability, efficiency, and effectiveness of automated test case generation using UML diagrams. By addressing these limitations, software developers can enhance test case generation to support more complex systems, stronger comparative analysis, and evaluation.

The review of the articles [42–46] highlights common limitations in current approaches, including a focus on specific diagram types, reliance on input diagram quality, customization demands, and limitations in algorithm design. The applicability and effectiveness of the suggested methods are restricted by these limitations. More research is needed to address these limitations, create better approaches that can be used more broadly, improve software testing effectiveness, and maximize test case generation and prioritization. By looking into and addressing these limitations, researchers can promote model-driven software development and test methodologies.

The review of the papers [47–54] highlights several limitations in the current test case prioritization techniques. These limitations include limited applicability outside of case domains, omission of crucial test cases, lack of comparison with current algorithms, reliance on a specific type of testing, and reliance on testers' interests. It is important to investigate and suggest improvements to current techniques in order to address these limitations and enhance the effectiveness and efficiency of test case prioritization. Considering the limitations of the reviewed papers, this research question allows for the exploration of potential solutions and advancements in test case prioritization.

5. Implications and possible future research. Software testing researchers and practitioners can benefit from a thorough literature review on the topic of creating test cases from UML diagrams in a number of ways. These ramifications point to potential directions for future investigation.

1. Research Gaps and Deficiencies: The review points out research gaps and deficiencies in the literature. By highlighting areas that haven't been addressed, these gaps can guide future research efforts. Filling these gaps can be the focus of empirical studies, novel methodologies, and alternative approaches to test case generation from UML diagrams.
2. Comparative Evaluation: The review demonstrates the dearth of empirical support and thorough comparative evaluations in many studies. Future research may highlight the necessity of conducting thorough comparative analyses to assess the effectiveness, efficiency, and applicability of different test case generation techniques. Comparative studies assist researchers and practitioners in selecting the most appropriate methodology for their specific requirements.
3. Generalizability and Applicability: Identify and discuss the applicability and generalizability limitations of the suggested approaches. Research should focus on creating techniques that can handle various types of UML diagrams rather than just specific diagram types like activity and design diagrams. In different software development contexts, look into ways to scale and make the approaches effective.
4. Algorithmic Improvements: Deal with the algorithmic elements' limitations of the suggested approaches. Future research should improve the methods' heuristic and evolutionary algorithms. Investigate advanced optimization techniques, optimize algorithm parameters, or develop hybrid approaches to improve the effectiveness and efficiency of test case generation and optimization.
5. Quality Assurance of UML Diagrams: Examine techniques to enhance the accuracy and coherence of UML diagrams used for test case generation and optimization. To ensure that diagrams are accurate and complete, develop automated validation and verification techniques. Investigate techniques to improve UML diagram clarity and interpretability to facilitate test case generation, as well as approaches to handle incomplete or inconsistent diagrams.
6. Integration of Testing Types: Go beyond functional tests and incorporate other test types using the recommended approaches. Security, performance, usability, and other testing types should be considered in test case generation and optimization. This ensures that various quality-related issues are addressed during software testing.
7. Automation and Tool Support: The review highlights the importance of automation and tool support in test case generation from UML diagrams. Future research can focus on automated frameworks and tools that provide effective test case generation and integrate with UML modeling tools. These tools can facilitate UML-based test practices, reduce manual labor, and enhance productivity.
8. Scalability and Complexity: For large and complex software systems, many studies overlook the scalability of test case generation techniques. Future research can address these issues because modern software systems are larger and more complex than ever. In order to address scalability concerns, system complexity, and resource efficiency during test case generation, this includes looking into techniques.

9. Criteria for Coverage and Optimization: Review highlights the importance of coverage criteria and optimization techniques in test case generation: In order to produce optimized test cases with high coverage and low redundancy, future research can create sophisticated optimization algorithms, hybrid approaches, and intelligent techniques. Another important area of research is the investigation of novel coverage criteria and the assessment of their effectiveness in detecting different types of faults.
10. Other Languages for Modeling and Integration: While other software engineering modeling languages are explored, UML diagrams are the main focus of the review. Future research can examine test case generation techniques for these alternative modeling languages in an effort to enhance the effectiveness and efficiency of test case modeling.
11. Real-world Evaluation: Several reviewed studies lacked adequate validation and assessment. In real-world software development projects, test case generation techniques can be applied and evaluated as a research topic. The proposed methods may be assessed for effectiveness and applicability by incorporating realistic software systems, industry partnerships, and case studies from diverse fields.

5.1. Test case Generation and Optimization.

- Utilizing evolutionary and heuristic algorithms, along with UML diagrams, can improve test case generation and prioritization.
- Comprehensive models, like System Graphs, offer a holistic approach to test case generation and optimization.
- Hybrid optimization algorithms and model-driven testing methodologies show promise for automated and cost-effective software testing.
- Evaluating and comparing the performance of different algorithms and approaches for test case generation and prioritization.
- Investigating the applicability of proposed techniques to various types of UML diagrams and software systems.
- Addressing limitations related to input diagram quality, scalability, customization requirements, and the need for flexible solutions.
- Assessing the effectiveness and efficiency of hybrid optimization algorithms in real-world scenarios and comparing them to alternative approaches.
- Exploring the impact of population size, operators, and optimization techniques on test case generation and optimization.
- Extending solutions to address various testing scenarios, assessing their effectiveness across various applications, and managing the growing complexity of software systems.

- For test case prioritization techniques to be effective in practical settings, test case dependencies and relationships must be taken into account.
- Integrating multiple factors, such as source code changes, fault-proneness, and test coverage data, into unified models shows potential for enhancing test case prioritization.
- Machine learning techniques offer improved efficiency but require careful consideration of data availability and the potential exclusion of critical test cases.
- To develop test case prioritization techniques that are capable of handling dependencies and inter-test case relationships, more research is needed.
- Investigate ways to incorporate metrics and factors into prioritization models to increase their thoroughness and accuracy.
- Look into alternative data sources that can offer trustworthy information for prioritization based on machine learning, or investigate ways to lessen the reliance on large amounts of data.
- Evaluate the suggested techniques' generalizability and scalability on complex software systems.
- To develop versatile techniques, broaden your research beyond particular domains or testing types.
- Consider ways to enhance or automate weight assignment for more effective test case prioritization. By addressing these implications and exploring the suggested future research directions, researchers can advance the field of test case generation from UML diagrams, improve the efficiency and effectiveness of software testing practices, and contribute to the development of reliable and high-quality software systems.

6. Conclusion. The current state of research in this field is clarified by the systematic literature review on creating, optimizing, and prioritizing test cases from UML diagrams. Along with implications and future research directions, we review research trends, approaches, limitations, and gaps. Several important areas for additional research are highlighted in the review. The development of automated tools and frameworks for seamless integration with UML modeling tools, scalability and complexity challenges in large and complex software systems, advanced optimization algorithms and coverage criteria to generate optimized test cases, and investigation of test case generation techniques for alternative modeling languages are a few examples. The field of test case generation from UML diagrams can progress by addressing these implications and following recommended future case research directions. The effectiveness and efficiency of software test case generation techniques can be improved, and researchers can offer testers useful solutions. To aid in the development of dependable and superior software systems, they can decrease manual labor, increase automation, and enhance test procedures. The researchers and practitioners in the field of software testing gain from this systematic review of the literature. It discusses current methods, makes recommendations for advancements, and establishes the foundation for future research. The software development industry will benefit from researchers' use of the insights from this review to enhance test case generation from UML diagrams.

REFERENCES

[1] R. G. Tiwari, et al., *Exploiting UML Diagrams for Test Case Generation: A Review*, 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), IEEE, 2021, pp. 457–460.

[2] C. Mingsong, Q. Xiaokang, and L. Xuandong, *Automatic test case generation for UML activity diagrams*, Proceedings of the 2006 International Workshop on Automation of Software Test, 2006, pp. 2–8.

[3] J. Cvetković and M. Cvetković, *Evaluation of UML Diagrams for Test Cases Generation: Case Study on Depression of Internet Addiction*, Physica A: Statistical Mechanics and Its Applications, vol. 525, 2019, pp. 1351–1359.

[4] K. Jin and K. Lano, *Generation of test cases from UML diagrams—A systematic literature review*, 14th Innovations in Software Engineering Conference (Formerly Known as India Software Engineering Conference), 2021, pp. 1–10.

[5] M. Rocha, et al., *Model-Based Test Case Generation from UML Sequence Diagrams Using Extended Finite State Machines*, Software Quality Journal, vol. 29, no. 3, 2021, pp. 597–627.

[6] B. N. Biswal, *Test case generation and optimization of object-oriented software using UML behavioral models*, Diss. 2010.

[7] D. K. Meena, *Test Case Generation From UML Interaction Overview Diagram and Sequence Diagram*, Diss. 2013.

[8] Y. Li and L. Jiang, *The Research on Test Case Generation Technology of UML Sequence Diagram*, 2014 9th International Conference on Computer Science & Education, IEEE, 2014, pp. 1067–1069.

[9] M. Felderer and A. Herrmann, *Manual Test Case Derivation from UML Activity Diagrams and State Machines: A Controlled Experiment*, Information and Software Technology, vol. 61, 2015, pp. 1–15.

[10] A. K. Jena, S. K. Swain, and D. P. Mohapatra, *Model based test case generation from UMLsequence and interaction overview diagrams*, Computational Intelligence in Data Mining-Volume 2: Proceedings of the International Conference on CIDM, 20-21 December 2014. Springer India, 2015, pp. 247-257.

[11] M. Felderer and A. Herrmann, *Comprehensibility of System Models during Test Design: A Controlled Experiment Comparing UML Activity Diagrams and State Machines*, Software Quality Journal, vol. 27, no. 1, 2019, pp. 125–147.

[12] S. Jagtap, et al., *Generate Test Cases From UML Use Case and State Chart Diagrams*, International Research Journal of Engineering and Technology (IRJET), vol. 3, no. 10, 2016, pp. 873–881.

[13] Y. Yin, *An Automated Test Case Generation Approach Based on Activity Diagrams of SysML*, International Journal of Performability Engineering, 2017.

[14] Y. D. Salman, et al., *Coverage Criteria for Test Case Generation Using UML State Chart Diagram*, p. 020125. DOI.org (Crossref), https://doi.org/10.1063/1.5005458.

[15] N. B. M. Efendi and H. Asmuni, *Exhaustive Search for Test Case Generation from UML Sequence Diagram and Statechart Diagram*, UTM Computing Proceedings Innovation in Computing Technology and Applications, Vol.2, 2018.

[16] V. C. Prakash, et al., *A Critical Review on Automated Test Case Generation for Conducting Combinatorial Testing Using Particle Swarm Optimization*, International Journal of Engineering & Technology, vol. 7, no. 3.8, 2018, p. 22.

[17] S. Kamonsantiroj, et al., *A Memorization Approach for Test Case Generation in Concurrent UML Activity Diagram*, Proceedings of the 2019 2nd International Conference on Geoinformatics and Data Analysis, ACM, 2019, pp. 20–25.

[18] N. E. H. Dehimi and F. Mokhati, *A Novel Test Case Generation Approach Based on AUML Sequence Diagram*, 2019 International Conference on Networking and Advanced Systems (ICNAS), IEEE, 2019, pp. 1–4.

[19] N. L. Hashim and Y. S. Dawood, *A Review on Test Case Generation Methods Using UML Statechart*, 2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE), IEEE, 2019, pp. 1–5.

[20] T. A. Alrawashed, et al., *An Automated Approach to Generate Test Cases From Use Case Description Model*, Computer Modeling in Engineering & Sciences, vol. 119, no. 3, 2019, pp. 409–425.

[21] E. M. Khalifa, D. Jawawi, and H. A. Jamil, *An efficient method to generate test cases from UML-use case diagram*, International Journal of Engineering Research and Technology, vol. 12, no. 7, 2019, pp. 1138–1145.

[22] G. Runal, et al., *FUNCTIONAL TEST CASE GENERATION AND REDUNDANCY REMOVAL BASED ON MODEL DRIVEN TESTING USING UML ACTIVITY DIAGRAM*, International Journal of Mechanical Engineering and Technology (IJMET), vol. 10, no. 05, May 2019, pp. 318–324.

[23] R. Swain, et al., *Automatic Test Case Generation From UML State Chart Diagram*, International Journal of Computer Applications, vol. 42, no. 7, Mar. 2012, pp. 26–36.

[24] H. M. B. M. ALI, *MODEL-BASED SEMI-AUTOMATED TEST CASE GENERATION APPROACH USING UML DIAGRAMS*, 2019.

[25] K. Kikuma, et al., *Preparation Method in Automated Test Case Generation Using Machine Learning*, Proceedings of the Tenth International Symposium on Information and Communication Technology - SoICT 2019, ACM Press, 2019, pp. 393–398.

[26] M. Rocha, et al., *Test Case Generation by EFSM Extracted from UML Sequence Diagrams*, 2019, pp. 135–140.

[27] B. Swathi and H. Tiwari, *Test Case Generation Process Using Soft Computing Techniques*, International Journal of Innovative Technology and Exploring Engineering, vol. 9, no. 1, Nov. 2019, pp. 4824–4831.

[28] P. Jain and D. Soni, *A Survey on Generation of Test Cases Using UML Diagrams*, 2020 International Conference on Emerging Trends in Information Technology and Engineering (Ic-ETITE), IEEE, 2020, pp. 1–6.

[29] N. M. Minhas, et al., *A Systematic Mapping of Test Case Generation Techniques Using UML Interaction Diagrams*, Journal of Software: Evolution and Process, vol. 32, no. 6, June 2020.

[30] J. M. Mburu, G. M. Muketha, and A. M. Oirere, *An Enhanced Multiview Test Case Generation Technique for Object-Oriented Software Using Class and Activity Diagrams*, International Journal of Recent Technology and Engineering (IJRTE), vol. 9, no. 4, Nov. 2020, pp. 185–196.

[31] M. Panda and S. Dash, *Test-case generation for model-based testing of object-oriented programs*, Automated Software Testing: Foundations, Applications and Challenges, 15, 2020, pp. 53–77.

[32] S. B. Tatale and V. C. Prakash, *A Survey on Test Case Generation using UML Diagrams and Feasibility Study to Generate Combinatorial Logic Oriented Test Cases*, International Journal of Next-Generation Computing, vol. 12, no. 2, 2021, pp. 254–269.

[33] K. Jin and K. Lano, *Generation of Test Cases from UML Diagrams - A Systematic Literature Review*, 14th Innovations in Software Engineering Conference (Formerly Known as India Software Engineering Conference), ACM, 2021, pp. 1–10.

[34] A. Ali, et al., *Model-Based Test Case Generation Approach for Mobile Applications Load Testing Using OCL Enhanced Activity Diagrams*, 2021 Tenth International Conference on Intelligent Computing and Information Systems (ICICIS), IEEE, 2021, pp. 493–499.

[35] C. Fan and P. Zou, *Research on Automatic Test Case Generation Method of Flight Control System Based on UML State Diagram*, Journal of Physics: Conference Series, vol. 1961, no. 1, July 2021, p. 012019.

[36] R. K. Sahoo, et al., *Test Case Generation from UML-Diagrams Using Genetic Algorithm*, Computers, Materials & Continua, vol. 67, no. 2, 2021, pp. 2321–2336.

[37] J. M. Mburu and J. G. Ndia, *A Systematic Mapping Study on UML Model Based Test Case Generation and Optimization Techniques*, International Journal of Computer Applications, vol. 184, no. 13, May 2022, pp. 26–33.

[38] S. Tatale and V. C. Prakash, *Automatic Generation and Optimization of Combinatorial Test Cases from UML Activity Diagram Using Particle Swarm Optimization*, Ingénierie Des Systèmes d'Information, vol. 27, no. 1, Feb. 2022, pp. 49–59.

[39] A. Hettab, et al., *Automatic Scenario-Oriented Test Case Generation from UML Activity Diagrams: A Graph Transformation and Simulation Approach*, International Journal of Computer Aided Engineering and Technology, vol. 16, no. 3, 2022, p. 379.

[40] A. Tamizharasi and P. Ezhumalai, *Genetic-based Crow Search Algorithm for Test Case Generation*, International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies, vol. 13, no. 4, 2022, pp. 1–11.

[41] S. Pradhan, et al., *Transition Coverage Based Test Case Generation from State Chart Diagram*, Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 3, Mar. 2022, pp. 993–1002.

[42] Z. A. Hamza and M. Hammad, *Generating Test Sequences from UML Use Case Diagram: A Case Study*, 2020 Second International Sustainability and Resilience Conference: Technology and Innovation in Building Designs, IEEE, 2020, pp. 1–6.

[43] V. Sumalatha and G. S. V. P. Raju, *Model-based test case optimization of UML activity diagrams using evolutionary algorithms*, Int J Comput Sci Mob Appl, vol. 2, no. 11, 2014, pp. 131–142.

[44] N. Khurana, R. S. Chhillar, and U. Chhillar, *A Novel Technique for Generation and Optimization of Test Cases Using Use Case, Sequence, Activity Diagram and Genetic Algorithm*, Journal of Software, vol. 11, no. 3, 2016, pp. 242–250.

[45] R. K. Sahoo, et al., *Model Driven Test Case Optimization of UML Combinational Diagrams Using Hybrid Bee Colony Algorithm*, International Journal of Intelligent Systems and Applications, vol. 9, no. 6, June 2017, pp. 43–54.

[46] L. C. D. Meiliana and A. Chandra, *Optimization of test case generation from uml Activity diagram and sequence diagram By using genetic algorithm*, vol. 13, no. 07, 2019, pp. 585.

[47] B. Swathi and H. Tiwari, *Integrated Pairwise Testing Based Genetic Algorithm for Test Optimization*, International Journal of Advanced Computer Science and Applications, vol. 12, no. 4, 2021.

[48] S. S. Panigrahi, et al., *Model-Driven Automatic Paths Generation and Test Case Optimization Using Hybrid FA-BC*, 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), IEEE, 2021, pp. 263–268.

[49] N. F. As' Sahra and F. Komputeran, *Test case prioritization technique using sequence diagram and labeled transition systems in regression testing*, Diss. Universiti Teknologi Malaysia, 2015.

[50] K.-W. Shin and D.-J. Lim, *Model-Based Test Case Prioritization Using an Alternating Variable Method for Regression*

Testing of a UML-Based Model, Applied Sciences, vol. 10, no. 21, Oct. 2020, p. 7537.

[51] E. K. Mece, et al., *The Application Of Machine Learning In Test Case Prioritization - A Review*, European Journal of Electrical Engineering and Computer Science, vol. 4, no. 1, Jan. 2020.

[52] L. Fan, et al., *Automatic Test Path Generation and Prioritization Using UML Activity Diagram*, 2021 8th International Conference on Dependable Systems and Their Applications (DSA), IEEE, 2021, pp. 484–490.

[53] V. Nguyen and B. Le, *RLTCP: A Reinforcement Learning Approach to Prioritizing Automated User Interface Tests*, Information and Software Technology, vol. 136, Aug. 2021, p. 106574.

[54] W. Sornkliang and T. Phetkaew, *Target-Based Test Path Prioritization for UML Activity Diagram Using Weight Assignment Methods*, International Journal of Electrical and Computer Engineering (IJECE), vol. 11, no. 1, Feb. 2021, p. 575.

Edited by: Anil Kumar Budati
Special issue on: Soft Computing and Artificial Intelligence for wire/wireless Human-Machine Interface
Received: Dec 11, 2023
Accepted: Apr 26, 2024