Automated Whitebox Test Case Generation for Statement Coverage Using U-NSGA-III

Author
Mehwish Naz
MS-19-00000321001

Supervisor
Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

December 2022

Automated Whitebox Test Case Generation for Statement Coverage Using UNSGAIII

Author:

Mehwish Naz

00000321001

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Software Engineering

Approved by:

____

Supervisor:
Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY

December 2022

DECLARATION

The substance of this thesis is the original work of the author and due references and acknowledgements have been made, where necessary, to the work of others. No part of this thesis has been submitted in candidature of any degree.

____
Ms. Mehwish Naz
00000321001

Countersigned:

____
Dr. Wasi Haider Butt

THESIS SUPERVISOR

PLAGIARISM REPORT (TURNITIN REPORT)

This thesis copy has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

Signature of Student

Mehwish Naz

00000321001

Signature of Supervisor

Dr. Wasi Haider Butt

Language Correctness Certificate

This thesis has been read by an English expert and is free of most typing, syntax, semantic, grammatical, and spelling mistakes. The thesis is set all according to the format given by the university.

Signature of Student

Mehwish Naz

00000321001

Signature of Supervisor

Dr. Wasi Haider Butt

Copyright Statement

- Copyright in text of this thesis rests with the student author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author and lodged in the Library of NUST College of E&ME. Details may be obtained by the Librarian. This page must form part of any such copies made. Further copies (by any process) may not be made without the permission (in writing) of the author.
- The ownership of any intellectual property rights which may be described in this thesis is vested in NUST College of E&ME, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the College of E&ME, which will prescribe the terms and conditions of any such agreement.
- Further information on the conditions under which disclosures and exploitation may take place is available from the Library of NUST College of E&ME, Rawalpindi.

Acknowledgements

I am extremely thankful to ALLAH Almighty for his bountiful blessings throughout this work. It was quite an arduous exercise that could not have been completed without the help of God Almighty and the strength that he bestowed upon me.

I would like to offer my sincere thanks to my incredible supervisor **'Dr. Wasi Haider Butt'** and the entire committee: **'Dr. Arslan Shaukat'** and **'Dr. Ahsan Shehzad'.** Without their support – in every sense of the word – and constant guidance, this thesis would never have been completed. I cannot thank them enough for their role in the completion of this thesis and report.

I am also eternally grateful to all my friends, who helped me and supported me to accomplish this herculean task.

Dedicated to my Parents for their love and endless prayers

&

My supervisor

Dr. Wasi Haider Butt

Who has provided me the guidance, encouragement, and advice throughout my time as his student

ABSTRACT

Software testing is a significant and complex phase of software development as it ensures the complete functionality of the software. To evaluate the behaviour of the software the testing process acquires generation of test cases as an input for the system under test. Generation of test cases remains a challenging task usually as it is done based on human intelligence and it has become much complicated and time-consuming process. Therefore, it can be enhanced by automating the test case generation mechanism to identify and eliminate bugs. Unified Modelling Language (UML) is a de facto standard that has been used in academia and industry currently. Test case automation using UML diagrams is more effective and efficient as it is done early in the software development life cycle. Therefore, to overcome the problem of time as well as budget constraint, it is required to optimize the entire test suite. Many researchers have proposed conventional based approaches for dealing with this problem and they have achieved optimized test cases by selecting, minimizing, or reducing the test cases. However, in this research UML sequence diagram and class diagram have been proposed for white box testing. To generate test cases these two diagrams have been proved as compatible from existing literature. Currently, existing approaches dealing with test case optimization have achieved 85% and 90% statement coverage for the System Under Test 1 (SUT1) and System Under Test 2 (SUT2) using genetic algorithm. However, in this research Unified Non-Sorting Genetic Algorithm (U-NSGA-III) has been proposed for automated test case generation and optimization. The proposed approach has achieved 95% statement coverage for SUT1 and 90.47% for SUT2. In this dissertation, two benchmark case studies have been used and controlled experimentation have been performed for optimization of test cases. The comparison of our approach has been done with GA-UNSGAIII and PSO-UNSGAIII. From our results, it has been concluded that proposed U-NSGA-III has performed better than other approaches.

Key Words: *Testing, White box testing, Optimization, Test cases, U-NSGA-III, Coverage, Genetic Algorithm, Unified Modeling Language*

TABLE OF CONTENTS

Contents

LIST OFTABLES

LIST OF FIGURES

Chapter 1. INTRODUCTION

Traditional software engineering life cycle goes through essential phases of software development such as requirement engineering, requirement analysis, design, implementation, testing and maintenance. However, among all these software testing is a vital and significant component of the software development life cycle. It is a process of analysing the actual and expected behaviour of a software. Testing can be done in two ways either manually or automatically. In manual testing, code is tested manually and it doesn't give effective results [1]. Test case generation is an essential activity of the software test process as the size and complexity of a software increases and automating it, increases the reliability of test cases, reduces errors and faults as well as saves time, cost, and effort. Test case generation methods are classified into the following categories i.e., specification-based testing that includes black box testing and program-based that comprises white box testing. Model-based testing (MBT) is a standard testing strategy in which test cases are extracted from different models. Therefore, the test can be derived from a behavioural model or requirement model. Model-based testing is widely being used to generate test cases, it is considered as the most crucial source of information for test design [2],[3]. To make a development process more versatile, testing is needed to be done before the implementation phase.

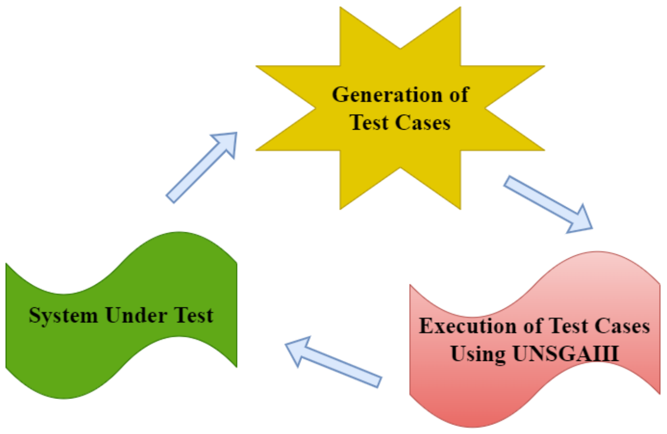


Figure 1: Generation of Test Cases

Test automation from UML specification is more useful as it permits the test plan to be carried out parallel in the development cycle and minimizes efforts, cost enhances software reliability and quality [4]. UML can present specific characteristics of software devices; it consists of the following two categories:

- Structure diagrams: These diagrams describe the structure of the system at the abstract and implementation level.
- Behavioural Diagrams: These diagrams show the behaviour of the system including the dynamic aspects of the system

Mostly, test data generation tools are the source to generate test data and the quality of generated test cases can be assured through test adequacy criteria. However, using different existing testing techniques many test data generators have been developed [7]. To specify the functional requirements of a system UML use case diagrams are widely used. A use case represents how user interacts with the system, its major elements are use cases, actors, and interaction among them with the concept of generalization, include and exclude relationships among the use cases [8]. Testcases can be generated from use cases through test scenario analysis methods. This approach identifies the scenarios from use cases and generates test cases from them. Requirements written in natural language are too uncertain and non-specific to be processed automatically therefore the behaviour of a use case is translated into a formal model for this purpose UML activity diagram has been used to characterize the behaviour of the use case [9]. However, in this study class diagram and sequence diagrams have been used for test case automation and optimization. For this purpose, unified non-dominated sorting algorithm is used that is much efficient for optimization.

1.1 Motivation behind the Research

The method of optimization and generation of test cases against a certain case study is static in nature and which requires the involvement of human experts. With the passage of time, the size and time of test cases increases, which requires more effort for the execution of the test cases in result the testing cost also increases. If a person attempts to manually generate the test cases by using his judgement, it is likely that several important test cases might skip out that has negative impacts on software quality. For addressing the above-mentioned problems, an adaptive approach is required that is also based on expert judgment. All the requirements can be fulfilled by unified non-dominated sorting genetic algorithm (U-NSGA-III). Currently, unified evolutionary optimization process i.e., U-NSGA-III has been used in various activities related to Software Engineering E.g., mono, multi and many optimization problems including constraints or unconstraint are resolved much accurately and efficiently.

1.2 Problem Statement

Automated test case generation and optimization has significantly reduced software testing resources such as cost, development time and other variables that led towards an efficient software testing process. Complicated software development consists of large number of test cases in the test suit which has becomes much difficult to handle manually. Therefore, automated software testing has overcome this problem by automatically generating test cases for such large systems through various approaches. On the other hand, optimization is another way that works by finding best solution and by eliminating redundant test cases from test suits. However, statement coverage with minimum test cases has become a challenge in automated white box testing and optimization techniques. Therefore, this study has proposed an approach that has maximized statement coverage by generating minimum number of test cases.

1.3 Objective

Test case optimization is one of the most challenging activities in software development life cycle. There are two approaches for optimization i.e., Single objective, Multi Objective. Only one objective is considered at a time in case of Single Objective approaches while Multi Objective approaches simultaneously consider several objectives. These approaches incorporate Genetic Algorithm based on multiple optimization objective, Fuzzy Logic and hybrid genetic algorithms etc. In short, for improving the quality of testing process in constrained time and budget, optimizing the test cases is a mandatory task. The basic objective of this research is performing optimization of test cases with maximum path coverage using unified evolutionary algorithms i.e., Unified Non-Dominated Sorting Genetic Algorithm (U-NSGA-III). Furthermore, analysis of genetic algorithm and U-NSGA-III under various circumstances will be done in this research.

1.4 Scope of Proposed Work

Scope of proposed research work includes the study of methods for performing model-based testing for test case automation and heuristic based evolutionary methods have been used to generate and optimize test cases. In this research automated test case generation and optimization has targeted small and medium size software's which need quick testing process containing huge number of test cases in a test suit.

- This research contributes the implementation of unified non-sorting genetic algorithm U-NSGA-III to get the most optimized test cases with maximum coverage from the test suit.

- The expected output of the proposed study is to generate minimum test cases with maximum coverage from the given domain model.
- It also focuses on comparing the proposed technique with the existing one.
- Benchmark case studies are used for validation purpose.

1.5 Title of Research

“Automated white box test case generation for statement coverage using U-NSGA-III”

1.6 Significance of Research

In this research, the area of model-based testing related to automated test cases generation and optimization has been explored and it initiates a new area of research. Following are the key considerations of this dissertation:

- Exploration and analysis of weaknesses in different state-of-the-art meta-heuristics as well as soft computing approaches for automated test case generation and optimization from UML diagrams
- Exploration of applicability of evolutionary algorithms in the field of automated testing.
- The proposed work signifies benefits to the software testing organizations focusing on automated testing by giving an alternative method of test case generation as compared to manual testing.
- UML models have been used to derived test cases as this method is time-efficient and identify errors early in the software development process.
- Software testing is a laborious process. Therefore, automated test case generation reduces development time and resources as well.
- This study contributes an additional knowledge of software testing to the researchers to explore more about automated software test case generation techniques.

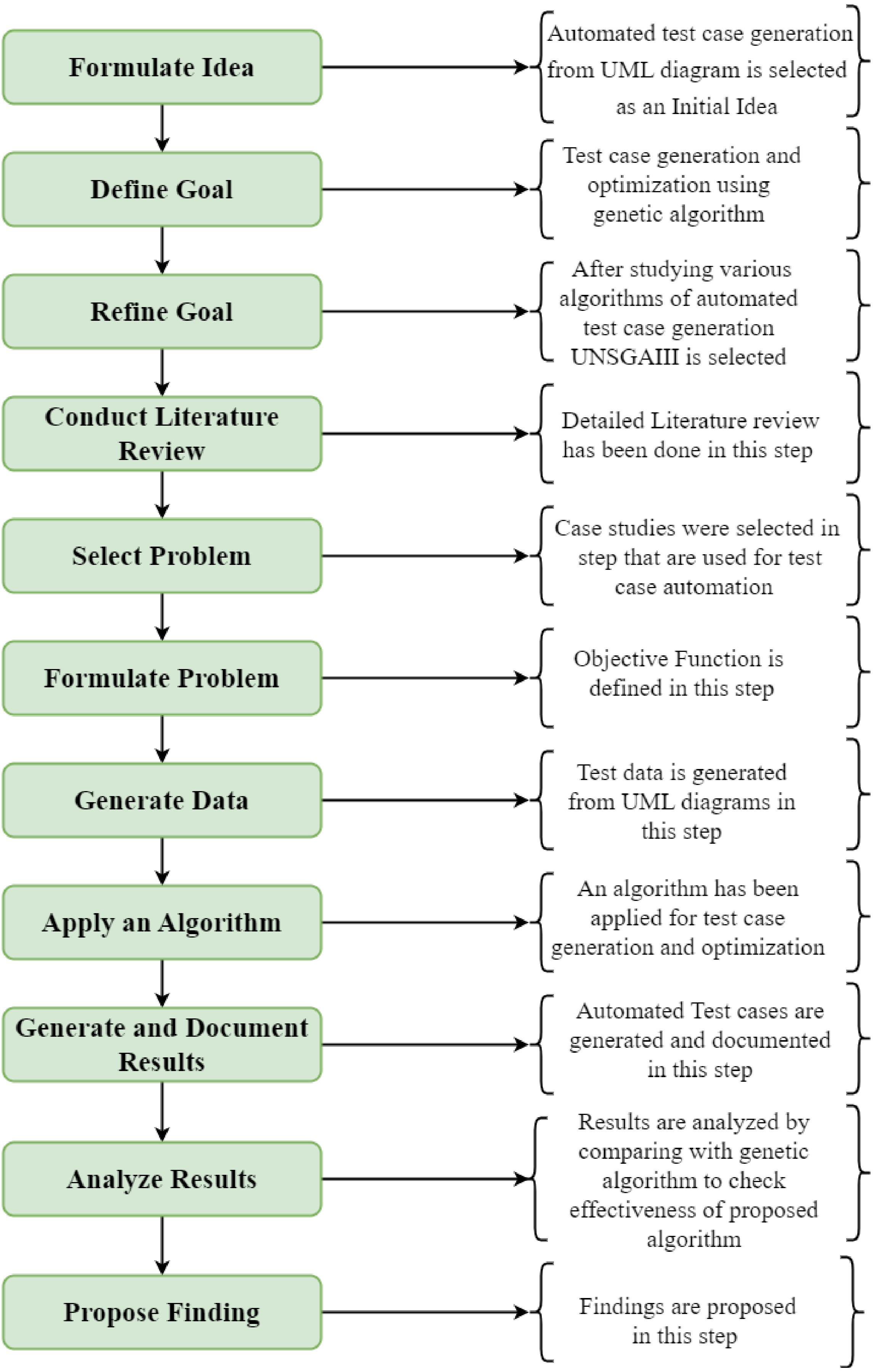
1.7 Methodology of Research

There are six key research questions for this study. To answer them, we conducted a systematic review of state-of-the-art approaches used for model-based automated white box testing for path coverage. During the review process, it has been observed that existing approaches did not efficiently deal with automated testing. So, there is a need to propose an approach that is capable or more efficient for automated test case generation and optimization. Therefore, the proposed study has selected an evolutionary algorithm i.e., U-NSGA-III for automated test case generation and optimization. The results of experimentation are performed on selected case studies.

1.1.1 Research Process

In figure 1, the research process is shown followed for this study. As an initial idea automated test case generation from UML diagram is selected. However, different approaches for automated test case generation and optimization are studied after a careful analysis of techniques a unified non- dominated sorted genetic algorithm has been selected as a goal of this research. After doing an analysis of several approaches based on evolutionary algorithms it is found that UNSGAIII has not been used for the purpose automated white box test case generation from UML diagrams. Although they have great potential for solving complex optimization problems. In the second step the goal is refined and designed the initial draft of this research. Similarly, based on this finalized goal a detailed literature review has been done to analyze the implementation details of several state-of-the-art studies. In the next step cases studies are selected as a test data for test case automation, then objective function is defined to solve the problem. Furthermore, test data is generated from selected case studies.

Finally, the results of optimization were generated, and their analysis was done in both quantitative and qualitative manner to propose our findings.



Formulate Idea
Automated test case generation from UML diagram is selected as an Initial Idea
Define Goal
Test case generation and optimization using genetic algorithm
Refine Goal
After studying various algorithms of automated test case generation UNSGAIII is selected
Conduct Literature Review
Detailed Literature review has been done in this step
Select Problem
Case studies were selected in step that are used for test case automation
Formulate Problem
Objective Function is defined in this step
Generate Data
Test data is generated from UML diagrams in this step
Apply an Algorithm
An algorithm has been applied for test case generation and optimization
Generate and Document Results
Automated Test cases are generated and documented in this step
Analyze Results
Results are analyzed by comparing with genetic algorithm to check effectiveness of proposed algorithm
Propose Finding
Findings are proposed in this step

Figure 2 : Process followed by proposed research

Structure of Thesis

The structure of the thesis is as follows:

Chapter 2 covers the basics and background of test case automation, unified modelling language and single-objective optimization.

Chapter 3 gives a review of the literature in detail and the significant work done by researchers in the past few years. The systematic literature review is composed of three main sections. First section is review protocol which gives details on the methodology using which the literature review is carried out. Section two presents' details on research works carried out in this area in form of research question and tables. Whereas section three highlights the research gaps that were encountered.

Chapter 4 consists of the proposed approach in detail. It discusses the approach in terms of overview of algorithm (U-NSGA-III, fitness functions), main components of approach and solution representation.

Chapter 5 includes implementation, validation and discussion of results accompanied by research question and relevant figures. It also brings detail to the comparison of our work with the state of the art. Additionally, it briefly explains the limitations of our work.

Chapter 6 concludes the thesis and reveals the future scope of this research.

Thesis outline is shown in the Figure 3.

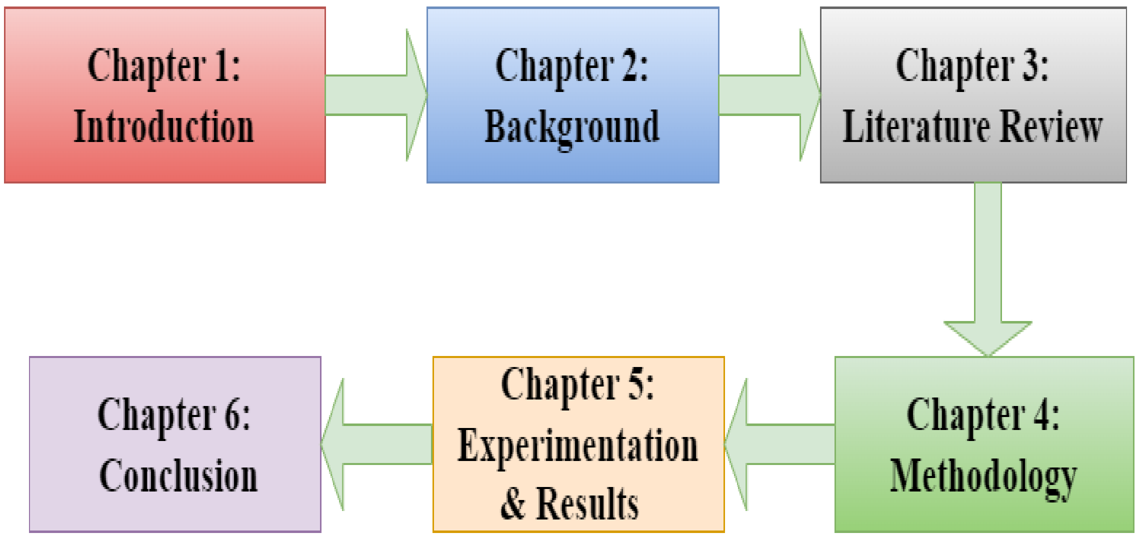


Figure 3. Thesis Outline

Chapter 2. Automated Test Case Generation

In this section, the necessary background related to Test case automation, Uniform Modeling Languages (UML) and single-objective optimization is described.

2.1 Test Case Automation

Software testing is a significant process during software development that ensures the quality of a product. In manual testing, code is tested manually and it doesn't give effective results [1]. Execution of test cases multiple times is not easy in manual testing, but it can be done easily via automated testing techniques. Test case generation is an essential activity of the software test process as the size and complexity of a software increases, automating it enhances the reliability of test cases, reduces errors, faults as well as saves time, cost, and effort. Software testing is done to make sure that software is reliable and works according to the expected output. Testing is done to identify and fix errors detected in software before its release. Software testing is a challenging and time-consuming process, it costs almost 50% of the software development resources. This problem requires a solution to optimize the test cases in minimum time and budget constraint. Therefore, test automation techniques have become a dire need in software testing domain. Automated methodologies are used to execute and generate optimized test cases much effectively as compared to manual test cases that are generated by humans. However, formal models are being frequently used as an input for automated test case generation. It provides more realistic, less expensive and better-quality testing process. Similarly, various methods have been used by researchers to produce optimized test cases such as ant colony optimization (ACO), fuzzy clustering algorithms, particle swarm optimization and many others. These methods may not be able to optimize test cases in a more efficient way as compared to other evolutionary algorithms i.e., unified non determinant sorting genetic algorithm.

2.2 Unified Modeling Languages

A model is a generalization of a reality. Unified Modelling Language (UML) is a standard language used to visualize the detailed aspects of a system. Model-based testing (MBT) is a standard testing strategy in which test cases are extracted from different models. Model based software testing is done in the start of software development process, therefore it lessens time and cost. A strong and good software can be developed by using this approach. It improves the quality of a product and reliability as well. Model-based testing consists of three major components, test generation algorithms, model notations and tools used as a supporting structure for the test. Automated test cases can be generated from several types of models, but UML models have been significantly used to derived test cases as this method is time-efficient and identify errors early in the software development process. Defects that come into requirement phase due to inconsistent requirements may cause heavy cost later in developmental stages if remained undetected [5]. To generate test cases MBT uses UML diagrams i.e., class diagram, activity diagram, use case diagram, sequence diagram, communication diagram, and state chart diagram as shown in figure 4. These diagrams are then converted into an intermediate form using different approaches and algorithms for automated test case generation [6].

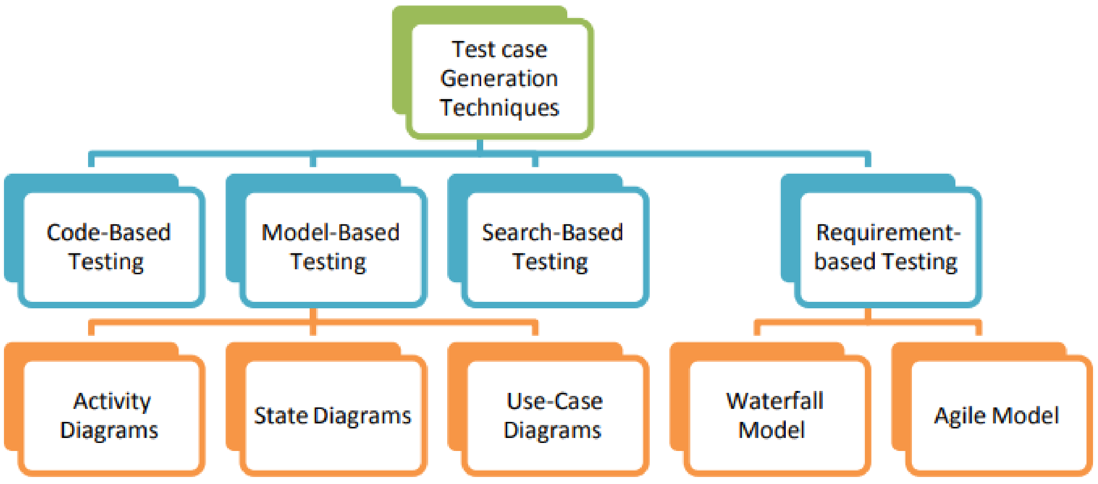


Figure 4. Survey Architecture

The software under testing (SUT) can be checked for errors using two significant testing techniques.

- White Box Testing
- Black Box Testing

2.3 White Box Testing

In white box testing the internal structure of the SUT is observed. It is also known as structural, glass box testing or program-based testing. Testing of an internal system is done by running the code. In this type of testing tester must have implementation knowledge as well. Appropriate outputs are determined by choosing inputs of SUT to execute paths through the code [7]. All the source code paths are examined by using this technique. Therefore, highly skilled resources and implementation knowledge is needed to carry out white box testing. There are various structural testing-based coverage criteria

2.4 Statement Coverage

In statement coverage every statement is executed at least once while testing. It is considered as weak because if a program consists of IF statements and ELSE statements then either IF statement will be executed or ELSE statement.

2.5 Branch Coverage

Branch coverage is considered as better than statement coverage. In this process, each branch must be executed at least once. However, decision coverage generally satisfies statement coverage. If every branch is executed, then every statement must have undergone through the execution process while testing is being done. Therefore, if the branches and statements are executed in a specific order then there is a possibility to detect some errors during the testing process. However, to detect these errors path testing is required.

2.6 Path Coverage

This method of testing is stronger as compared to statement and branch testing, because in path testing each possible path of SUT must be executed. In this way probability of error identification gets increased. If a path is traversed by giving an input during code execution, then it is called as feasible path. Similarly, if the path is not traversed during program execution it is known as unfeasible paths.

2.7 Black Box Testing

In black box testing overall functionality or external behavior of a system against its specification is checked. In this method test data is generated from SUT specification early in the development life cycle. Black box testing manly focuses on expected output of an application for given input values [7]. The internal working of the software is not known to the tester in this testing technique thats why it is called as behavior or specification-based testing. Both valid and invalid inputs, according to the users' requirements are handled by this type of testing. Black box testing includes different types i.e., equivalence partitioning, boundary-value analysis, decision table based and error guessing.

2.8 Equivalence Partitioning

In equivalence partitioning, inputs of a program are partitioned into equivalent classes so that the test values of each class is equivalent to test of any other value in the equivalent class could be easily assessed. If one test case from the corresponding class identifies bugs, then rest of the test cases that belong to the same class would be expected to find same bugs. Similarly, if a test case of the same class does not find an error, then no other test case will identify the same error from the that class.

2.9 Boundary Value Analysis

In boundary value analysis, the boundary values of input data of SUT are taken. Test cases are designed and then one test case is selected from each boundary value containing both valid and invalid inputs. However, this testing technique do not analyze multiple input data of a system.

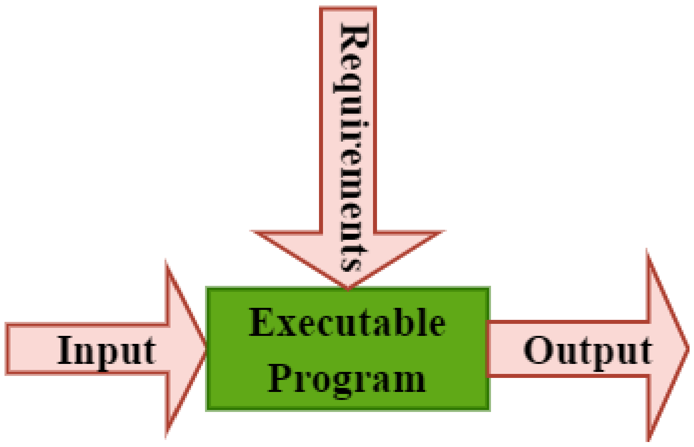


Figure 5. Illustration of Black Box Testing

2.10 Genetic Algorithm

Genetic algorithm (GA) works on the principle of natural selection and genetics, inspired by the biological evolution of human being. Genetic algorithms are considered as well-known optimization tool and are very easy to use. Generally, GA consists of two main steps, in the first process individuals based on fitness values are selected from the population using specific selection method. In the second step production of new individuals take place through cross over and mutation techniques. The search space of genetic algorithm consists of candidate solutions representing the problem called chromosomes. These algorithms provide more than one solution also known as alternate possible solutions. The goal is to produce such solutions that are able to optimally satisfy each objective concurrently. These credentials are termed as ‘Objectives’ and an algorithm may use a single objective, may incorporate multiple objectives, or may utilize many objectives. For example, an algorithm considering only one objective (i.e., test case automation) may be termed as a single objective utilization algorithm. The algorithm that takes into account two or three objectives is termed as a multi-objective optimization.

A random population of random pop size known as individuals is initiated in the beginning of Genetic Algorithms. It is reproduced iteratively by exploring the fittest individuals from the search space. The target problem is defined and then evaluated by using a fitness function (one fitness function for one respective objective) for the population. A random selection of 2 chromosomes as parents is selected to perform crossover and mutation operators on them. While applying selection operator, it is ensured to select the individuals with highest fitness values so that the new individuals come up with higher probability to undergo crossover and mutation operators. A new population is created from old population with a purpose to keep producing the better population every next time. The stopping criteria is defined to stop the algorithm where the individuals/population found is the best and fittest population. The flowchart of a genetic algorithm is displayed in Figure 6.

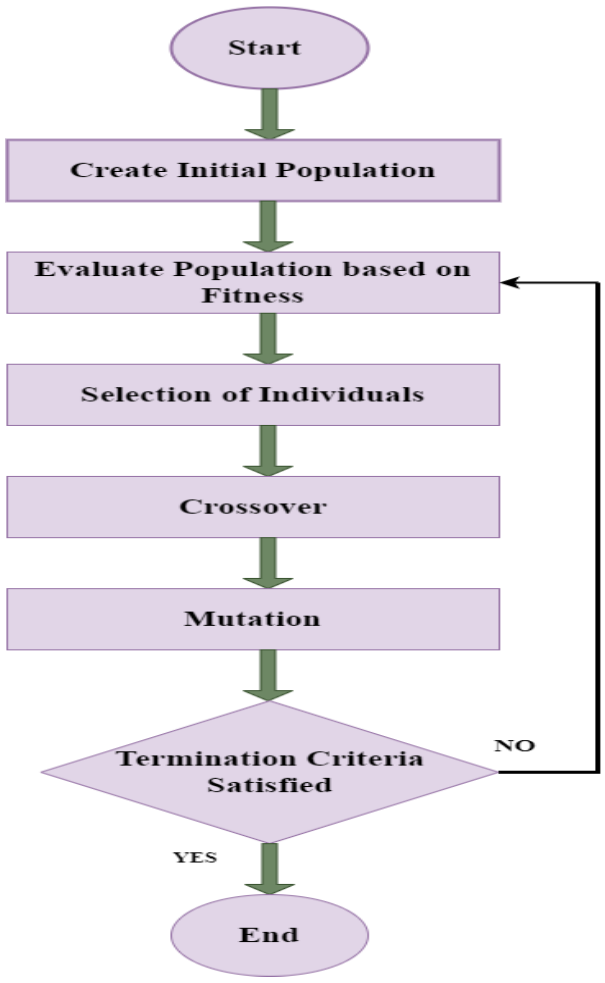

Start
Create Initial Population
Evaluate Population based on Fitness
Selection of Individuals
Crossover
Mutation
Termination Criteria Satisfied
NO
YES
End

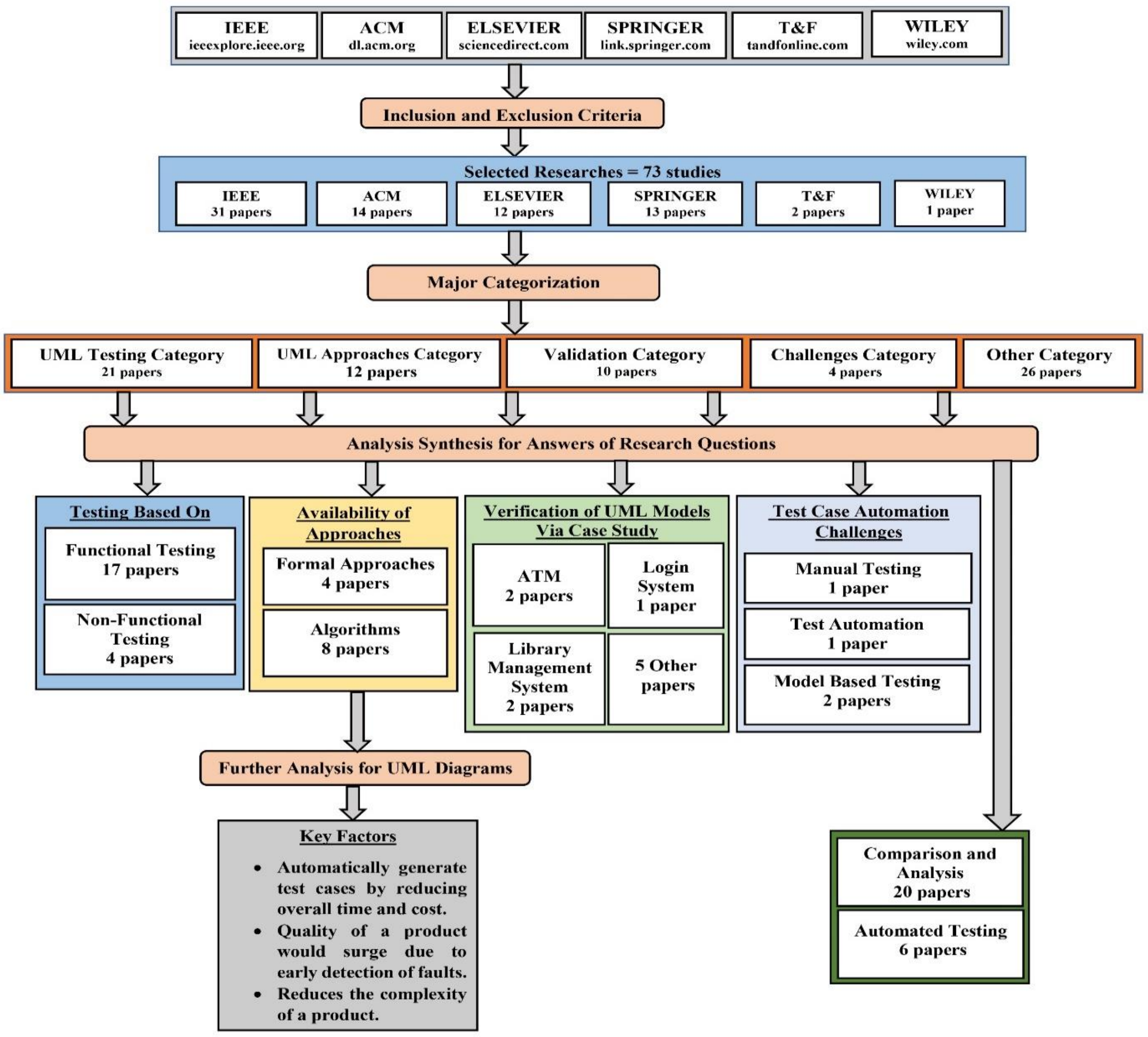
Figure 6. Genetic Algorithm Flowchart

Chapter 3. Literature Review

This chapter contains the systematic literature review that has been done for the proposed research. It includes overview and major outcomes of SLR, review methodology, research questions, category definitions, review protocol, results and analysis, answers to the research questions for literature and conclusion.

3.1 Overview and Major outcomes of SLR

Several studies are contributing to this systematic literature review, where various approaches, case studies, algorithms/techniques, and validation of various case studies are highlighted in bits and pieces. Moreover, comprehensive research covering all the major aspects concerning the automated test case generation as part of optimization, and detailed study of UML diagrams, test coverage criteria are found in the existing studies.



IEEE ieeexplore.ieee.org
ACM dl.acm.org
ELSEVIER sciencedirect.com
SPRINGER link.springer.com
T&F tandfonline.com
WILEY wiley.com
Inclusion and Exclusion Criteria
Selected Researches = 73 studies
IEEE 31 papers
ACM 14 papers
ELSEVIER 12 papers
SPRINGER 13 papers
T&F 2 papers
WILEY 1 paper
Major Categorization
UML Testing Category 21 papers
UML Approaches Category 12 papers
Validation Category 10 papers
Challenges Category 4 papers
Other Category 26 papers
Analysis Synthesis for Answers of Research Questions
Testing Based On
Functional Testing 17 papers
Non-Functional Testing 4 papers
Availability of Approaches
Formal Approaches 4 papers
Algorithms 8 papers
Verification of UML Models Via Case Study
ATM 2 papers
Login System 1 paper
Library Management System 2 papers
5 Other papers
Test Case Automation Challenges
Manual Testing 1 paper
Test Automation 1 paper
Model Based Testing 2 papers
Further Analysis for UML Diagrams
Key Factors
Automatically generate test cases by reducing overall time and cost.
Quality of a product would surge due to early detection of faults.
Reduces the complexity of a product.
Comparison and Analysis 20 papers
Automated Testing 6 papers

Figure 7 Overview of the SLR

The structure and organization of the systematic literature review with complete detail is shown in Figure 7. Six well known scientific repositories are selected in the search process and as a result 73 studies are selected that fully meets the inclusion and exclusion criteria. Furthermore, the selected research is classified into five major categories i.e., UML testing category, UML approaches category, validation category and general category. For ease of analysis and to explore the studies as per the research questions, studies were further categorized into five groups i.e., UML testing category (21 studies), UML approaches category (12 studies), validation category (10 studies), challenges category (4 studies) and general category (26 studies). To obtain the required and precise results, a combination of qualitative and quantitative analysis is performed on the selected studies. As highlighted in Fig. 7. Furthermore, there two types of UML test case generation i.e., functional and non-functional. Functional testing includes (17 studies) non-functional testing (4 studies). Similarly, verification of case studies includes different case studies such as ATM (2 studies), LMS (2 studies), login system (1 study) and other (5 studies). Moreover, test case automation challenges incorporate manual testing (1 study), tests case automation (1 study), model-based testing (2 studies). The answers to the research questions are extracted with the help of data analysis and synthesis method. In general category, comparison and analysis (20 studies) and automated testing (6 studies). Testing based on functional and non-functional testing (References, [8],[9] etc. Similarly, availability of approaches (References, [10],[11]) and verification of UML models is done via case study (References, [12],[13]). Challenges in test case automation includes (References, [14],[15] etc. The detailed discussion on these categories is carried out in section III. Answers to research questions are discussed in section IV.

The main contribution of this study is summarized below:

- This research has selected 73 studies published during 2011-2021 containing detailed analysis of UML diagrams followed by the defined guidelines of conducting SLR.
- For test case automation different techniques and approaches of UML diagrams have been identified by following the ways due to which model-based automated testing is better than other methods.
- Validation of an identified approaches is done via case studies.

- Finally, this research has highlighted the more efficient UML diagram for test case automation in software testing based on coverage criteria and defects detection.
- Challenges in test case automation from UML diagrams.

The purpose of a Systematic Literature Review is to identify, assess and to do a detailed analysis of existing relevant research that has been done in previous years in a particular domain. The SLR ensures comprehensive analysis and is conducted by a precise search plan by following the Kitchenham guidelines[16]. These guidelines have been performed in a very systematic way including the five major steps category definition, Development of a review protocol, search process, Inclusion and Exclusion criteria, Quality assessment, Data extraction and synthesis.

3.2 Review Protocol

A review protocol is developed for the selected studies. The major parts of this methodology are planning, preparing and reporting the review as shown in figure 8. It focuses on the background of the research, research questions related to a particular topic, inclusion and exclusion rules associated with quality evaluation, search process, data extraction and synthesis. Furthermore, data sources and search criteria have been highlighted in table 1. The goal is to review the latest research available on automated test case generation from UML diagrams. This systematic Literature Review (SLR) contributes the techniques or approaches used to derive automated test cases from UML diagrams and how model-based testing is better than traditional testing methods. Similarly, it has also identified the UML diagram that is more appropriate in the automated test case generation process.

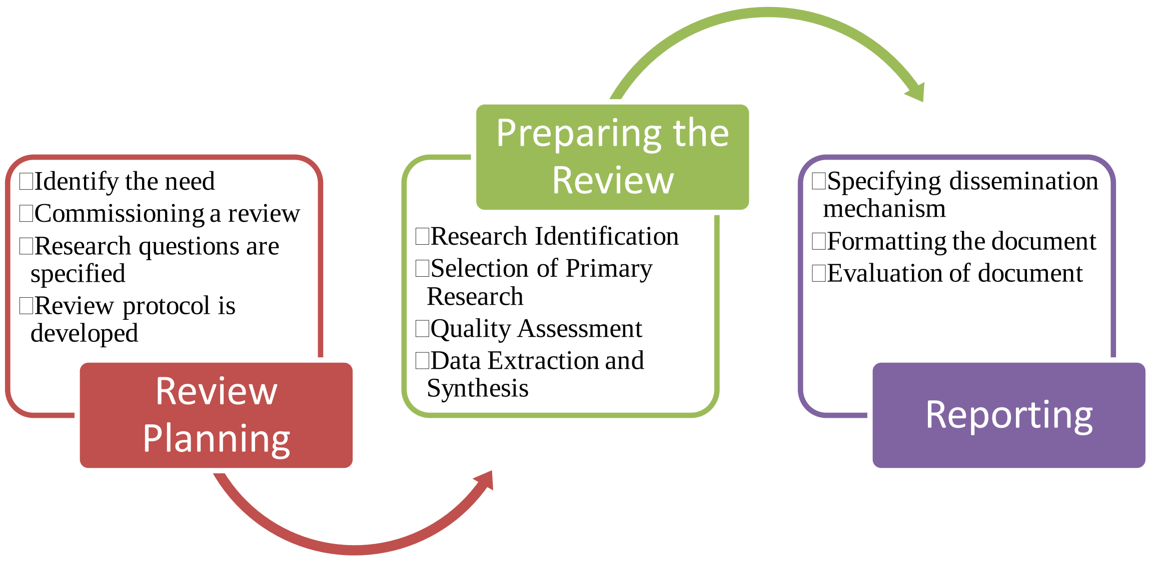


Figure 8. Steps Followed in SLR

3.2.1 Research Questions

Following research questions have been identified:

RQ1: What are the foremost research that have been published in the past ten years for test automation?

RQ2: What type of automated testing has been performed by UML diagrams?

RQ3: What are the techniques and case studies used in existing work for test automation from UML diagrams?

RQ4: Which UML diagram can be considered as a better option for test case automation?

RQ5: How automated test case generation from UML is better as compared to other testing ways?

RQ6: What are the challenges in test case automation which have been overcome by UML?

Table 1. Data Sources and Search Criteria

Electronic Repositories	1. IEEE Xplore (https://ieeexplore.ieee.org/Xplore/home.jsp) 2. Springer (https://www.springer.com/in) 3. ACM Digital Library (https://dl.acm.org/) 4. ScienceDirect (https://www.sciencedirect.com/) 5. Taylor& Francis (https://www.tandfonline.com/) 6. Wiley (https://onlinelibrary.wiley.com/)
Language	English
Publication Period	2011-2021
Searched Items	Books, journals, and conference papers having any of the search strings in their title or keywords

3.2.2 Category Definition

The proposed research is organized into five major categories for classifying each selected study that provided us with an appropriate way of answering our research questions.

1) **UML Testing Category:** Functional and non-functional are the two major types of testing. The research studies specifically dealing with the functional and non-functional testing types are placed in this category. This includes the studies that are based on automated generation of test cases from UML diagrams e.g., the automated test cases by UML models are generated by doing functional testing such as system-level testing, integration testing, regression testing and unit level testing. Additionally, components of non-functional testing i.e., security testing, performance testing and robustness testing are also placed under this category.
2) **Approaches Category:** The research studies dealing with techniques and approaches for automated test case generation from UML diagrams are placed under this category. Various algorithms and approaches are comprehensively investigated that have been

used for automated test case generation from UML diagrams are defined in the respective category. For example authors in [11] and [12] have used genetic algorithm that have decreased the number of unrealistic test cases and made the testing process faster. However, all such studies where these approaches are utilized for test case automation comes in this section.

3) **Validation Category:** There are researches where different case studies have been taken to validate the UML diagrams for automatic test case generation. For example, authors in [10] have validated the model diagrams through a case study. Similarly, in another study [13] automatic teller machine (ATM) is used to accomplish the automated test case generation process. Therefore, all such studies where case studies are considered to achieve this goal are included in the validation category.
4) **Challenges Category:** This category includes the research studies that focus on challenges that are faced in test case automation. For example authors in [15] have presented issues with manual testing that have been overcome by UML diagrams by automating them. Therefore, research studies dealing with challenges in test case automation are incorporated in the corresponding category.
5) **Other Category:** All the above defined studies belong to this category. In few studies, UML testing along with functional (e.g., regression, integration, system level, unit testing) and non-functional (security testing, robustness, performance) types of testing is done for the automatic generation of test. Such studies instantaneously targeting UML testing are placed under general category. Similarly, there are few studies (e.g., [17]) where certain techniques and approaches for automated test case generation are identified. On the other hand, some studies has proposed case studies for example (movie tickets booking system [10] etc) for the validating the proposed approaches and UML input models come in this section. Furthermore, research dealing with challenges of automated test case generation, that do not belong to either validation nor approaches category are also placed in general category.

3.2.3 Selection and Rejection Criteria

Four renowned databases incorporating IEEE, ACM, Springer, and Elsevier are chosen in selection and rejection criteria. However, some other databases i.e., Wiley online library, Taylor and Francis are also included to carry out this systematic review. In selection and rejection criteria some rules are defined as explained below.

Table 2. Search Terms with Results

Sr.No	Search Keywords	Operator (AND/OR)	No. of Search Results					
			IEEE	Wiley Online Library	ACM	Springer	T & F	Elsevier
1	Unified modelling language	AND	30	2	15	40	1	63
		OR	48	5	30	24	3	28
2	Test case automation	AND	20	0	10	25	2	10
		OR	35	2	20	12	5	17
3	Software testing using UML	AND	45	0	28	0	0	35
		OR	15	3	32	10	2	11
4	Test case automation using UML	AND	30	1	0	45	0	0
		OR	34	4	25	29	5	15
5	Automation of software testing using UML	AND	20	5	15	23	4	20
6	UML, Test case generation	AND	10	0	0	26	0	12
		OR	40	1	9	35	2	3
7	UML, Test case automation	N/A	32	2	20	22	6	8

1) **Subject:** The research studies in which the words such as UML diagram and test case automation are used in the abstract are selected. We have not considered those research in which UML diagrams and test case automation are not discussed in the abstract. Some studies are also rejected based on the title where test case automation or UML diagrams were not present.
2) **Publication Year:** Only the latest studies published during 2011- 2021 are included, all other research papers other than the defined limits are excluded.
3) **Publisher:** To carry out this SLR four well-known databases IEEE, ACM, Springer, and Elsevier are chosen. Moreover, some other search engines i.e., Wiley online library, Tylor & and Francis have also been considered. Studies other than these repositories are rejected.
4) **Application Research:** Result-oriented research studies which have effective methodology with the correct outcome are considered, whereas weak approaches with unsatisfactory results are excluded.
5) **Repetition:** Research with the most similar content are discarded, only relevant studies are included.
6) **Validation of proposal:** Proper validation of research studies is really an integral part of research studies. Therefore, only those studies are selected that are properly validated through some appropriate methods like experimentation and case study etc.

3.2.4 Search Process

Six major databases IEEE, Springer, ACM, Elsevier, Wiley, and Taylor & Francis have been selected to carry out the search process as per in selection and rejection rules. We have used different search terms or keywords for this process as summarized in Table 2. Particularly, most relevant search terms like test case automation etc were used to start the search process. However, these search terms extracted thousands of results that were difficult to analyzed. For example, ACM digital library returned 513,934 results in default settings for "Test case automation" search term. Therefore, we used different filters like "Publication Year" (2011-2021) to optimize the search results. Similarly, two operators such as AND, OR are used to obtain more relevant results. The outcomes gathered from AND operator was not enough that's why the OR operator is used. However, the outcome gathered by utilizing OR operator was too large, and it was not possible to check all operators therefore, an advance search option has also been used provided by these scientific databases. To speed up the search process we further have used search options such as "where title or abstract contains" etc.

After doing the comprehensive analysis only 73 studies that are relevant to our proposed domain have been selected and 63 studies are rejected. The following steps are carried out in the search process as shown in figure 9.

- By using different search terms in well-known databases overall, we have collected 1131 results.
- Among the collected results, we have rejected 348 studies based on title, and 142 studies based on abstract as defined in our selection and rejection rules.
- Moreover, 63 studies are rejected based on general studies and further, an analysis of 136 studies has been done.
- Finally, after investigations, 63 studies are rejected and 73 research have been selected.

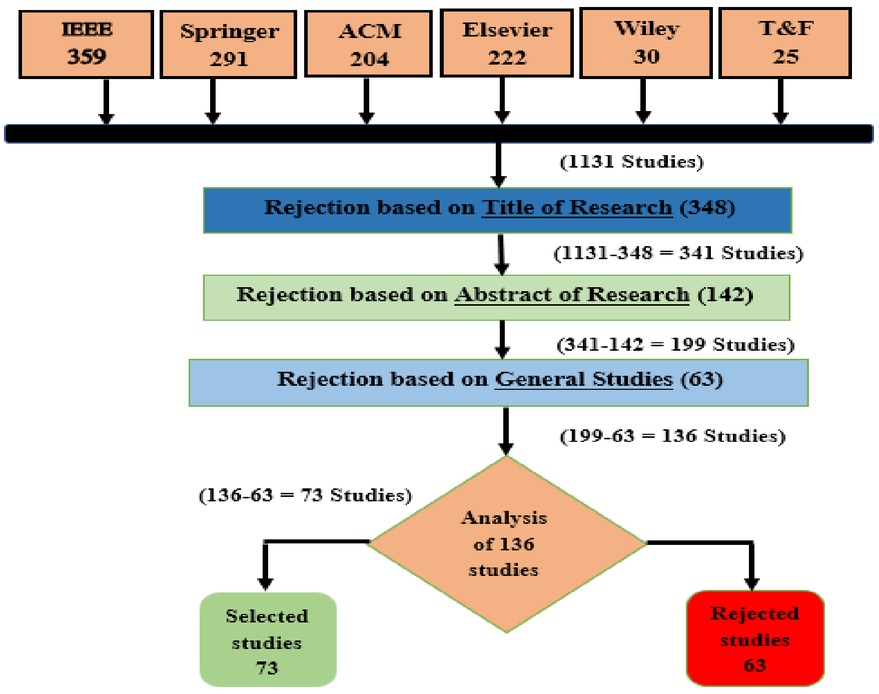


Figure 9. Search Process

3.2.5 Quality Assessment

High impact studies from well-known repositories that are accepted all over the world have been selected to assure the reliable outcome of this SLR. It can be seen from Table 3 that Thirty-one (31) research studies are selected from IEEE, thirteen (13) studies are extracted from springer, fourteen (14) from ACM, twelve (12) from Elsevier two (02) from other Taylor and Francis and one (01) from Wiley databases. It is worthwhile to note that the identified studies are selected without any biasness.

In fact, results are searched and analyzed properly from each renowned databases, where comparatively higher number of studies were found from IEEE and ACM. In contrast, few numbers of studies are obtained from Elsevier, Wiley and Taylor & Francis. It highlights that the high impact studies and latest results are chosen to develop this research methodology. The Database shows the name of databases, Type represents that either the research study is a conference or journal whereas references describe the selected studies, and the total gives a number of references from given databases. However, parameter 2 (publication years) ensures the identification of latest studies only. Similarly, the parameter 4 (Application Research) of inclusion and exclusion criteria indicates that only those studies are selected that have genuine techniques or framework. This substantially improves quality of this SLR as the insignificant research articles are not considered. This leads to analyze the generation of test case automation from UML diagrams.

Table 3:Summary of Selected Studies

Sr. No	Database	Type	References	Total
1.	IEEE	Conference	[12], [18] ,[19], [20], [21], [22], [23], [24], [25] [26], [27], [28], [29], [30], [31], [32], [17], [10], [33], [34], [35], [36],[9], [37], [38], [39], [40], [41], [42], [43], [44]	31
2.	Springer	Conference	[45], [46], [13], [11], [47], [48], [49], [50], [51], [52], [53]	13
		Journal	[54], [55],	
3.	ACM	Conference	[56], [57], [58], [59], [60], [61], [8], [62], [63]	14
		Journal	[64], [65], [66], [67]	
4.	Elsevier	Conference	[68], [69], [70], [15]	12
		Journal	[71], [72], [73], [14], [74], [75], [76], [77]	
5.	Tylor & Francis	Journal	[78], [79]	02
6.	Wiley	Journal	[80]	01

Moreover, the quality of SLR is ensured by the type (i.e., conference or journal) of selected studies. Although, we tried to choose journal studies as much as possible and we extracted 19 journal studies (out of 73) which were completely complaint with the inclusion and exclusion criteria. Therefore, the distribution of selected studies with respect to publication type is shown below in figure 10. Where 26% of the identified studies are from well-known journals while rest of the 74% studies are selected from conferences.

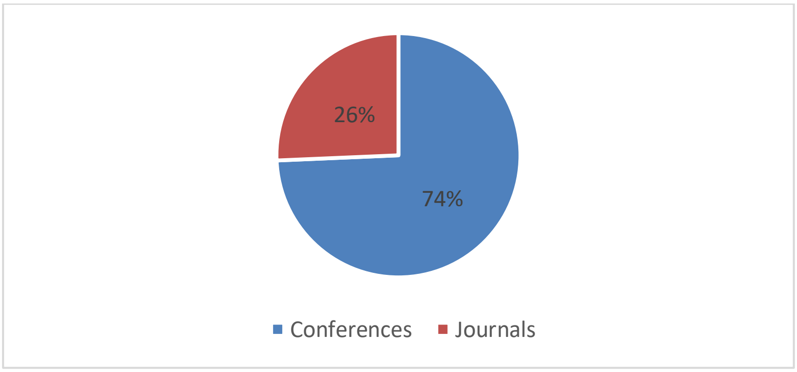


Figure 10. Distribution of selected studies with respect to publication type

3.2.6 Data Extraction and Synthesis

After the identification of studies according to the inclusion and exclusion criteria, the process of data extraction and synthesis is executed. Various data extraction elements have been identified to find the answers to research questions as shown in table 5. Specifically, the key elements are first extracted from serial no 1 to 5 of Table 4. Similarly, for other important elements, data extraction with synthesis is performed as shown from serial no 6 to 10 in the respective Table. The mining of bibliographic information of each selected study is done. After that data collection with the proposed methodology and results are extracted. This is the best way to achieve the goal of the proposed research. Furthermore, a grouping of each selected study to the corresponding categories is done and the validation of the proposed methodology is observed in the selected studies. Moreover, techniques or approaches that are utilized to generate test case automation are identified in the chosen studies. Finally, a detailed analysis is done to answer the research questions proposed in our systematic literature review.

All the major components of review protocol have been discussed. Moreover, we have selected 73 studies based on inclusion and exclusion criteria. Furthermore, data extraction is carried out to obtain relevant studies from the identified studies. This has led to compile the results accurately as given in following section.

Table 4: Data Extraction and Synthesis

Sr. No	Description	Details
Data Extraction		
1.	Bibliographic Information	The Title of the paper, its publication year and publication type is observed
2.	Data collection	Either data is qualitative or quantitative
3.	Proposed Methodology	Methods that are followed in selected studies are analysed
4.	Results	Results of each study are comprehensively analysed
5.	Validation	Methods of validation of respective techniques are observed
Data Synthesis		
6.	Research Overview	Overview of proposed SLR incorporating introduction and research methodology
7.	Grouping	Selected studies are grouped into their corresponding categories as summarized in Table 3.
8.	UML Diagrams techniques	Techniques of test case Automation from UML are analysed as shown in Table 7.
9.	Testing Mechanism	Ways to derive test case automation
10.	Findings of each category	Detailed analysis of each research question to get the answer and results of research answers are shown in Table 7 and Table 8

3.3 Results & Analysis

The identified studies are classified into five categories as given in Table 11. For further exploration the references of each study are provided against each category. The primary aim of this review is to examine and query the selected literature in order to reliably answer the identified research questions. This section reports the results after careful examination of various aspects of extracted data. It is worthwhile to mention that the Journals generously contributing to our quest for the latest trends in test case automation and optimization from UML diagram are; Journal of Systems and Software, Software & Systems Modeling, International Journal of Software Engineering, International Journal of System Assurance Engineering and Management, Journal for Science and Engineering, SIGSOFT Software Engineering Notes, Journal of King Saud University-Computer and Information Sciences, Journal of Software: Evolution and Process. Similarly, various significant conferences are; International Conference on the Quality of Information and Communications Technology, International Conference on Evaluation and Assessment in Software Engineering, Conference on Computer Science & Software Engineering, International Conference on Information and Software Technologies and many other are included in this SLR.

3.3.1 Model-Based Testing

Software testing is a pivotal activity in the software development life cycle. It evaluates the quality of a product and detects the potential failures of the software under test [12]. Testing can be done in multiple ways i.e., code-based testing, requirements-based, and model-based. Code-based testing is done on code, it is not very effective as it is carried out later in a developmental phase that leads to many errors, takes much time and intolerable cost. Whereas, in requirements-based testing, test cases are generated from requirements' written in a natural language which is not a very successful way to derive test cases. Requirements written in a natural language can be ambiguous that causes errors in the next phase of development. Moreover, it is challenging to test a complete set of requirements by single test cases[59]. The model-based testing technique is one of the automated testing techniques that has improved the traditional testing methods [72]. Model-based testing produced high quality product as compared to code-based testing and requirement-based testing as shown in table 5.

However, UML diagrams are extensively used in software modeling process test case generation from UML diagram shows higher efficacy, reduces development time and cost [36]. Software testing took a lot of time, effort, less productivity and produced low-quality products by using traditional testing models. Therefore, test automation not only improves the quality of a product but also increases test coverage[47]. Different test cases could be generated from the UML diagram by identifying more errors and a very appropriate way for testing the entire system. Automated test case generation from UML diagrams has been done through various methods so far, test cases can be generated using one diagram and by using a combination of multiple diagrams. Test generation remained a problem for the system based on concurrency rules, but the usage of the UML activity diagram has overcome this issue by generating test cases automatically for such type of software system[71]. Similarly, UML models represent the behavior of the system abstractly which is a practical way to automate test activities. UML designs are very supportive and suitable for modelling several types of specifications[54].

Table 5: Testing Methods

Sr. #	Approaches of testing	Cost	Development Time	Efficiency	Quality of Product
1.	Code based testing	High	High	Low	Low
2.	Requirement based testing	High	High	Low	Low
3.	Model based testing	Low	Low	High	High

3.3.2 Testing Types Based on UML Models

Software testing is a crucial element in development of a product. In this study 20 research studies have been identified as shown in Table 6. Various UML based techniques for software testing are proposed [64]. Uses cases are investigated for test case generation that express the functional requirements in a very simple way. For functional testing comprehensive research has been done that includes different UML models such as use cases, sequence diagrams, activity diagrams and state machine diagrams. However, test case generation from UML models for functional testing is done using following classified methods i.e., heuristic testing, and UML based specification. This is carried out at unit testing, integration testing and system level testing [51]. Similarly, UML activity and use cases are utilized for non-functional testing that consists of performance testing, robustness and security testing. Security vulnerabilities are widely detected by UML based testing. Furthermore, system performance under a particular workload is tested to examine the responsiveness, stability and various other attributes of the system that involves the model-based testing techniques to generate test cases, test data, test execution and evaluation.

Table 6: Types of Testing

Sr. NO	Testing Type	Count	References
	Functional Testing		
1.	System level testing	01	[8]
2.	Regression Testing	06	[9], [55], [53], [75], [37], [76]
3.	Unit Testing	03	[38], [39], [81]
4.	Acceptance Testing	02	[40], [41]
5.	Integration Testing	04	[79], [39], [62], [77], [51]
Non-Functional Testing			
6.	Performance Testing	02	[63], [42]
7.	Security Testing	01	[43]
8.	Robustness Testing	01	[44]

Graphical representation of functional and non-functional testing has been shown below in figure 11.

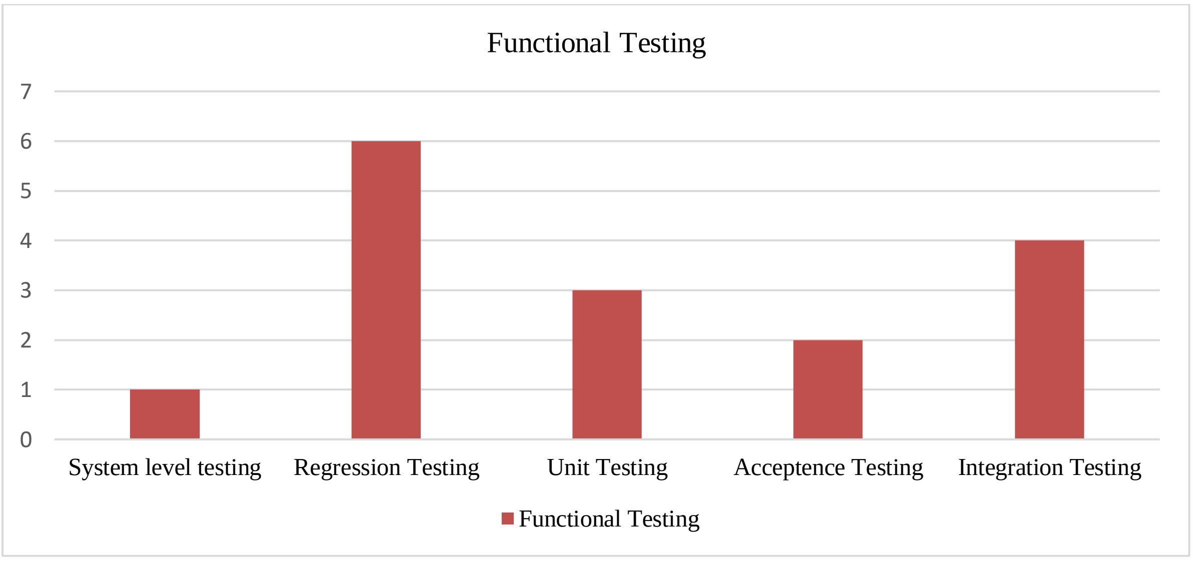


Figure 11. Functional Testing

3.3.3 Approaches for Test Case Automation

In this research 18 techniques/approaches have been identified for test case automation from UML diagrams. A genetic algorithm is widely used to generate test cases and to optimize them because of its simplicity and effectiveness. Test cases produced by genetic algorithms are better in quality than the test cases generated by some other random methods. GA is an efficient optimization algorithm that decreases the number of unfeasible test cases and makes the testing procedure faster. It is applied for testing object-oriented software as well as procedural software specifically in scenario testing. Similarly, the depth first search (DFS) algorithm has been used mostly for test automation because it is one of the optimized algorithms. Test cases generated from this algorithm are not only exhaustive but optimum. The use of a hybrid genetic algorithm i.e., the combination of both genetic algorithm and particle swarm optimization helps in finding the optimal solution of a problem.

Table 7: Analysis of Approaches for Test Automation by UML diagrams

Sr. No	Input Model	Algorithm/Approach		Language	Reference
1.	State Machine	Ant Colony Optimization (ACO)		NLP	[17]
2.	Sequence diagram and UML use case Model	Synthesis algorithm for Automata & Test cases set generation algorithm	Incremental approach based on Finite Automata & Event Deterministic Finite Automata	OCL	[10]
3.	Sequence diagram	ATGSD		Graphical Notation	[13]
4.	Use case	GEN SCENARIOS, GEN TEST INPUT DATA GEN TCSL MODEL		NLP	[21]
5.	Usecase specification	GenerateInput /UMTG Approach		NLP	[60]
6.	Activity diagram	Genetic Algorithm		NLP	[11],[12]
7.	Sequence, state chart & activity diagram	Formal Specification-based testing, Graph testing, Heuristic testing		OCL	[64]
8.	UML StateChart	DFS		NLP	[25]
9.	Activity diagram	Activity graph		NLP	[28]
		Hybrid Genetic Algorithm		N/A	[22]
		GenerateTest Scenerio Algorithm	STAD (Synthesis of Test Scenario from activity diagram)	N/A	[45]
10.	Activity diagram	Graph transformation technique and AToM^3 tool		N/A	[56]
11.	Sequence diagram & State chart diagram	Genetic Algorithm		N/A	[68]
12.	Activity & Sequence diagram	DFS		NLP	[70]
13.	Class, Activity & Usecase Diagram	Comparison has been done		N/A	[69]
14.	Sequence diagram	Model-Driven Approach AndroMDA		NLP	[32]
15.	State Diagram	Genetic Algorithm		N/A	[65]
16.	State Machine Diagram	SMTSGA (State Machine To Test Scenario Generation Algorithm) DFS		NLP	[27]
17.	Activity Diagram	Genetic Algorithm Fuzzy Clustering Algorithm		NLP	[34]
		Memorized const path Algorithm		NLP	[61]

Moreover, by prioritizing the test scenarios probability of fault detection can be increased at early stages that will further reduce the testing cost and efforts. the algorithms such as 'Strength Pareto Evolutionary Algorithm', 'Niched Pareto Genetic Algorithm' and 'Pareto Archived Evolution Strategy' are single-objective optimization algorithms. Similarly, 'Non-dominated Sorting Genetic Algorithm', 'Ant Colony Optimization' and 'Particle Swarm Optimization' are multi-objective optimization algorithms [15], [19].

Various case studies have been identified from selected studies against each input model. For example in [13] Automatic Teller Machine (ATM) is used. It is a money dispenser machine to draw money. Y indicates the case studies are generically created and these are publicly available. However, test cases generated from input model have been validated by various case studies as shown in table 8. For sequence, activity and class diagram are validated by ATM, tickets booking system, login system and Library Management System (LMS). Similarly, state machine diagram, use case models are validated by online voting system, driverless train and library management system.

Table 8: Case Study Verification

Sr.N0	Input Model	Case Study	Availability	Reference
1.	Sequence diagram and UML use case Model	Movie Tickets Booking System	Y	[10]
2.	Sequence diagram	Automatic Teller Machine (ATM)	Y	[13]
		Scrum Process	Y	[32]
3.	Use case specification	Body Sense	Y	[60]
5.	Sequence, state chart & activity diagram	ATM	Y	[64]
6.	Activity diagram	ATM with drawl Book ordering system	Y	[22],[11],[12]
		Cell Phone System	Y	[45]
7.	Sequence diagram & State chart diagram	Online Voting system	Y	[68]
8.		Heating Kettle problem	Y	[34]
9.	Activity & Sequence diagram	Login System	Y	[70]
10.	Class, Activity & Use case Diagram	ATM & Library Subject System	Y	[69],[29]
11.	State Diagram	Driverless Train	Y	[65]
12.	State Machine Diagram	Library Management System	Y	[27]

3.3.4 Comparison and Analysis

Test case generation from UML specification is more beneficial because test planning is done side by side in the development stages of software. It helps in reducing cost, time and efforts meanwhile enhancing software quality and reliability. UML models represent the abstract view of the system, by using these models different types of faults can be detected from the system at the early stages of software development [31]. According to the selected studies as shown in table 10, it gives the detailed comparison and analysis of UML diagrams. The first column indicates the UML diagrams used, second column represents the coverage criteria, and the third column shows types of faults that are detected by these diagrams. Similarly, the fourth column signifies the intermediate form used in different approaches. The execution column indicates the execution of test cases, where M represents manually, and A shows automatically. The last column is optimization, which tells whether the generated test cases are optimized or not. The use of sequence diagram as an input model is more effective for test case automation as it reduces the cost and time of development process by detecting operational, dependency, loops, scenario, and interaction faults at initial stages. Moreover, a control flow graph is generated from the sequence diagram that generates the test paths which are efficient for test case generation as it includes complete path coverage based on extracted CFG. A complete path coverage emphasis on both node and edge coverage in the control flow graph. Therefore, the enhanced test coverage in testing contributes to the stability of the software. UML sequence diagram allows interaction between the objects by a sequence of messages in the system [78]. Moreover, evolutionary genetic algorithms used to generate test cases satisfy the branch coverage by producing a smaller number of test cases as compared to conditional algorithm.

3.3.5 Research Productivity

The demographic of the selected studies are presented in figure 12 including the number of researches published from 2011- 2021. Four databases have been selected in a proposed systematic review includes IEEE, ACM, Elsevier, Springer, and others whereas the other consists of Wilsey, Taylor & Francis. An increasing trend in the number of publications can be seen in Fig 8. A detailed list of primary studies is given in Table 9 few well-known journals and conferences have been published in this domain [80]. Most of the publications are observed in IEEE conference.

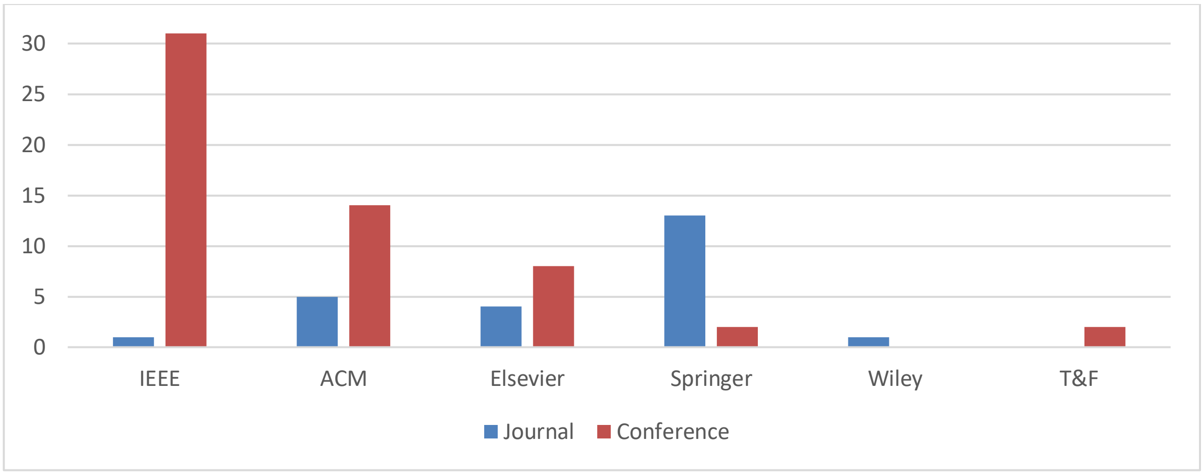


Figure 12. Research Productivity between 2011-2021

3.3.6 Challenges in Test Case Automation

Software testing has always been a challenging task despite the accessibility of several tools and techniques. A software system needs to be tested to verify whether the system is functioning according to requirements or not. It can be done in two ways either manually or automatically. Manual testing does not deliver appropriate results that cause ineffective, inefficient output and effects the quality of a product [15].

- However, automated generation of test cases is not cost-effective if a proper test plan has not been designed. In the automated test case generation process, deciding which part of the system needs to be tested remained a particular challenge for the testers.
- Many researches have been done in previous years that have tackled the challenges of test automation but still, there is room for focusing on the concepts of test automation during the entire testing procedure from the design of the test case leading to test output evaluation [35].
- Before model-based testing, designing test cases was almost ad-hoc.

Test case generation and test specifications of a formal or semi-formal software system can be created from MBT. These models include both structural and behavioral models such as state machine diagram, class diagram, sequence diagram, use cases and activity diagram. Meanwhile, test cases can be generated using more than one model input [14]. Therefore, the generation of test cases using MBT remained much effective in detecting faults at the early stages of development from each test suite. Moreover, test suites are developed in less time by utilizing minimum resources [74].

Table 9: Comparison and Analysis of UML Diagrams

Sr. No	Input Diagram	Coverage Criteria	Fault Detection	Intermediate Form	Execution	Optimization	References
1.	Sequence, State chart & Activity	Branch Coverage, Path Coverage	Interaction, synchronization	NULL	A	Y	[64]
2.	State chart	Path Coverage	N/A	An Extended Flow Graph	A	Y	[57]
3.	Sequence	Prime Path coverage	Operational, Scenario, Interaction	Control Flow Graph	A	Y	[20]
		Message Sequence Path Criteria (MSPC)	Interaction, Operational and Scenario	Concurrent Composite Graph	A	N	[19]
		Object Coverage, Path Coverage	Decision, Loop, Interaction, Synchronization, Message path	Sequence Diagram Graph	A	N	[26]
4.	Sequence Use case	Predicate Coverage	Dependency, Scenario, and interaction	Use case Dependency Graph, Concurrent Current Flow Graph	A	N	[49]
5.	Sequence, Class & State chart	All Path Coverage, All Transition Coverage, All State Coverage	Interaction and operational	Intermediate Test Model PSSETM	A	N	[30]
6.	Activity Diagram	Path Coverage, Activity Coverage	Deadlock Removal	Activity Graph	A	Y	[12],[66] [18],[58]
		Hybrid Coverage Transition Coverage Branch Coverage	N/A	Activity Dependency Graph	A	Y	[33],[50]
7.	Sequence	Message Sequence Path	Interactions	Sequence Flow Chart Message Control Flow Graph	A	Y	[46],[23]

8.	Activity Sequence	N/A	Scenario, Operational, Integration, Concurrent Execution problem	Sequence Graph & Activity Graph	A	Y	[48]
9.	State Chart	Transition Coverage (All Transitions, Round Trip Path and All Transition Pairs)	N/A	State Chart, Intermedia te Graph	A	N	[73]
10.	Class State diagram	State Coverage, Transition Coverage, Path Coverage	N/A	Control Flow Graph	A	N	[24]
11.	State Chart Diagram	State Coverage, Transition Coverage, All path Coverage	N/A	Testing Flow Graph	N	N	[25]
12.	Collaborati on diagram	Path coverage	Avoid redundancy & infinite loop				[52]

3.4 Answers of Research Questions

RQ1: What type of automated testing has been performed by UML diagrams?

With the growing complexity and increasing size of software applications, the importance of test case automation has been increased to reduce the testing effort, cost and time. We have identified 20 research studies corresponding to this category as summarized in Table 6. Software testing is divided into two categories i.e., functional testing and non-functional testing. The first category includes the automated generation of test cases by UML diagrams in functional testing that further consists of system-level testing, unit testing, regression testing, integration testing and acceptance testing. Similarly, the second category incorporates test case generation of non-functional testing by utilizing UML models. Non-functional testing includes performance testing, security testing and measuring the robustness of the software system.

RQ2: What are the techniques and case studies used in existing work for test automation from UML diagrams?

Overall, 18 techniques and 11 case studies have been identified for automated test case generation from UML diagrams as given in Table 8. It has been observed that genetic algorithms and depth-first algorithms have been widely used for automated test case generation by utilizing UML diagrams. Furthermore, it is analyzed that model-based test case automation using various identified techniques are much efficient and cost-effective in automating test case procedure in software testing. As it is done early in the software development life cycle. Moreover, the validation of the above-mentioned algorithms and approaches has been done through identified case studies as summarized in Table 8.

RQ3: Which UML diagram can be considered as a better option for test case automation?

In this category 20 research studies have been highlighted Table 10. According to the above-mentioned studies sequence diagram has been considered as an appropriate option for test case automation because it is found in the existing literature that sequence diagram has ability to detect more faults such as operational, synchronization, loop, scenario and interaction faults during testing procedure. Moreover, it covers the complete path coverage extracted from control flow graph (CFG). A complete path coverage incorporates both the node and edge coverage in a complete graph.

RQ4: How automated test case generation from UML is better as compared to other testing ways?

Software test case automation has been done in various ways such as code-based testing, requirement-based testing and model-based testing. Code - based and requirement-based testing is done later in the developmental stages. Automated test case generation using traditional ways consumes a lot of time and cost however, model-based testing is done at an initial level of software development life cycle. Therefore, model-based testing for automated test case generation remained a better approach as it detects faults early in the software development life cycle.

RQ5: What are the challenges in test case automation which have been overcome by UML?

Test case generation is a crucial but a tiresome task. To develop a high quality and cost-effective system brings up with new challenges, for that the manual testing is not an adequate solution. The reliability of the needed system cannot be managed through it. Moreover, it cannot ensure the coverage of ever-changing system. Therefore, introducing the innovative ways for systemization and automation of testing process are required. This complicated method has become a challenge to ensure the quality delivery of the software. Similarly, proper test plan design is very imperative for cost and time-efficient automated test case generation. Test case design was ad-hoc before model- based testing.

Currently, many test case automation tools are available to generate test cases, test planning, test executation and monitoring. The available tools are used for test case design, test data selection and test data evaluation but for a human tester they remained time consuming and demanding activity [67]. Model based testing has come up with appropriate approaches such as depth first algorithm, genetic algorithms, ant colony optimization and many others as summarized in Table 8. UML Diagrams as an input model have been used to generate test cases automatically for the system. UML modelling has been recognized as a defecto standard both for academia and industry.

RQ6: What are the foremost researches that have been published in the past ten years for test automation?

We have identified 73 research studies from 2011- 2021 as per our selection and rejection criteria, where UML diagrams are used for automated test case generation.

3.5 Conclusion

In this research a systematic literature review (SLR) has been conducted to identify and examined 73 research studies that are published during 2011-2021 in the software testing domain. The identified studies are classified into four categories i.e., test automation from UML, techniques or approaches utilized for test automation using UML, foremost researches in the testing domain and efficient test case automation UML diagram. These categories are thoroughly analyzed to summarize the techniques/approaches for test automation using UML diagrams in the software testing domain. Moreover, further analysis has been done to identify how test case automation using UML is effective as compared to other testing ways. It is concluded from this review that model-based test case automation using identified techniques or approaches is more effective than other testing methods being used in this domain. Moreover, it is highlighted that the use of a sequence diagram and genetic algorithm is very fruitful in software testing as it covers all the paths in coverage criteria and results in a lesser number of test cases.

Research Gap

This sub-section discusses the research gaps and limitations encountered in existing literature. Analysis of 73 selected primary studies (in terms of approaches, algorithms, validation case studies etc.) has been done by following an extensive review process to look for research that provide automated test case generation methods from UML diagrams.

After the deep analysis of selected studies, it was found that there’s no research that focused on U-NSGA-III using single-objective algorithm (extended form of NSGA-III) for test case automation from UML diagrams. There was no single fully automated approach proposed to facilitate test case automation and optimization for statement coverage from UML diagrams. Though partially proposed or semiautomated tools i.e., models and algorithm’s pseudocodes are available in literature, but no framework is proposed using them.

Chapter 4. Methodology

In this section an approach for test case automation and optimization has been discussed. Optimized test cases are generated by using an optimization algorithm UNSGAIII. Test case automation and optimization remained a challenging concern in automated software engineering. However, several optimization techniques have been developed to solve this emerging problem in software engineering field. Moreover, genetic algorithms are well-known optimization methods for test case automation. Genetic algorithm works based on principle of natural selection. However, basic steps of genetic algorithm are shown in figure 13.

Genetic algorithm differs from other traditional techniques in multiple ways

- Encoded representation of variables is directly used by genetic algorithm instead of manipulating them by themselves.
- GA uses stochastic instead of using deterministic operators.
- GA works using blind search mechanism by ignoring complete information and without ignoring the output of the sample

4.1 Testing Objectives

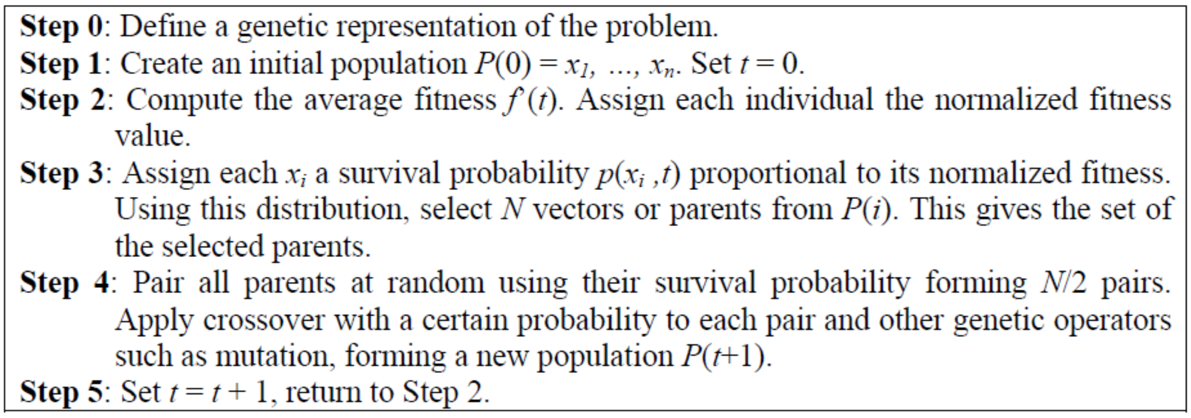
Testing objectives highlight the way the measures that are used to evaluate the ways the testing process is being conducted. In white box or structural testing, if the path coverage has been chosen it is important to know how to check whether the program execution with specific test data has reached to certain criteria or not.

4.2 Fitness Function

Fitness function is also known as objective function. It evaluates the fitness of each chromosome in the population [82]. The objective function is optimized according to the proposed problem to find a solution. Fitness function is composed in such a way that it becomes much effective and efficient in achieving the target goal. In this research, fitness function is designed in such a way that it maximizes the statement coverage with minimum test cases.

$$F(X) = \frac{1}{\sum_{j=0}^{n(x)} W_j g_j(x_i)}$$

Where, W_j(x) represents weight of each node corresponding to the path of control flow graph, and x_i shows the chromosome in the given population. Similarly, $n(x)$ represents number of statements covered by the path. The weight of each node is calculated using a formula i.e., Number of indegree × number of outdegree. Where, indegree (N) denotes the incoming control to the node N whereas, outdegree indicates number of nodes called by succeeding node N.



Step 0: Define a genetic representation of the problem.
Step 1: Create an initial population $P(0) = x_1, \ldots, x_n$. Set $t = 0$.
Step 2: Compute the average fitness $f'(t)$. Assign each individual the normalized fitness value.
Step 3: Assign each x_i a survival probability $p(x_i, t)$ proportional to its normalized fitness. Using this distribution, select N vectors or parents from $P(i)$. This gives the set of the selected parents.
Step 4: Pair all parents at random using their survival probability forming $N/2$ pairs. Apply crossover with a certain probability to each pair and other genetic operators such as mutation, forming a new population $P(t+1)$.
Step 5: Set $t = t + 1$, return to Step 2.

Figure 13. Steps of Genetic Algorithm

4.3 GA Operators

Genetic algorithm parameter includes selection, cross-over and mutation. Performance of genetic algorithm is highly affected by selection method, probability rates of crossover and mutation operators. Crossover and mutation are used to reproduce the offspring in the next generation. After applying these parameters, the entire population gets evolved and new pool of chromosomes is formed. Good operators lessen the search time, fastens the search and reduces the search space substantially.

4.4 Selection

Selection is used to select the fitter chromosome from a pool of a randomly generated population. Selection of fitter chromosomes can be done through different selecting ways such as random selection, roulette wheel selection, rank selection and tournament selection. In this work tournament selection has been used, other selection methods are also explained shortly.

4.5 Roulette Wheel selection

In this selection method chromosomes with better fitness are more likely to be selected. Therefore, in this method selection of chromosomes is based on their fitness values. Higher fitted chromosomes will be selected more time to produce new offspring. The mechanism of roulette wheel selection is shown in figure 14.

4.6 Ranked Fitness Selection

In this selection method fitness values are sorted in ascending or descending order then each chromosome is selected based on ranked its ranked value. Indices of individuals are mapped according to their selection probability by using a mapping function. However, this mapping function could be linear or non-linear, but it does not affect the notion of ranked based selection method. Mapping function significantly evaluate the performance of selection scheme.

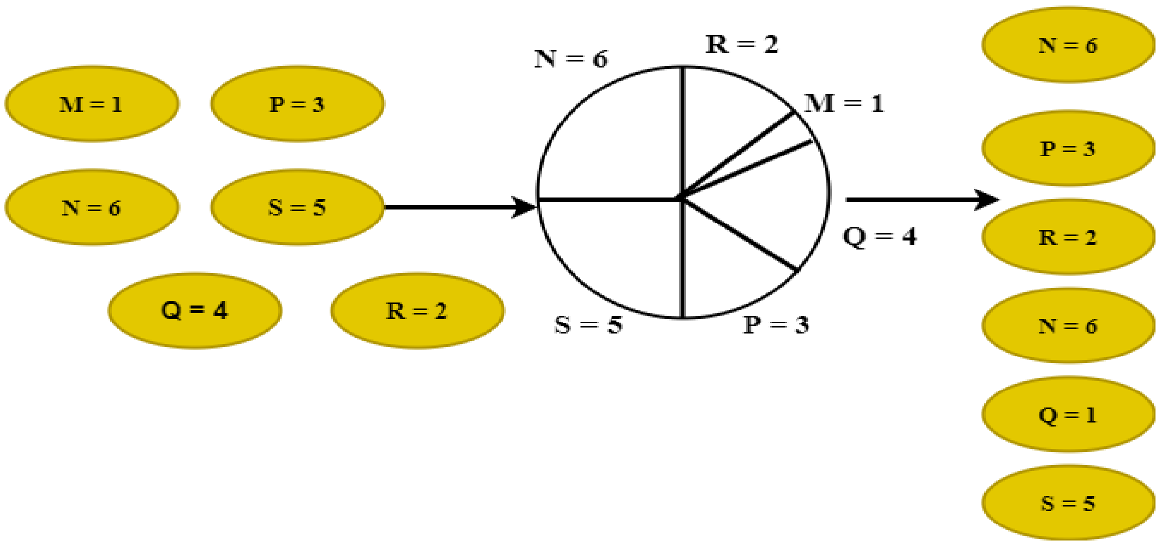


Figure 14. Selection through Roulette Wheel Method

4.7 Tournament Selection

Tournament selection is very simple, efficient, and popular selection method in genetic algorithms. In tournament selection number of individuals (n) are selected randomly from the given population, then these selected chromosomes compete with one another. Individuals with high fitness value win the tournament and are included in the next generation. For binary tournament, the tournament pressure is set 2 that's mean two individuals are randomly selected for the tournament and one with high fitness wins and goes to next generation [83]. Tournament pressure is basically number of individuals who have participated for the competition. Therefore, in this way every chromosome or an individual gets a chance of selection that in return enhances the population diversity. Figure 15 describes the overall mechanism of tournament selection.

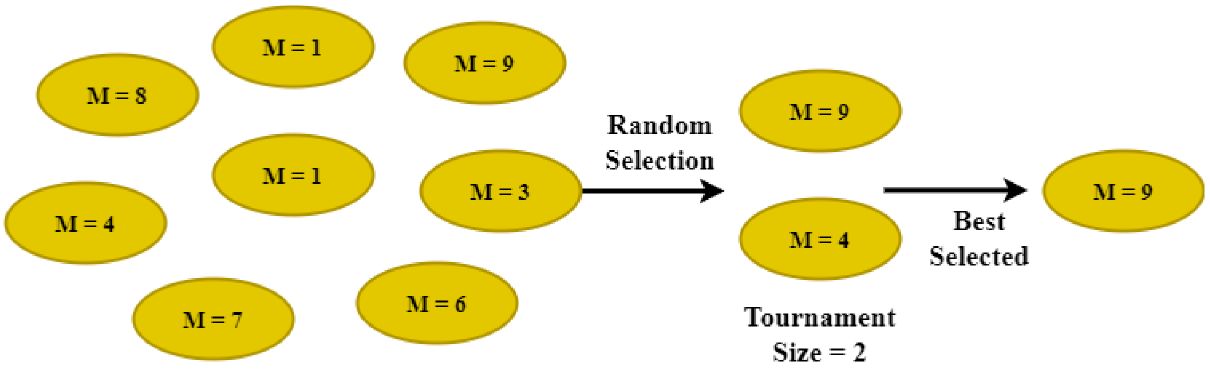


Figure 15. Tournament Selection Mechanism [83]

The advantages of using this selection method are that it does not need requirement of sorting the fitness values of an individuals. It is much efficient towards time complexity, and it ensures the equal participation of all individuals. In the above figure the tournament size is fixed to two i.e., only two individuals are contributing and competing each other. For reproduction, only best chromosome is selected having fitness value nine. However, in this mechanism bigger tournament size leads to loss of genetic diversity which makes the search greedy in nature.

4.8 Crossover Operators

Crossover plays a significant role in providing variations in genetic Algorithm. By copying selected bits from each parent, the crossover operator produces new offspring from two selected parents' string. Crossover probability can be set that ranges between 0 and 1. Crossover operation includes three types of crossover methods i.e, single point, two point and uniform crossover. In the proposed research uniform crossover has been used. In single point or one point crossover a random point is chosen from both parents, by swapping them from cut point produces two new offspring as shown below.

Chromosome 1: 11011|00100110010

Chromosome 2: 11011|11001100110

Offspring1: 1101111001100110

Offspring2: 1101100100110010

Similarly, in two-point crossover two random points are selected from the parent chromosomes, by exchanging genes between selected points produces two new off springs.

4.9 Uniform crossover

Uniform crossover performs better then single- and double-point crossover. In this mechanism bits from two parents are combined uniformly. It takes place by exchanging bits of parents to make them incorporate in the offspring by selecting binary numbers randomly i.e., 0 and 1. In uniform crossover two parents are selected randomly for crossover. By swapping the n genes among them uniformly, produces new population containing traits of parents [84]. Crossover and mutation rates highly effect the variations in new individuals. Higher the crossover rate higher are the chances of completely getting mix the characteristics of both parents into its offspring. Similarly, higher mutation rate produces offspring that has different characteristics which do not exist in the parents.

Chromosome 1: 111010010

Chromosome 2: 100010110

Offspring 1: 110010110

Offspring 2: 101010010

4.10 Mutation Operator

Mutation is the second protagonist in genetic algorithm. It changes the solution by altering the genes of offspring and enhances the diversity of population as well. it also works based on random changes. However, like crossover rates, mutation rates also need to be set for reproducing new population. Mutation operator prevents premature convergence by jumping out of local solution or suboptimal solution [85]. Various mutation operators are being used for automated test case generation and optimization such as inversion mutation, swap mutation, scramble mutation and insert mutation however, in this research bit flip mutation operator has been used with varying mutation probabilities as it works effectively with binary numbers. Bit flipping included changing of bits from 1 to 0 and 0 to 1. Mutated genes are shown below.

Offspring1: 1101111001110110

Offspring2: 1101100100100010

Bits that are highlighted in black color are mutated genes of offspring1 and offspring 2.

4.11 Stopping Criteria

Stopping criteria signifies how search effort should be stopped. It could be either defined by number of generation or by the specified targeted path.

4.12 Unified Non- Sorting Genetic Algorithm (U-NSGA-III) Approach

In this research one of the variants of genetic algorithm i.e., U-NSGAIII has been proposed for white box test case generation and optimization. UNSGAIII works well with one and more than one objective problems. NSGAIII is extended to UNSGAIII, because for single objective problems NSGAIII works with random selection process and with very small population size. Similarly, for successful multi and many optimizations problems normalization and niching operators are very essential but these operators of NSGAIII become useless for mono objective problems. Therefore, a unified approach has become efficient for both single and multi-objective optimization problems. The proposed approach makes niching and normalization operator non-functioning for mono objective and functioning for multi to many optimization problems. However, to solve the above-mentioned mono objective problem large number of population size N has been used which is greater than number of reference point H i.e, N>H by using UNSGAIII approach.

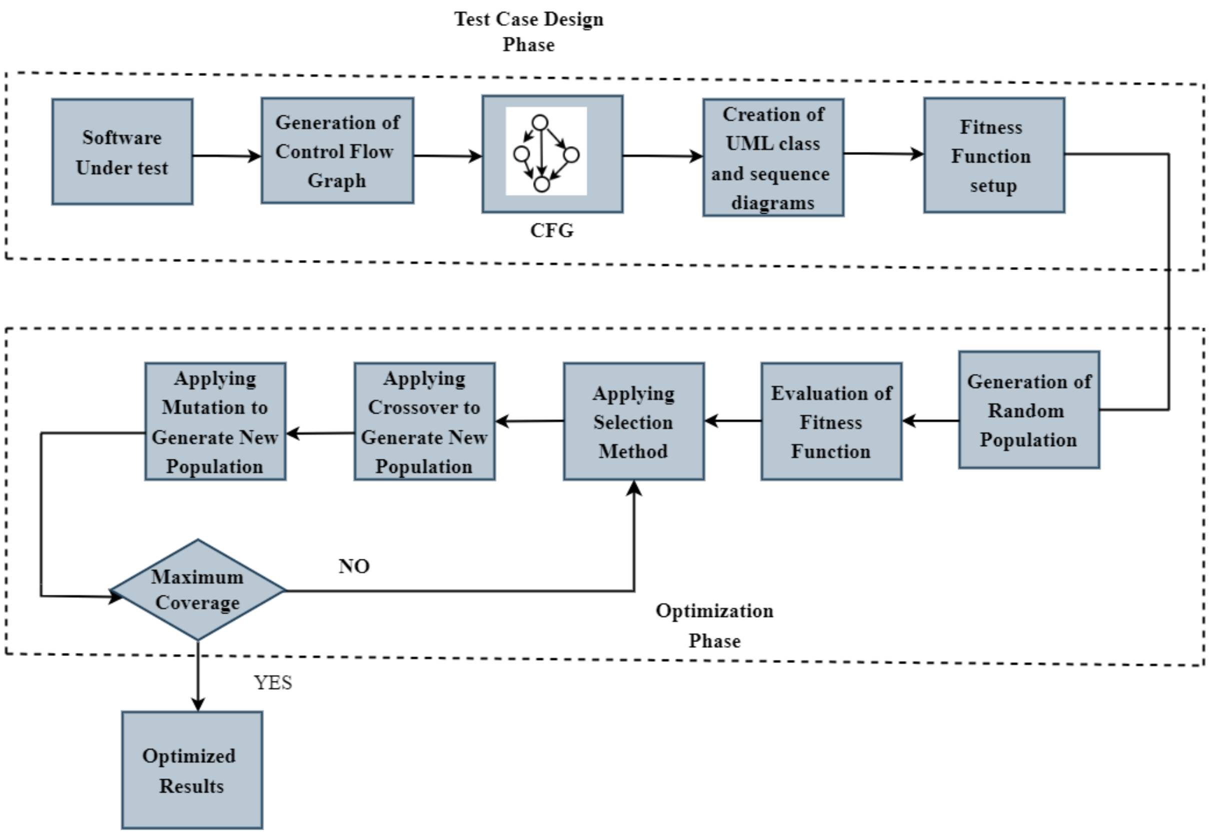

Test Case Design Phase
Software Under test
Generation of Control Flow Graph
CFG
Creation of UML class and sequence diagrams
Fitness Function setup
Generation of Random Population
Evaluation of Fitness Function
Applying Selection Method
Applying Crossover to Generate New Population
Applying Mutation to Generate New Population
Maximum Coverage
NO
YES
Optimization Phase
Optimized Results

Figure 16. Proposed Flow Diagram

Chapter 5: Experimentation and Results

For implementing the proposed approach and comparing it with existing algorithm, different experiments are performed. Two case studies have been selected for performing experiments and the focus of all these experiments is on reduction of test cases with maximum statement coverage. Implementation of unified non sorting genetic algorithm is done on each case study. The discussion about experiments and their results has been done in this chapter. Experimental flow is shown in figure 17.

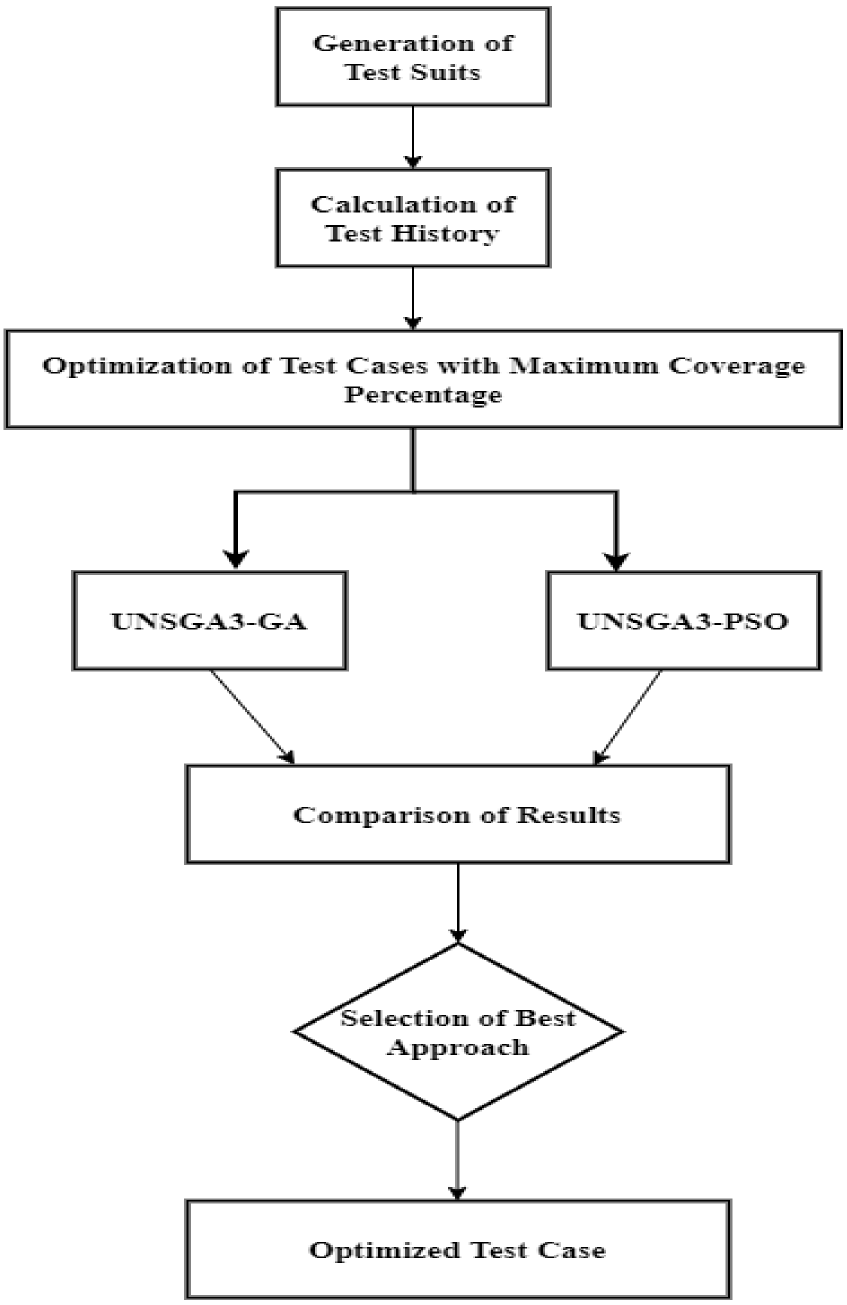

Generation of Test Suits
Calculation of Test History
Optimization of Test Cases with Maximum Coverage Percentage
UNSGA3-GA
UNSGA3-PSO
Comparison of Results
Selection of Best Approach
Optimized Test Case

Figure 17. Experimental Flow

Case Study 1: Automatic Teller Machine (ATM)

For evaluating and performing comparison of our approach with other approaches that have already been implemented, ATM case study is selected. This specific case study has been selected because it is not difficult for the readers to understand its results and it has been widely used in software testing domain. A sequence diagram of ATM withdrawal and deposit system has been created using Draw.io for the presentation of a case study as shown in Figure 20. Moreover, based on sequence diagram a control flow graph (CFG) is created as shown in Figure 18. Genetic algorithm is applied to generate test case and to optimize them. Weights of each node are calculated that are used in the fitness function to calculate fitness of each test case. ATM system consists of GUI based transaction mechanism as shown in UML class diagram Figure 19. The ATM card is inserted by the user then it gets validate using pin and account number. However, on successful login user can perform different transaction actions such that withdrawal, money deposit, balance inquiry and received receipt.

ATM should be able to provide following services to the customers:

A user should be able to withdraw cash upon his request and received an approval message form the relevant bank before the cash is dispensed. If balance is less than an entered amount, then an error message should be displaced on the screen. Similarly, if a user wants to deposit cash, he should be able to transfer it between the two linked accounts. A user must be able to check balance through the card. If the bank finds that the pin entered by the customer is invalid, the customer needs to enter the pin again before the transaction proceed. If the customer is not able to enter right pin till three tries, then the card will be blocked by the machine. For each successful transaction the ATM provides a receipt showing the transaction date, time and type of transaction.

Generation of Test Scenario

All possible test scenarios are generated from the CFG before generating test cases. The possible identified test paths are as follows:

Test Path 1: 1->2->3->4->5->7->8->9->10->12->13->14->16->20

Test Path 2: 1->2->3->4->5->7->8->9->10->12->13->17->20

Test Path 3: 1->2->3->4->5->7->8->9->11->15->18->19->20

Test Path 4: 1->2->3->4->5->7->8->20

Test Path 5: 1->2->3->4->6->20

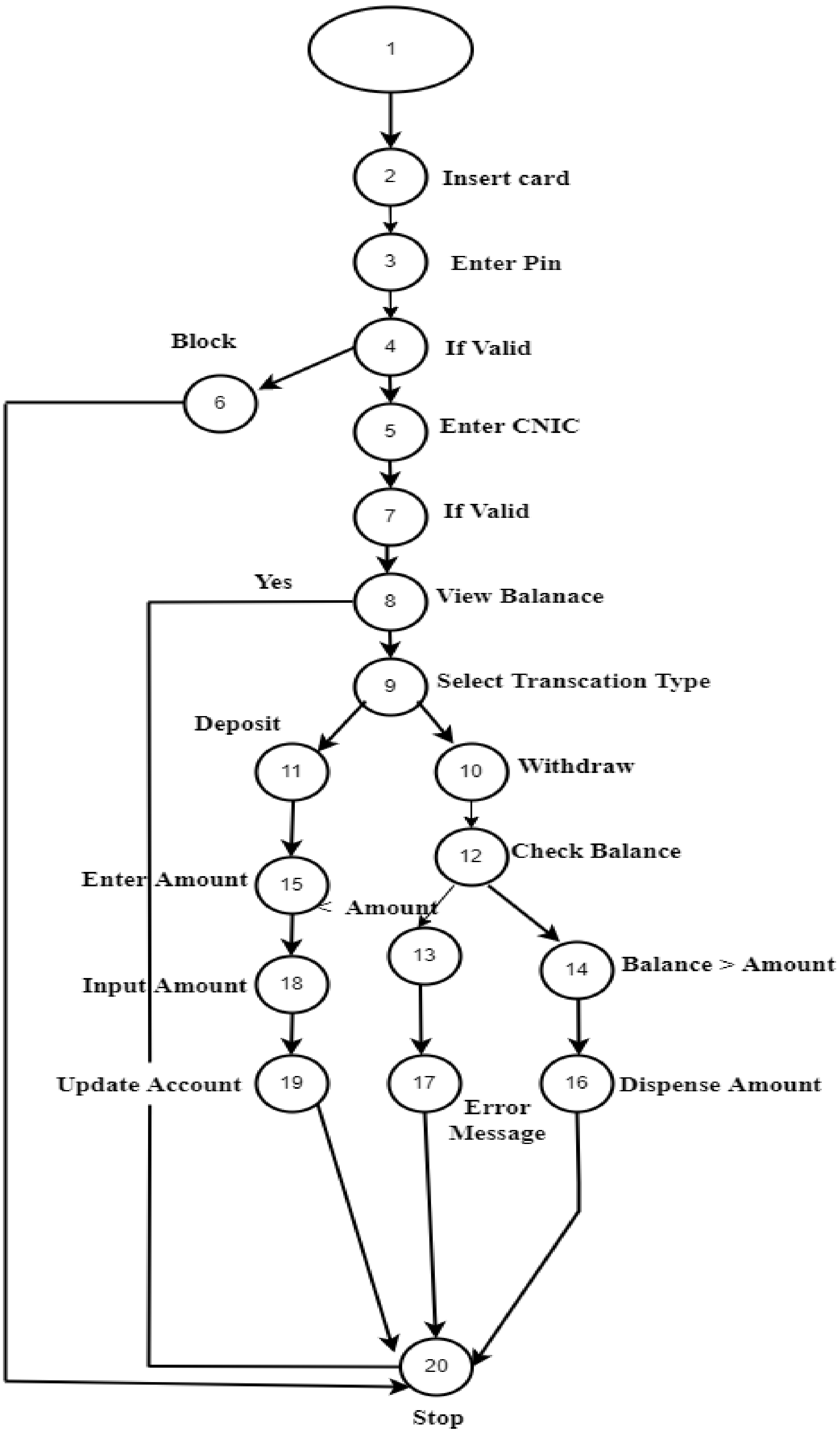


Figure 18: Control Flow Graph (ATM Machine)

Cyclomatic Complexity

Cyclomatic complexity of the above CFG is given as:

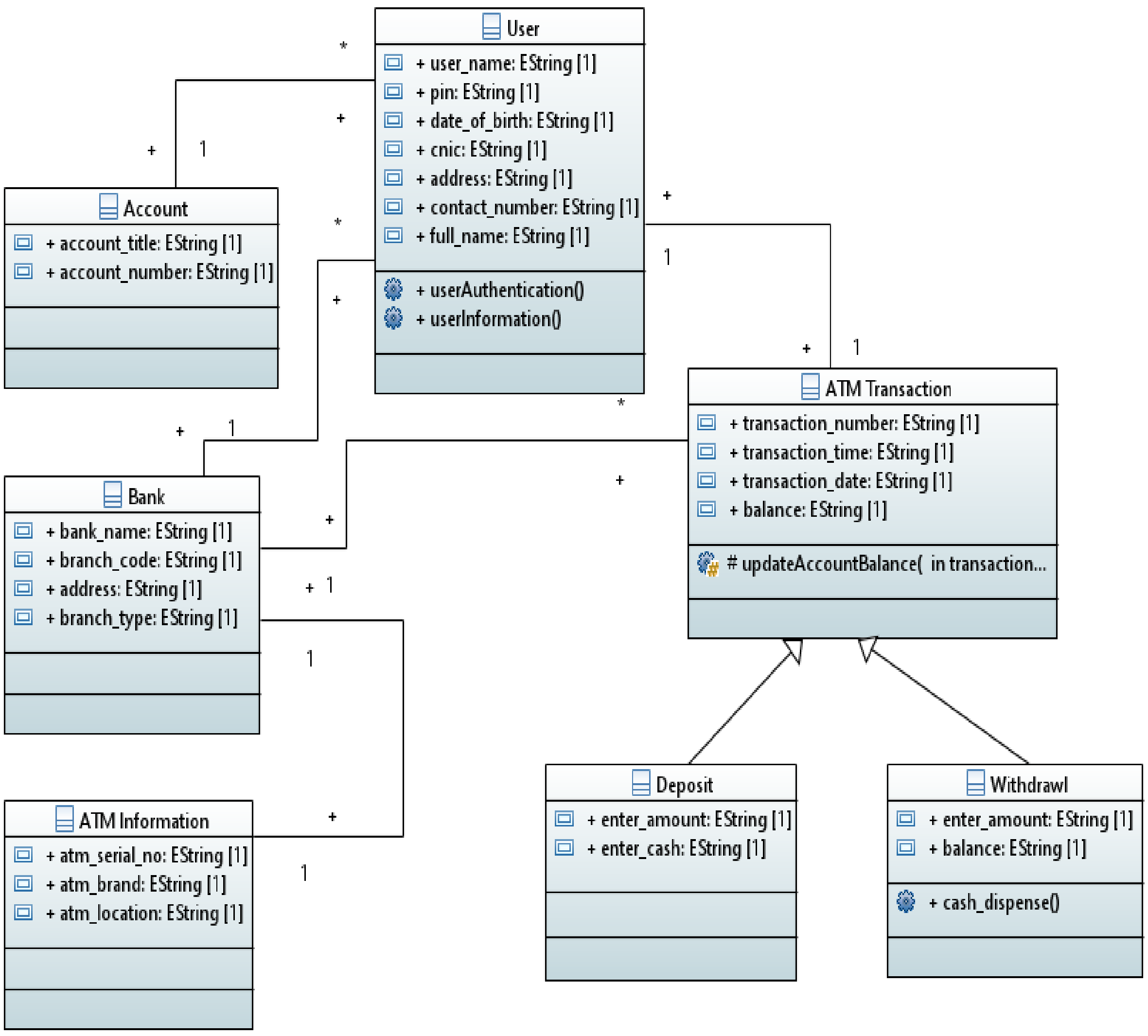
$$V(G) = E-N+2$$

$$V(G) = 24-20+2$$

$$= 6$$

Where, E represents number of edges in the graph and N denotes number of nodes in the above graph.

Class Diagram



User
+ user_name: EString [1]
+ pin: EString [1]
+ date_of_birth: EString [1]
+ cnic: EString [1]
+ address: EString [1]
+ contact_number: EString [1]
+ full_name: EString [1]
+ userAuthentication()
+ userInformation()
Account
+ account_title: EString [1]
+ account_number: EString [1]
Bank
+ bank_name: EString [1]
+ branch_code: EString [1]
+ address: EString [1]
+ branch_type: EString [1]
ATM Information
+ atm_serial_no: EString [1]
+ atm_brand: EString [1]
+ atm_location: EString [1]
ATM Transaction
+ transaction_number: EString [1]
+ transaction_time: EString [1]
+ transaction_date: EString [1]
+ balance: EString [1]
# updateAccountBalance(in transaction...
Deposit
+ enter_amount: EString [1]
+ enter_cash: EString [1]
Withdrawl
+ enter_amount: EString [1]
+ balance: EString [1]
+ cash_dispense()

Figure 19: Class Diagram

Sequence Diagram

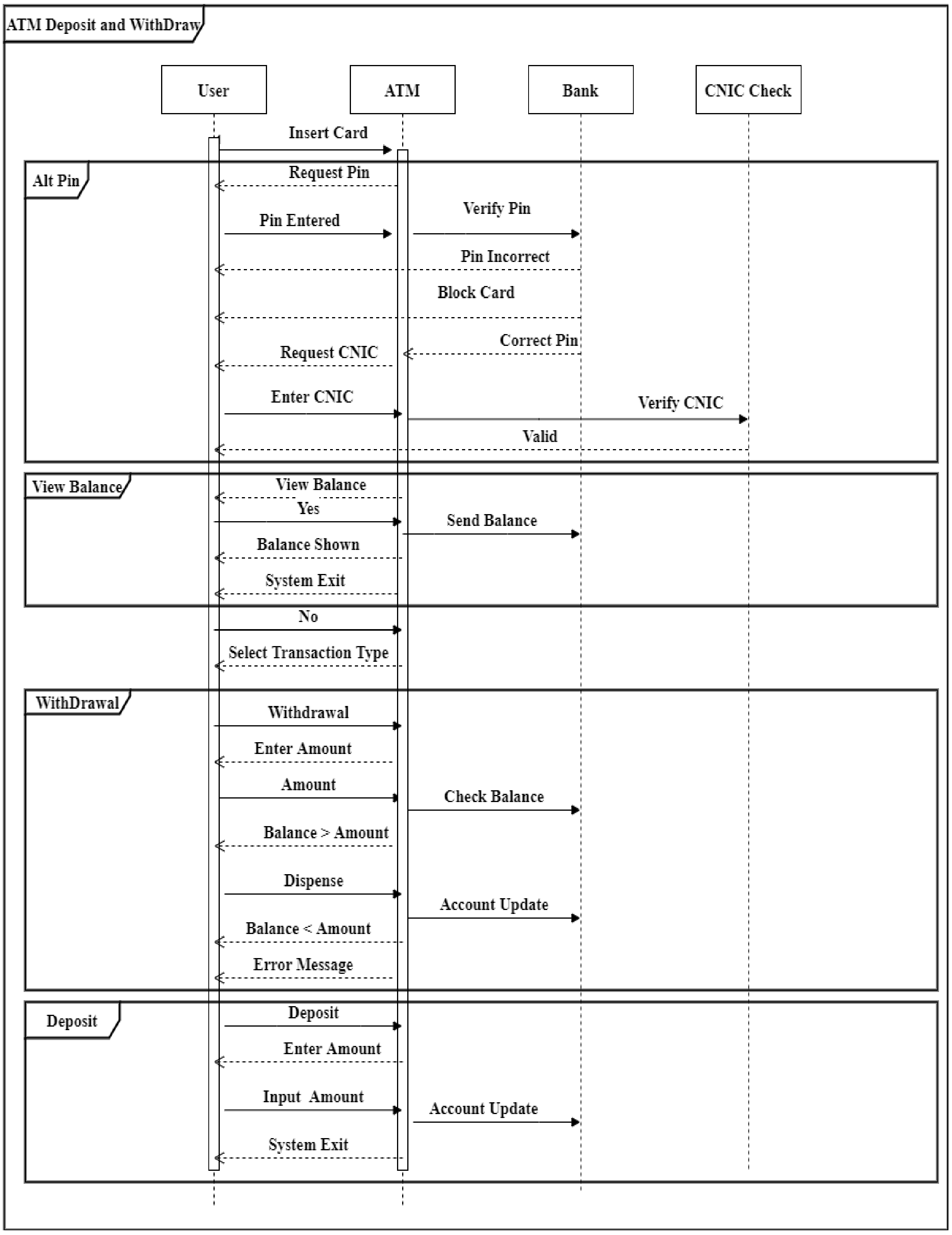


Figure 20: Sequence Diagram

Test Cases

After identifying all the possible test path from the control flow graph, test cases are generated from the respective test paths. Test cases are generated based on coverage activity. Generated test cases are shown in Table 10.

TC1: {Input: Insert card, pin invalid, block card, display screen}

TC2: {Input: pin valid, CNIC valid, view balance (yes), output: Display balance}

TC3: {Input: pin valid, CNIC valid, view balance (no), cash withdraw (yes), balance > amount (yes), output: cash dispense, update an account, exit screen

TC 4: {Input: pin valid, CNIC valid, view balance (no) cash withdraw (yes), balance < amount (yes) output: show error message, exit screen.

TCS5: {Input: pin valid, CNIC valid, view balance (no) cash withdraw (no), cash deposit (yes), output: Input amount, update an account, exit screen

Observed Test cases

Table 10: Observed Test Cases

Test Case #	Pin number	CNIC	Entered Amount	Balance in account	Expected Output	Actual Output
1	7630 (invalid)	valid	1000	5000	Invalid pin	Block Card
2	7360 (valid)	valid	1000	5000	Valid pin	Successful
3	7360	valid	3000	5000	Dispense Cash, Update Account	Dispense cash Successfully, Account Updated
4	7360	valid	6000	5000	Insufficient Balance, Error Message	Show an Error Message
5	7360	valid	2000	8000	Show Balance	Show Balance

Two examples of system under test (SUT) have been proposed for this research.

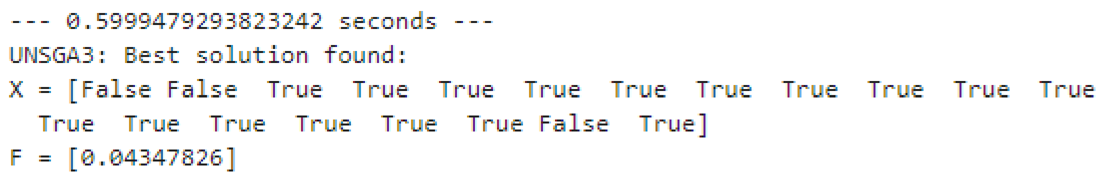
Experimental Results for SUT1

Test case generation method was tested by designing a first system SUT1. GA is an evolutionary approach is developed on the idea of population of chromosome and their recombination. GA is free of derivation and solves complex problems quite easily. UNSGA3 gives better results in minimum time. A total of five experiments with different population size have been carried out that have covered 95% of program statements. Therefore, an algorithm is run multiple times to obtain the maximum coverage of program statements by using lesser number of iterations. Manually calculated coverage for SUT1 is shown in table 12.

Table 11. Control Parameters of U-NSGA-III

Sr #	Parameters Names	Assigned Values
1.	Number of Generations	100
2	Population Size	50
3	Crossover Probability	0.8
4	Mutation Probability	0.2

When Population size10 and number of generations are 100



```
--- 0.5999479293823242 seconds ---
UNSGA3: Best solution found:
X = [False False  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True False  True]
F = [0.04347826]
```

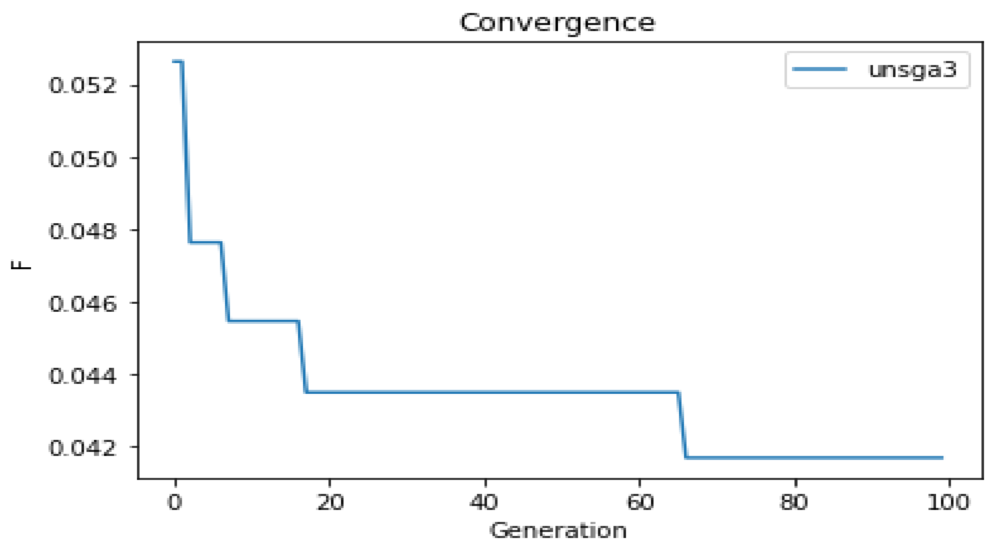


Figure 21. Achieved Statement Coverage with initial population

When Population size is 20 and number of generations are100

```
UNSGA3: Best solution found:
X = [False  True  True  True  True False  True  True False  True  True  True
  True  True  True  True  True  True  True  True]
F = [0.04761905]
```

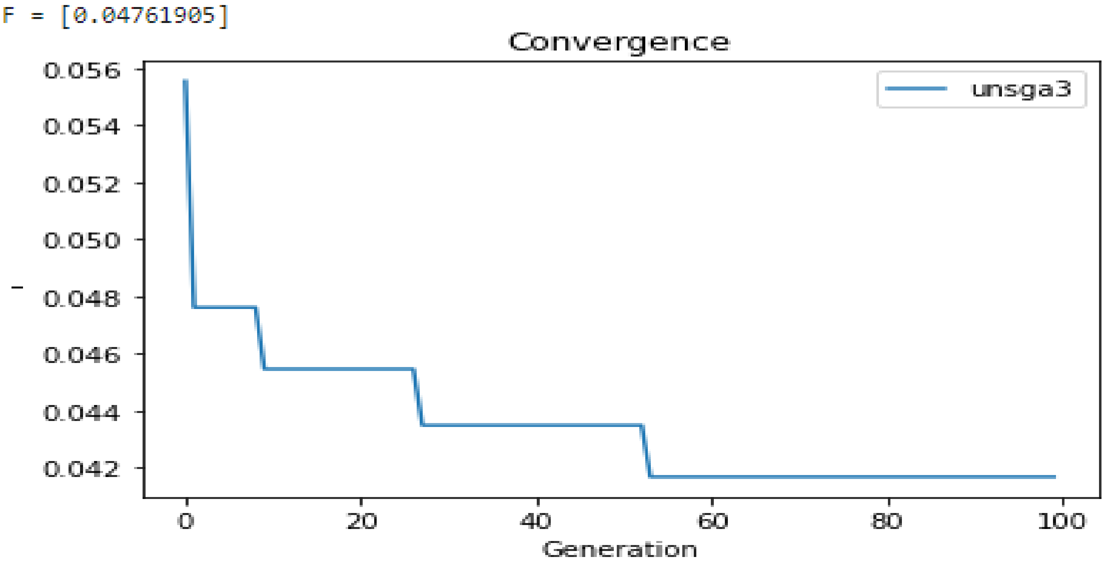


Figure 22. Achieved Statement Coverage with Population Size 20

When Population size is 30 and number of generations are 100

```
--- 0.4959535598754883 seconds ---
UNSGA3: Best solution found:
X = [ True  True  True  True  True  True  True  True  True  True  True  True
  True False  True  True  True  True False  True]
F = [0.04347826]
```

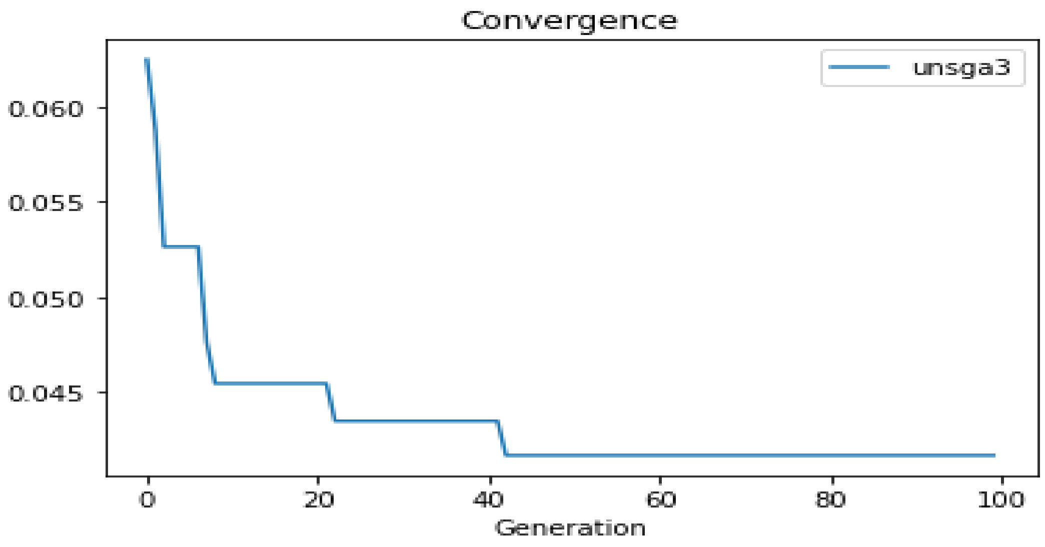


Figure 23. Achieved Statement Coverage with Population Size 30

When Population size is 40 and number of generations are 100

```
--- 0.496349573135376 seconds ---
UNSGA3: Best solution found:
X = [ True  True  True  True  True  True  True  True  True  True  True  True
  True False  True  True  True  True False  True]
F = [0.04347826]
```

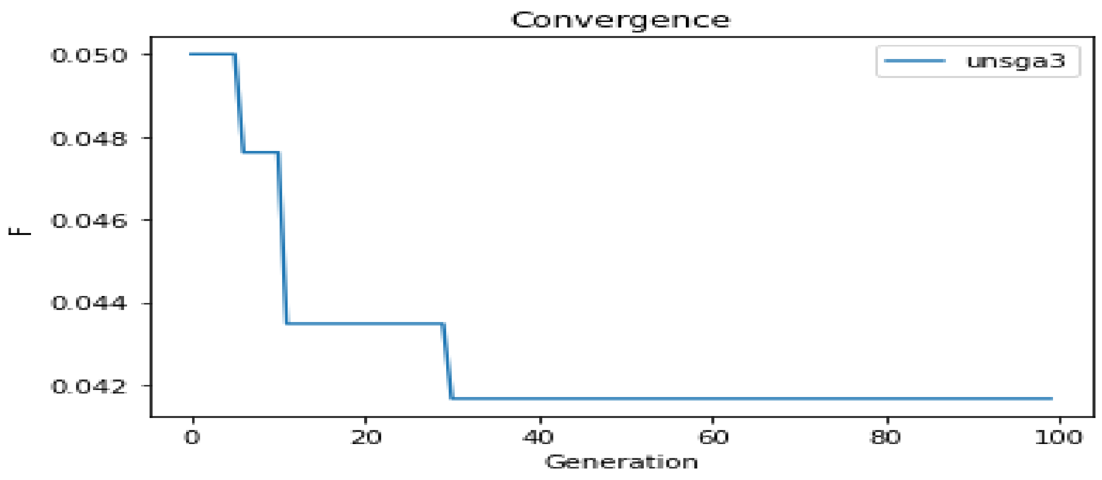
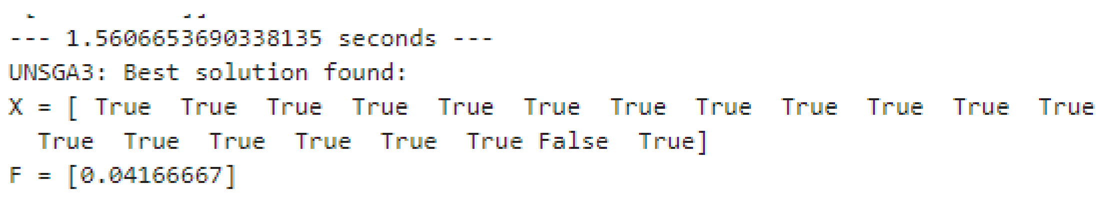


Figure 24. Achieved Statement Coverage with Population Size 40

When Population size is 50 and number of generations are 100



```
--- 1.5606653690338135 seconds ---
UNSGA3: Best solution found:
X = [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True False  True]
F = [0.04166667]
```

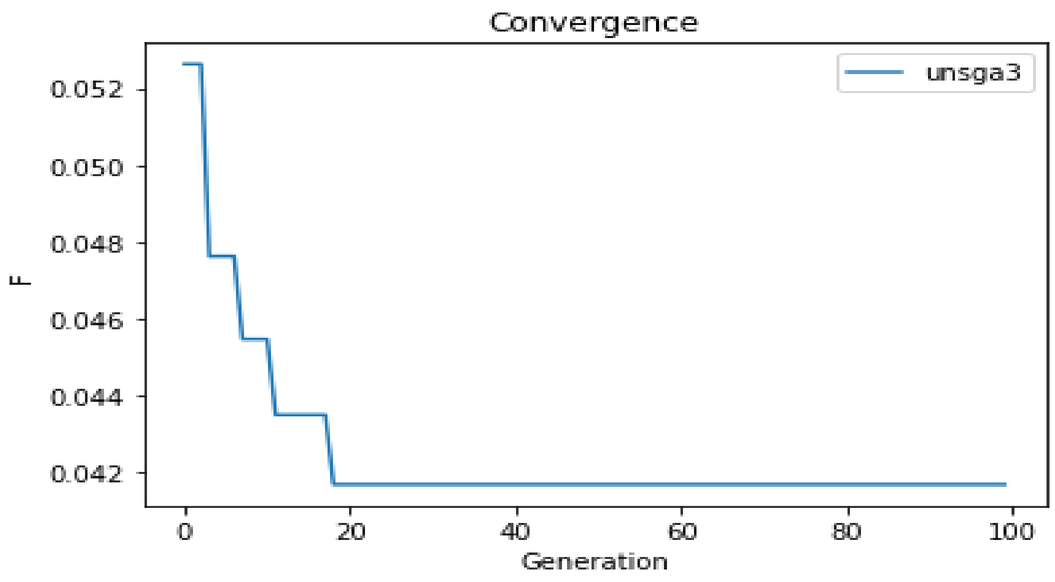
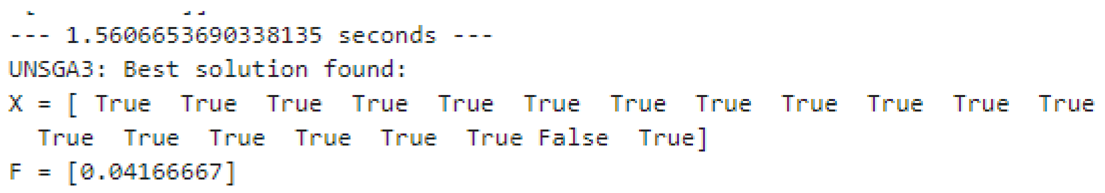


Figure 26. Achieved Statement Coverage with Final Population Size

Table 12. Coverage Percentage Manually Calculated

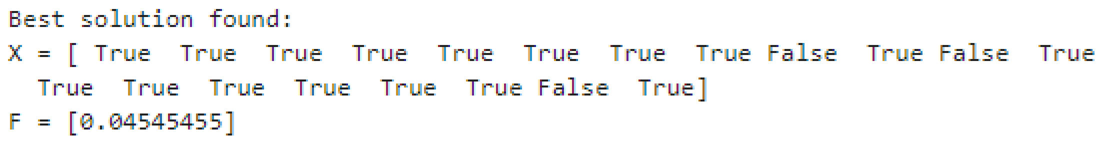
Test Case ID	Coverage %	Statement Coverage Indicator
1	65%	11111111010100100110
2	65%	*11111110000000000010*
3	70%	11111100000000000011
4	70%	*11110000000000000011*
5	80%	11111111010101010010
6	95%	11111111101000001010

Genetic algorithm (GA) works on the principle of natural selection and genetics, inspired by the biological evolution of human being. Genetic algorithms are considered as well-known optimization tool and are very easy to use. Generally, GA consists of two main steps, in the first process individuals based on fitness values are selected from the population using specific selection method. In the second step production of new individuals take place through cross over and mutation techniques. Therefore, by doing comparison of UNSGA3 with GA, it is observed that by applying UNSGA3 95% statement coverage has been achieved with population size of 50. Whereas, by applying GA using same parameters 85% statement coverage has been achieved as shown in figure 26.



```
--- 1.5606653690338135 seconds ---
UNSGA3: Best solution found:
X = [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True False  True]
F = [0.04166667]
```

Best solution Found by GA



```
Best solution found:
X = [ True  True  True  True  True  True  True  True False  True False  True
  True  True  True  True  True  True False  True]
F = [0.04545455]
```

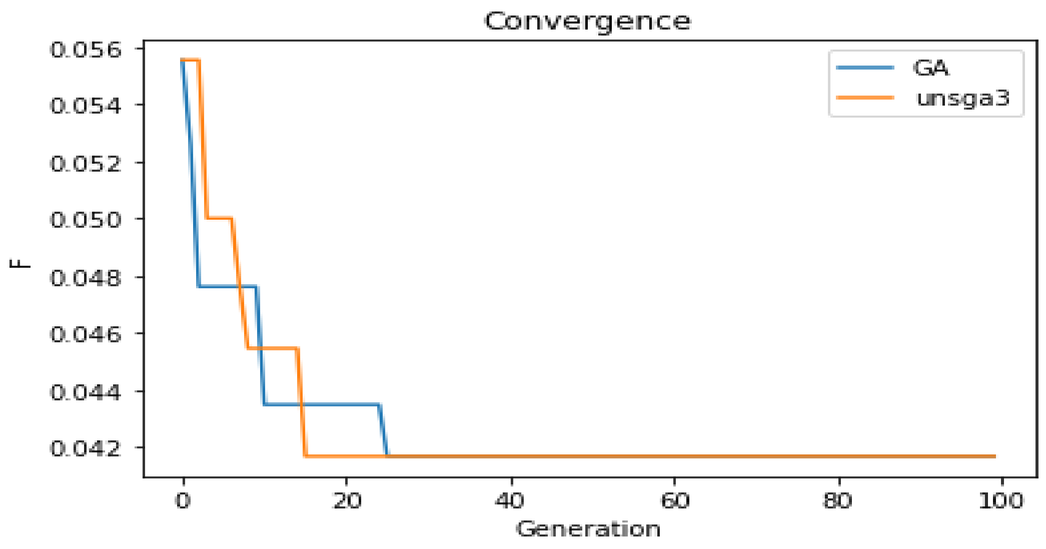


Figure 27. Comparison of UNSGA3 with GA

Comparison with Particle Swarm Optimization (PSO)

The proposed UNSGA3 algorithm tuned with PSO is used for optimization of UNSGA3 structure. PSO is an evolutionary algorithm that is based on the social behavior. It repeatedly attempts for improving the candidate solution in correspondence to a specific measure of quality. Random velocity is assigned to each particle and every particle is drawn to the fitness value that is attained by that particular candidate particle. For optimizing the performance and finding the best solution, tuning of parameters has been used [86]. But there are some deficiencies in PSO as well, for example, for solving the scattered problems in which search space is refined, PSO is not a good choice. Therefore, this algorithm has achieved 65% coverage of an optimized test case.

In table 13 controlled parameters of PSO are shown and the results of case study are presented below.

Table 13. Control Parameters of PSO

Sr #	Parameters Names	Assigned Values
1.	Number of Generations	100
2	Population Size	50
3	Weight of Inertia	0.9
4	Co-efficient of personal Learning	0.6
5	Co-efficient of global learning	0.6
7	Velocity Max	0.3
8.	Velocity Min	0.1

```
Best Solution:  [1 1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 1 0]
```

```
F = [0.04166667]
```

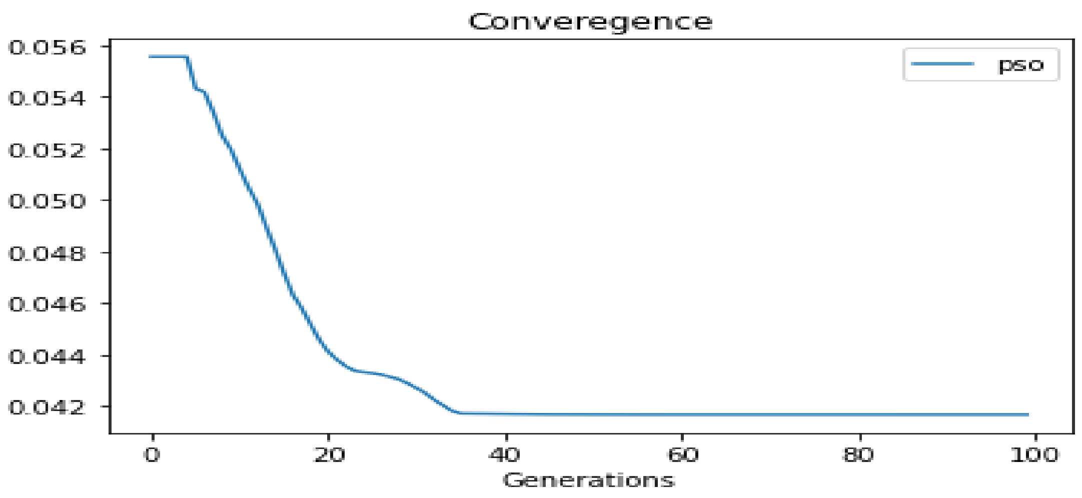


Figure 28. Statement Coverage with Final Population size

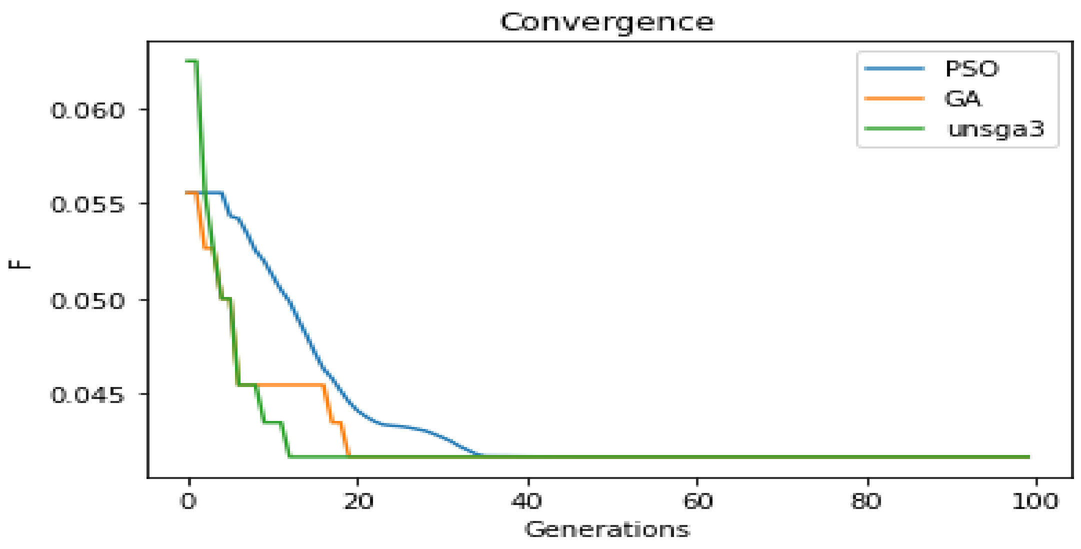


Figure 29. Comparison with GA and PSO

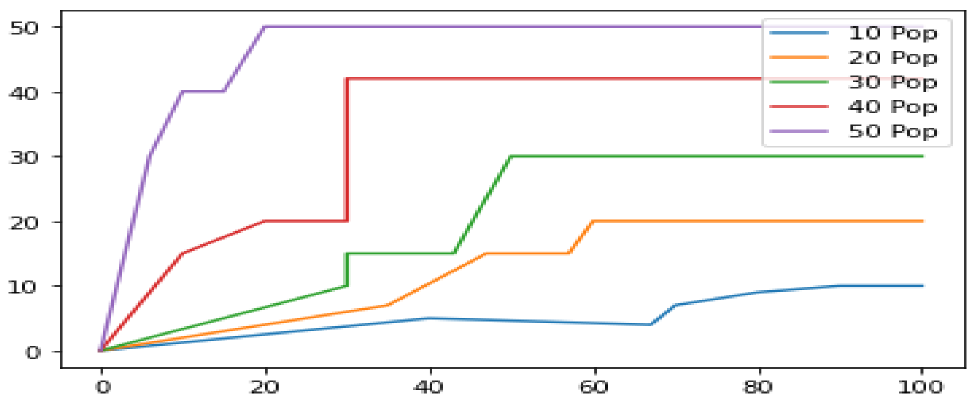


Figure 30. Comparison of algorithm’s convergence speed with different population size

Case Study 2: Library Management System (LMS)

The implementation of our approach has also been done using another case study i.e., Library Management System (LMS) as shown in figure 32. A control flow graph consists of number of nodes and edges. Nodes represent the statements and edges show the flow of the program from one statement to another. Many institutions and organizations utilize LMS to manage their books and keeping tracks of their members who borrow, issue, and return the books available in the library. A library management system, or LMS, allows members of an organization to find, borrow, and return books. They can even reserve selected books. LMS allows the user to find a book using search engine and a user can also locate a specific rack where the book is placed. This system only provides access to its registered members i.e., librarian and a member. Therefore, this LMS supports the following features:

- It allows member to search a book
- It allows member to reserve a book
- It allows member to issue a book
- It allows member to return a book

Similarly, LMS allows the librarian to create, update and delete the record. Librarian can also register a book, update and delete the record. Two types of users work with the system, one is librarian and another one is registered member. Both types of users interact with the system in their own way having different responsibilities. The librarian uses the system with provided login credentials and perform his/her task by managing books and keeping the record of members and books as well. Similarly, a member can issue, reserve and return a selected book by using valid authentications. Furthermore, a sequence diagram has been created for library management system it consists of three actors a user, librarian and a system as shown in figure 31. It shows that how a registered member and a librarian interact with the system. Class diagram for LMS is created that consists of various classes like login, member, librarian, book and a class of search book. Each class has its own attributes and is associated with another class with many to many and 1 to many multiplicities. A login class has its attributes username and password, a member class has its attributes member name, password, registration number, CNIC, and email. Similarly, a book class contains its attributes book title, ISBN number, volume and addition.

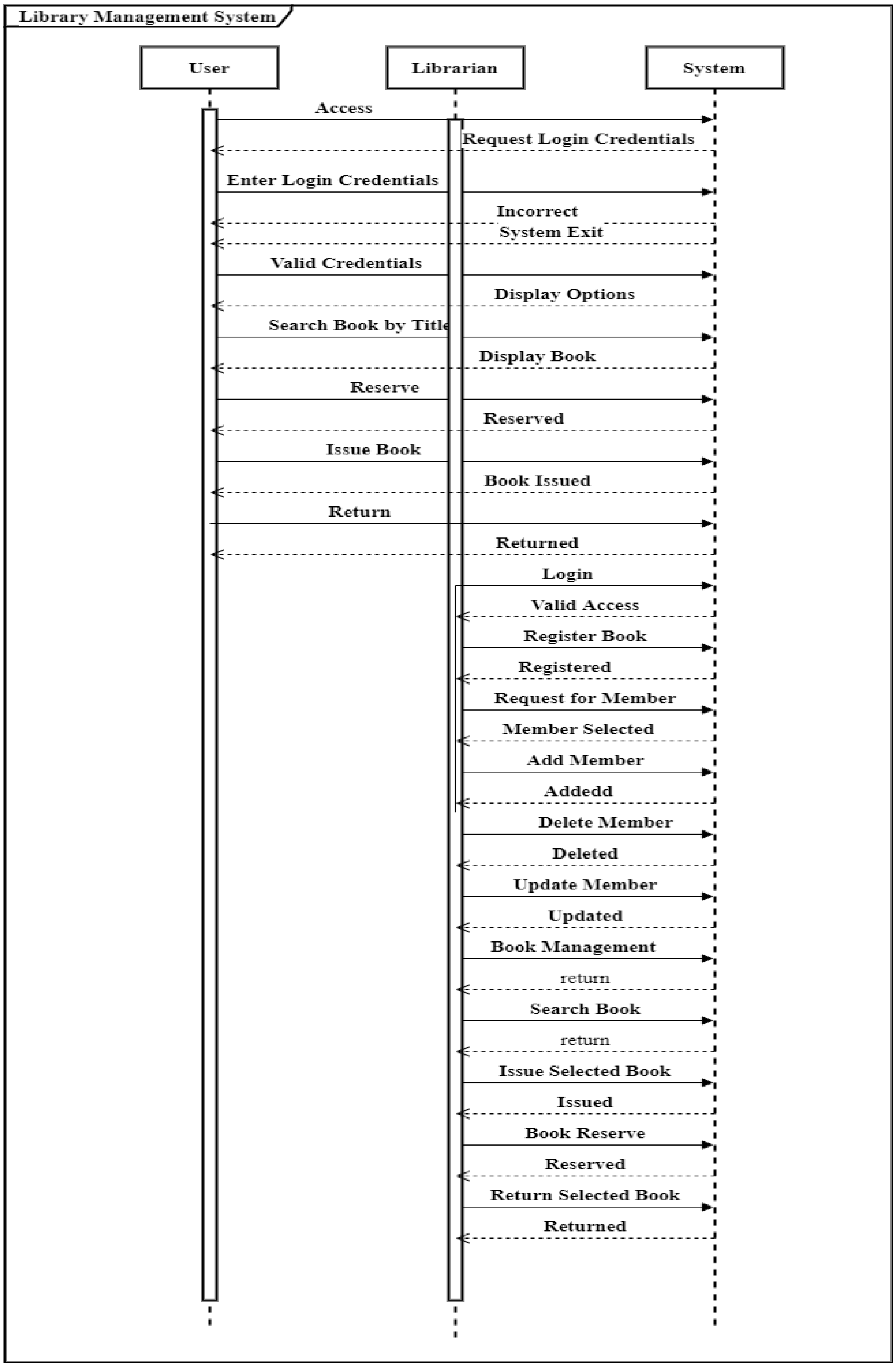


Figure 31. Sequence Diagram of LMS

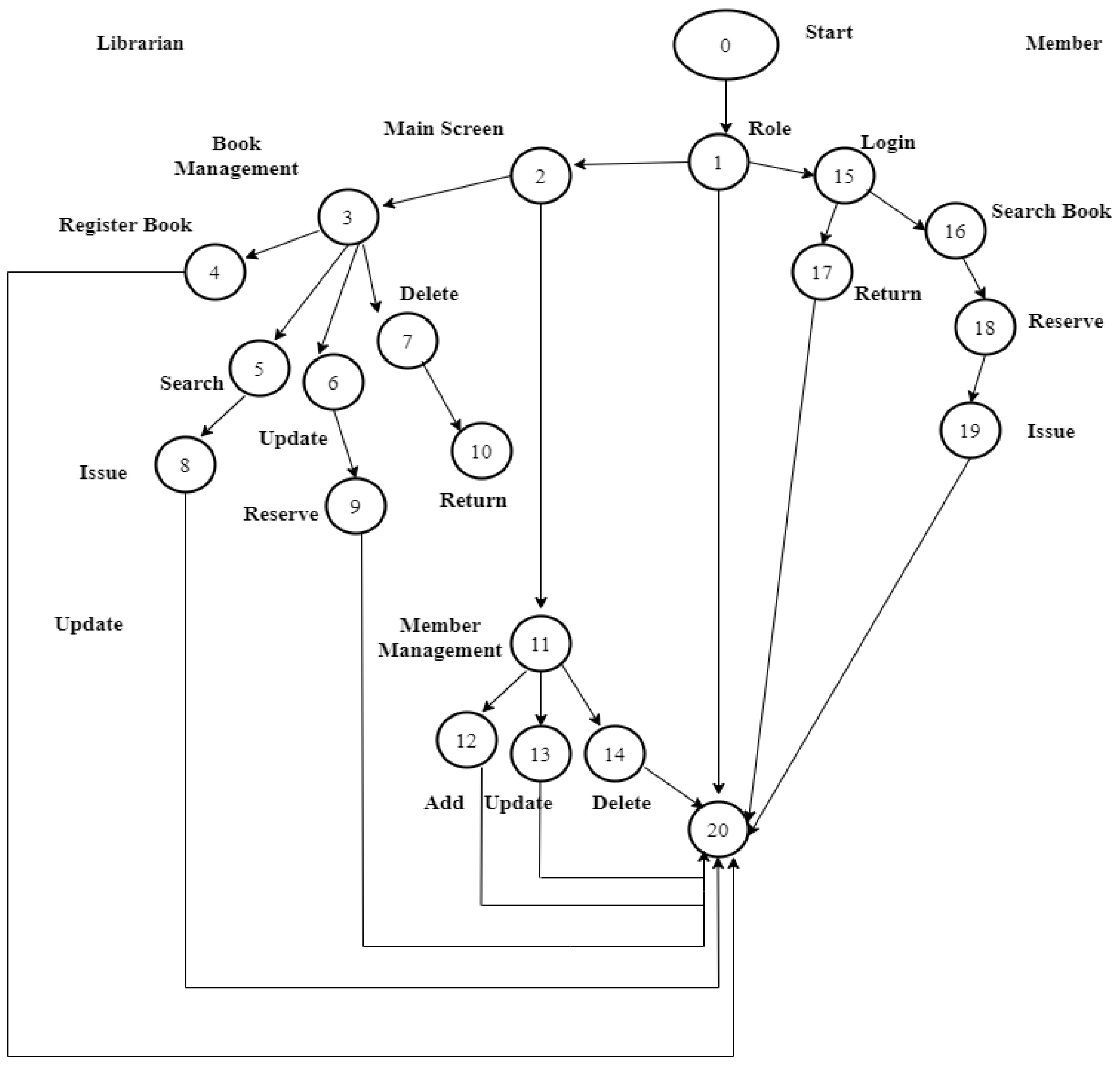


Figure 32. Control Flow Diagram of LMS

Cyclomatic Complexity

Cyclomatic complexity of the above CFG is given as:

$$V(G) = E-N+2$$

$$V(G) = 33-21+2$$

$$= 14$$

Where, E represents number of edges in the graph and N denotes number of nodes in the above graph.

Test case Scenario

0->1->2->11->12->20

0->1->2->11->13->20

0->1->2->11->14->20

0->1->2->3->4->20

0->1->2->3->5->8->20

0->1->2->3->5->9->20

0->1->2->3->5->10->20

0->1->2->3->6->20

0->1->2->3->7->20

0->1->15->16->18->20

0->1->15->16->19->20

0->1->15->17->20

0->1->20

After identifying all the possible test path from the control flow graph, test cases are generated from the respective test paths. Test cases are generated based on coverage activity. Observed test cases are shown in table 14

TC1: {Input: login, Invalid credentials, Output, show an error, display screen}

TC2: {Input: login, valid credentials, main screen, book management, register book, output: Book registered, display screen}

TC3: {Input: login, valid credentials, main screen, book management, update book, output: Book updated, display screen}

TC4: {Input: login, valid credentials, main screen, book management, delete book, output: Book deleted, display screen}

TC5: {Input: login, valid credentials, main screen, book management, search book, issue book output: Book issued, display screen}

TC6: {Input: login, valid credentials, main screen, book management, search book, reserve book, output: book reserved, display screen}

TC7: {Input: login, valid credentials, main screen, member management, delete member, output: member deleted, display screen}

TC8: {Input: login, valid credentials, main screen, member management, register member, output: member registered, display screen}

TC9: {Input: login, valid credentials, main screen, member management, update member, output: member updated, display screen}

TC10: {Input: login, valid credentials, member screen, book search, reserve book, output: book reserved, display screen}

TC11: {Input: login, valid credentials, member screen, search book, reserve book, issue book, output: book issued, display screen}

TC12: {Input: login, valid credentials, member screen, return book, output: book returned, display screen}

Table 14. Observed Test cases of LMS

Test Case #	Test cases	Expected Output	Actual Output
1	Enter valid login credentials	A main screen should be displayed	Main screen displayed
2	Enter invalid login credentials	Main screen should not be displayed	Successful
3	Click on Add button	User should be able to add book in the database	Book added successfully
4	Click on update button	User should be able to update the record	Record updated successfully
5	Click on delete button	User should be able to delete the entered data	Book deleted successfully
6	Click on Add Button	Admin should be able to add new member	New member added successfully
7	Click on update button	Admin should be able to update database	Database updated successfully
8	Click on delete button	Admin should be able to delete member	Member deleted successfully
9	Click on search button	System should be able to display the details of books against the entered title or ISBN number	List of entitled books are displayed
10	Book Issue: “Click on issue book icon”	System should display details of available books	List of available books displayed
11.	Book Reserve: “click on book reserve icon”	System should display details of reserved books	List of reserved books displayed

12.	Click on search button	Member should be able to search the books	List of searched books displayed
13.	Click on issue book icon	Member should be able to issue book	List of issued books displayed
14.	Click on book return icon	Member should be able to return book	Book returned successfully

Therefore, through above observed test cases, we can predict the expected and actual results of our requirements. High quality results are obtained by designing test cases from UML models.

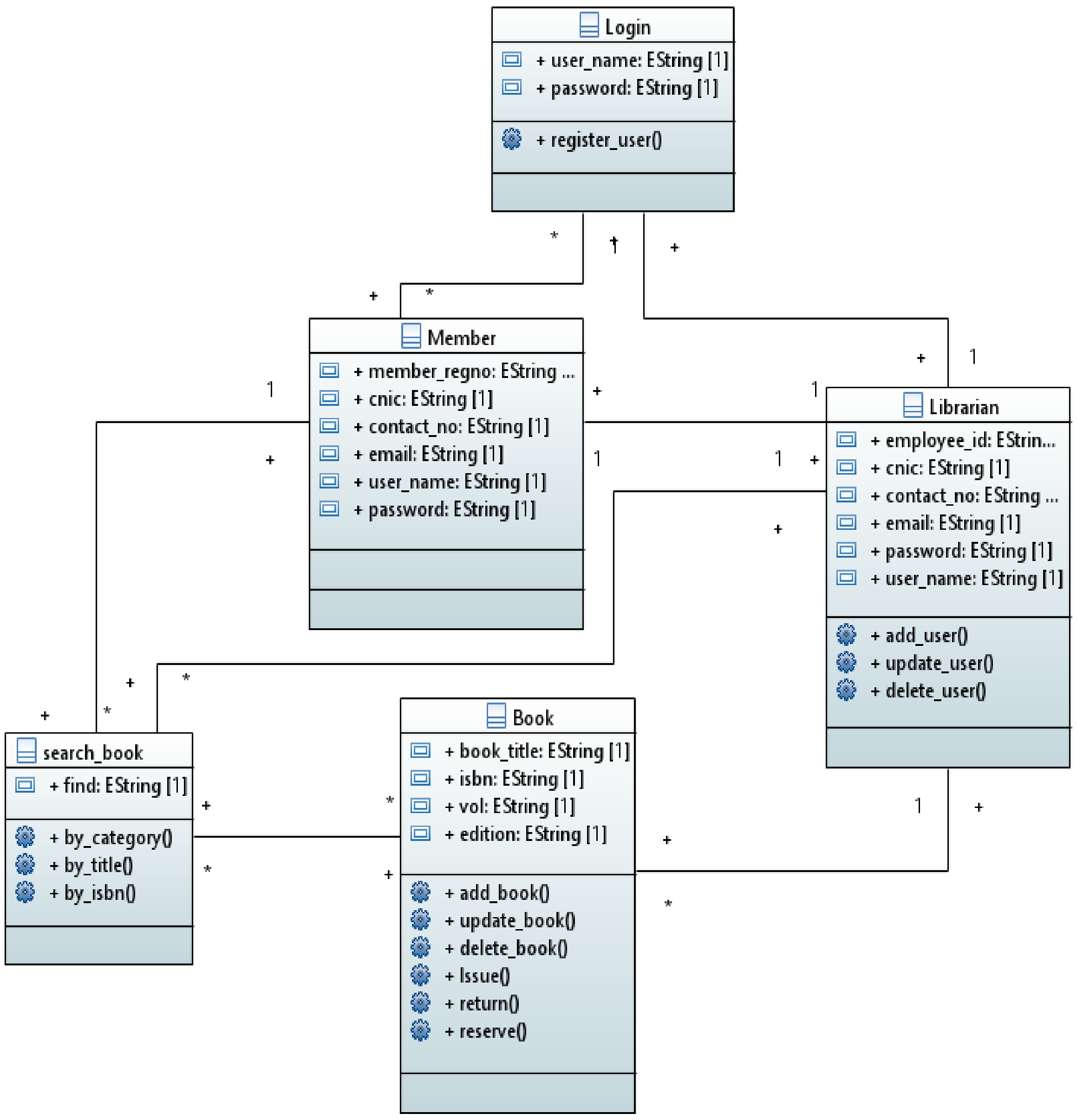


Figure 33. Class Diagram of LMS

Experimental Results of SUT2

In System under test 2 large number of operations are covered. The control flow graph CFG has been shown in figure 32. Genetic Algorithm is applied to generate optimized test cases with maximum statement coverage. Therefore, 90% statement coverage has been achieved in minimum time by using an optimization algorithm i.e., UNAGA3. On the other hand, 90% statement coverage is achieved manually by going through a series of steps that is a much time-consuming process. Generated test cases are shown below in table 15.

Table 15. Generated Test Cases of SUT2

Test cases	Coverage, %	Statement Coverage Indicator
1	24%	111001010000000000000
2	33%	111101000010000000000
3	33%	*111101000010000000000*
4	42%	111101001000000100000
5	42%	*111101001000000100000*
6	48%	111101001000001000000
7	52%	111101001000000010000
8	62%	111100100000100000000
9	67%	111100100000010000000
10	71%	111100100001000000000
11	71%	*110100000000000000000*
12	85%	110101000000000001011
13	90%	110110000000000001010
14	90%	110110000000000000000

When population size is given 10 with 100 iterations

```
UNSGA3: Best solution found:
X = [False  True  True  True  True  True  True  True False  True False  True
 False False  True  True  True  True  True  True]
 F = [0.03030303]
```

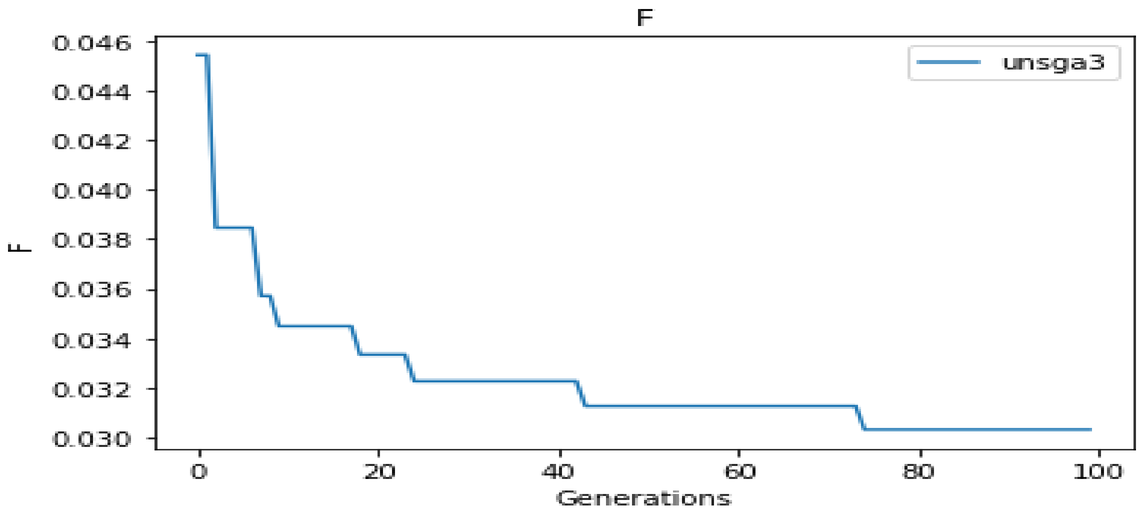


Figure 34. Achieved statement Coverage with population size 10

When population size is 20 and number of generations are 100

```
UNSGA3: Best solution found:
X = [False False  True  True  True  True  True  True  True  True  True  True
  True  True  True  True False False False  True]
F = [0.03030303]
```

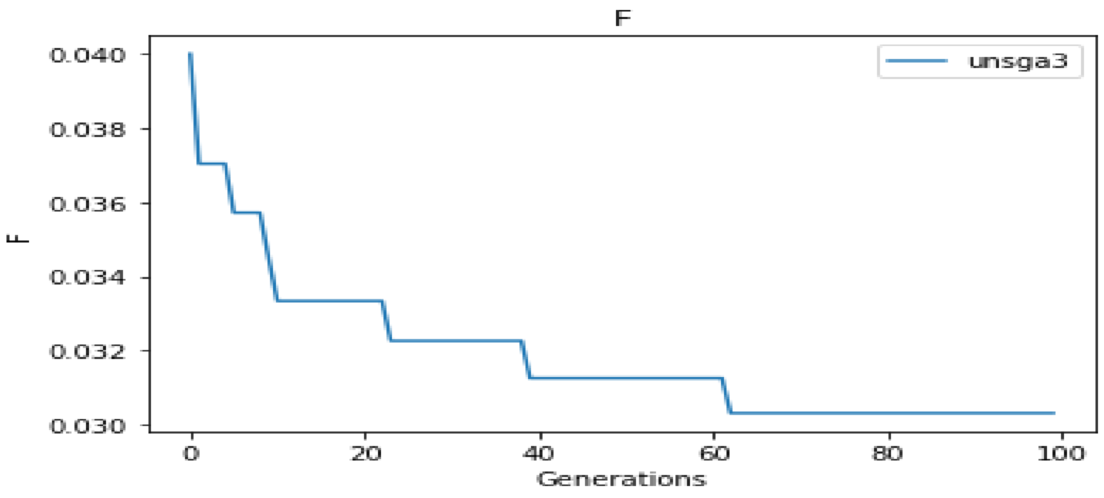


Figure 35. Achieved Statement Coverage with population size 20

When population size is 30 and number of generations 100

```
UNSGA3: Best solution found:
X = [False  True  True  True  True  True  True  True  True  True  True  True
  False  True  True False  True  True False  True]
F = [0.03030303]
```

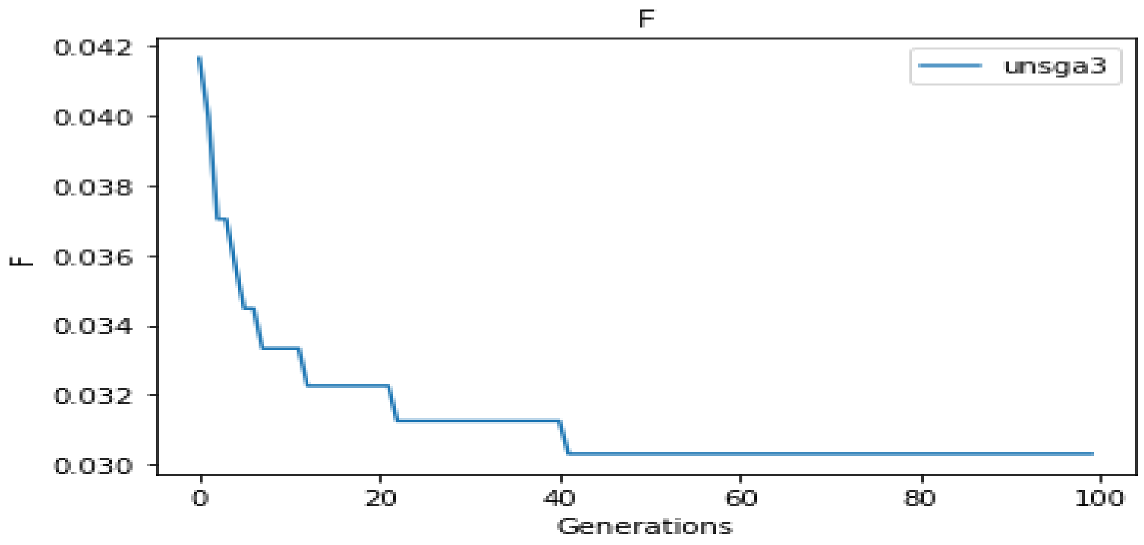


Figure 36. Achieved Statement Coverage with population size 30

When population size is 40 and number of generations are set 100

```
UNSGA3: Best solution found:
X = [False  True  True  True  True  True  True False  True  True False  True
  True  True  True  True  True  True  True  True  True]
F = [0.03225806]
```

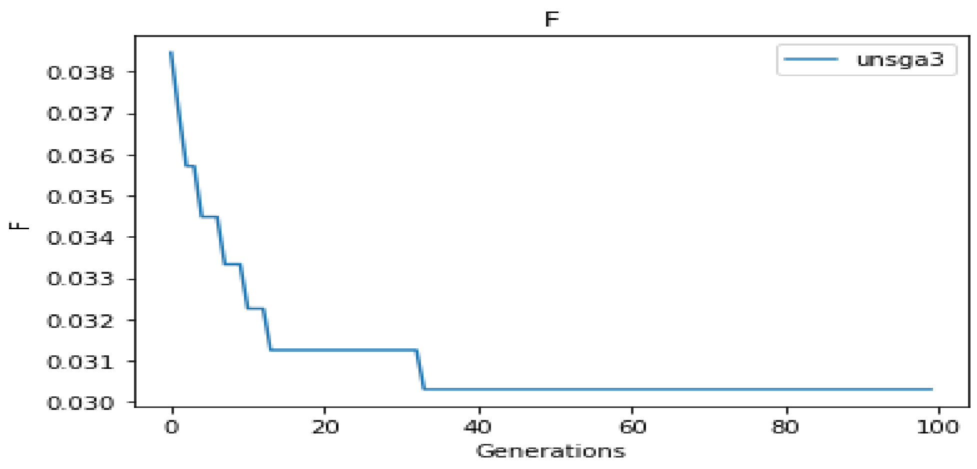


Figure 37. Achieved Statement Coverage with population size 40

When population size is 50 and number of iterations are 100

```
UNSGA3: Best solution found:
X = [False  True  True False  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True]
F = [0.03030303]
```

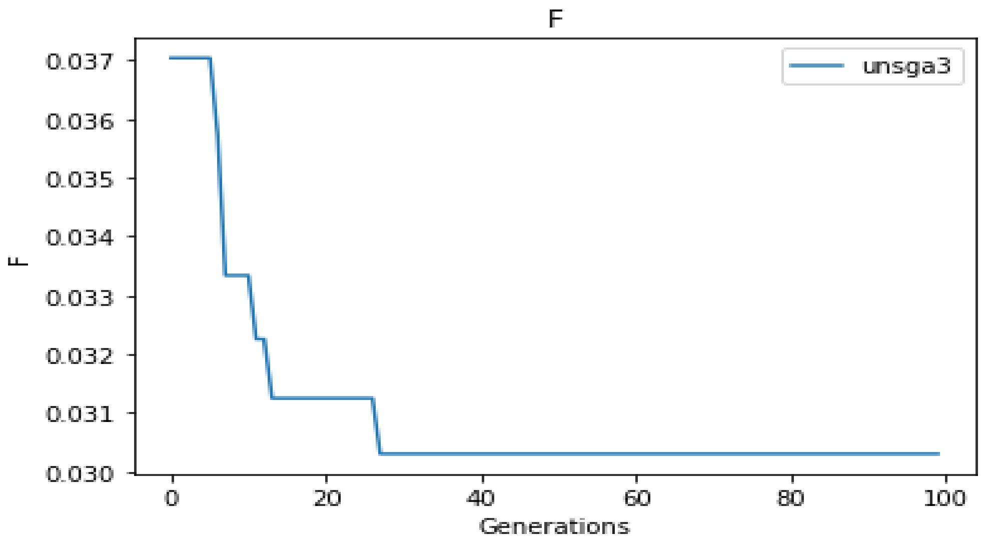


Figure 38. Achieved Statement Coverage with population size 50

Comparison with GA

Best Result by UNSGA3 with final population

```
UNSGA3: Best solution found:
X = [False  True  True False  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True]
F = [0.03030303]
```

Best Result Found by GA

```
Best solution found:
X = [False  True  True  True  True  True  True  True  True  True  True  True
  True  True False  True  True  True False  True]
F = [0.04347826]
```

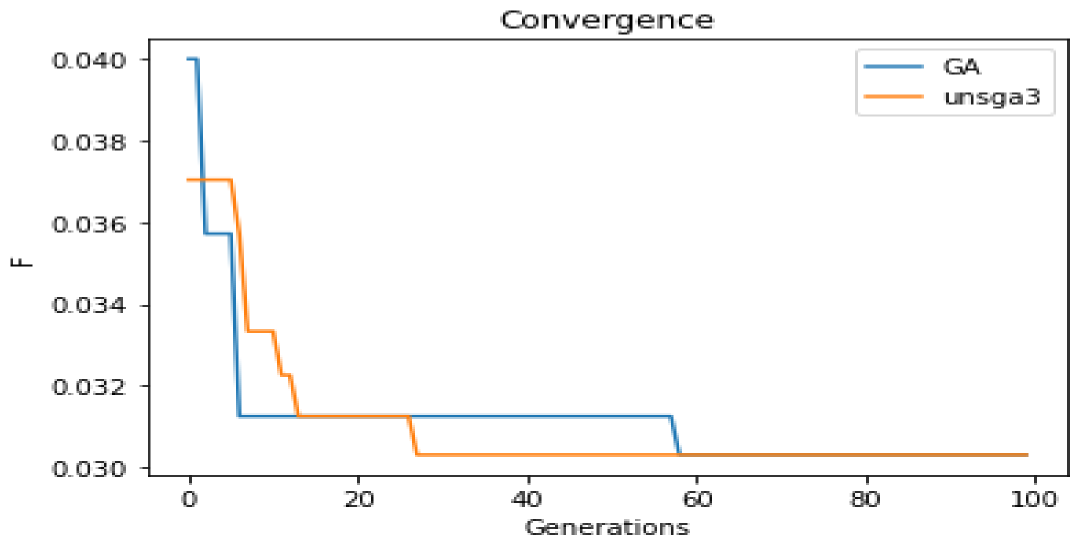


Figure 39. Comparison of UNSGA3 with GA

Comparison with Particle Swarm Optimization Algorithm (PSO)

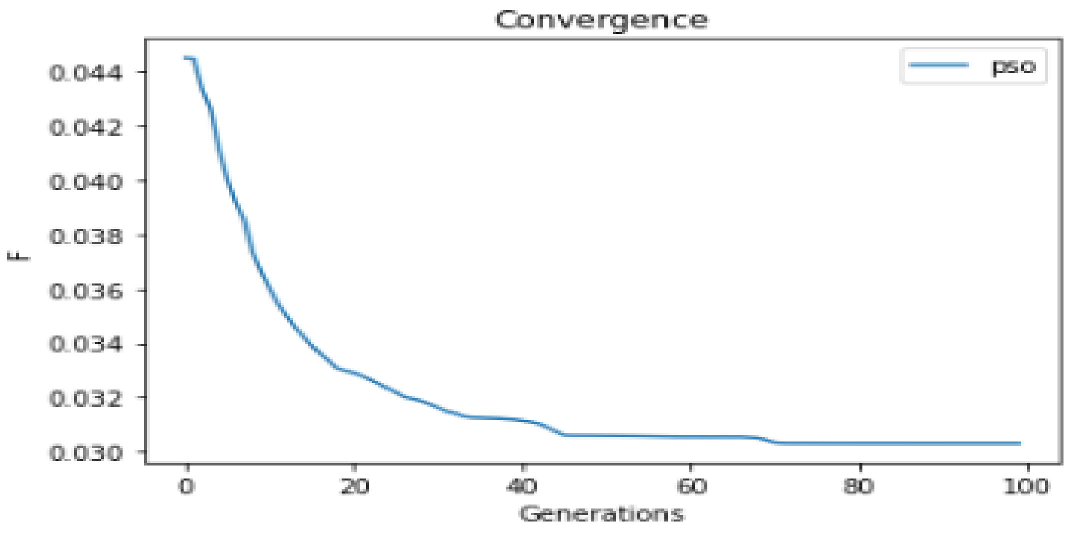


Figure 40. Statement Coverage with PSO

Table 16. Comparison with GA and PSO

Parameters			SUT 1			SUT 2		
Algorithms	Pop Size	No.of iters.	Convergence	Execution Time (sec)	Coverage %	Convergence	Execution Time (sec)	Coverage %
U-NSGA-III	50	100	**15**	**1.56**	95%	**25**	**1.65**	90.47%
GA	50	100	25	1.83	90%	59	1.81	85%
PSO	50	100	35	1.62	75%	70	2.455	65%

Chapter 5. Discussion And Conclusion

An automated test case generation technique has been presented in this research to generate test cases automatically from unified modelling language. The research is carried out by doing a systematic literature review (SLR). 73 research studies have been selected published between 2011-2021 based on review protocol. This study has identified 3 different test case automation approaches of which the model-based test case generation approach has been widely used. 14 techniques have been identified of which genetic algorithm have remained an efficient method for test case generation and optimization. Moreover, 11 validation protocols are found in the review, ‘Automated Teller Machine’ (ATM) is commonly utilized to validate the identified techniques. Although, this study has presented a comprehensive overview of model based automated test case generation and optimization techniques but, there is still a need to do a comparative analysis of identified techniques to identify their strengths and weaknesses.

For optimization of test cases from UML a unified non-sorting genetic algorithm has been introduced in this research. UNSGA3 has been implemented and a comparative analysis is done with UNSGA3-GA and UNSGA3-PSO in terms of statement coverage percentages. After performing experiments on both case studies, it has been observed that UNSGA3 performed better in terms of coverage percentage, convergence rate and execution time. However, performance of a proposed methodology is evaluated through two benchmark case studies. Average calculated statement coverage is 95% and 90% for both case studies. It has been observed that manually generated test cases are much time consuming as compared to automated test case generation and optimization method. Furthermore, a comparative analysis is carried out with other optimization algorithms and the proposed methodology has shown promising performance in every aspect.

The benefits of using UNSGA3 for model-based testing are as follows:

- This approach is much efficient in finding a best test case from an optimized test suit.
- This unified approach is used for automated test case generation
- The proposed approach can be used for single objective as well as multi-objective problems
- For white box testing, this methodology is capable enough to achieve 95% coverage
- It can be used for different domain models by simply replacing the inputs
- The unified approach works well with binary tournament selection
- It is capable enough of handling constraint problems

The proposed approach has been implemented and compared with other optimization algorithms. Results have been listed in chapter 4. By using this approach minimum test cases with maximum statement coverage have been achieved.

In this thesis, for automated test case generation and optimization multiple experiments have performed for each case study. It is **concluded** from the results that U-NSGA-III performed better for both case studies as compared to other selected approaches. The proposed approach not only reduces the number of test cases and minimizes the time of execution but also maximizes statement coverage. It shows that it is not only a safe approach but also behaves as a human expert for optimization of test cases by selecting the most suitable among them. Moreover, in this research single objective problem has been selected for model-based automated test case generation and optimization but this technique can also be used for multi-objective optimization problems. Therefore, by using U-NSGA-III approach test cases can be reduced effectively by maximizing coverage criteria. However, in future the formulation of new fitness function and integration of other evolutionary approaches with U-NSGA-III seems promising for model base test case generation and optimization.

References

[1] K. Kaur and V. Chopra, "Review of automatic test case generation from UML diagram using evolutionary algorithm," *International Journal of Inventive Engineering and Sciences (IJIES), ISSN*, pp. 2319-9598, 2014.

[2] P. N. Boghdady, N. L. Badr, M. Hashem, and M. F. Tolba, "A proposed test case generation technique based on activity diagrams," *International Journal of Engineering & Technology IJET-IJENS*, vol. 11, no. 03, pp. 1-21, 2011.

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

[4] R. G. Tiwari, A. P. Srivastava, G. Bhardwaj, and V. Kumar, "Exploiting UML Diagrams for Test Case Generation: A Review," in *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*, 2021: IEEE, pp. 457-460.

[5] S. S. Priya and P. Malarchelvi, "Test path generation using uml sequence diagram," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 3, no. 4, 2013.

[6] S. U. Ahmed, S. A. Sahare, and A. Ahmed, "Automatic test case generation using collaboration UML diagrams," *World Journal of Science and Technology,* vol. 2, 2013.

[7] A. Verma, A. Khatana, and S. Chaudhary, "A comparative study of black box testing and white box testing," *Int. J. Comput. Sci. Eng,* vol. 5, no. 12, pp. 301-304, 2017.

[8] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, "An automated model based testing approach for platform games," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2015: IEEE, pp. 426-435.

[9] E. Fourneret, J. Cantenot, F. Bouquet, B. Legeard, and J. Botella, "Setgam: Generalized technique for regression testing based on uml/ocl models," in *2014 Eighth International Conference on Software Security and Reliability (SERE)*, 2014: IEEE, pp. 147-156.

[10] C. Zhang, Z. Duan, B. Yu, C. Tian, and M. Ding, "A test case generation approach based on sequence diagram and automata models," *Chinese Journal of Electronics,* vol. 25, no. 2, pp. 234-240, 2016.

[11] S. Karakatič and T. Schweighofer, "A novel approach to generating test cases with

genetic programming," in *International Conference on Knowledge Management in Organizations*, 2015: Springer, pp. 260-271.

[12] A. K. Jena, S. K. Swain, and D. P. Mohapatra, "A novel approach for test case generation from UML activity diagram," in *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 2014: IEEE, pp. 621-629.

[13] V. Panthi and D. P. Mohapatra, "Automatic test case generation using sequence diagram," in *Proceedings of International Conference on Advances in Computing*, 2013: Springer, pp. 277-284.

[14] K. L. Petry, E. OliveiraJr, and A. F. Zorzo, "Model-based testing of software product lines: Mapping study and research roadmap," *Journal of Systems and Software,* vol. 167, p. 110608, 2020.

[15] D. Kumar and K. K. Mishra, "The impacts of test automation on software's cost, quality and time to market," *Procedia Computer Science,* vol. 79, pp. 8-15, 2016.

[16] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," Citeseer, 2007.

[17] S. Arifiani and S. Rochimah, "Generating test data using ant Colony Optimization (ACO) algorithm and UML state machine diagram in gray box testing approach," in *2016 International Seminar on Application for Technology of Information and Communication (ISemantic)*, 2016: IEEE, pp. 217-222.

[18] P. Mahali, S. Arabinda, A. A. Acharya, and D. P. Mohapatra, "Test case generation for concurrent systems using uml activity diagram," in *2016 IEEE Region 10 Conference (TENCON)*, 2016: IEEE, pp. 428-435.

[19] M. Khandai, A. A. Acharya, and D. P. Mohapatra, "A novel approach of test case generation for concurrent systems using UML Sequence Diagram," in *2011 3rd International Conference on Electronics Computer Technology*, 2011, vol. 1: IEEE, pp. 157-161.

[20] B. Hoseini and S. Jalili, "Automatic test path generation from sequence diagram using genetic algorithm," in *7'th International Symposium on Telecommunications (IST'2014)*, 2014: IEEE, pp. 106-111.

[21] C. T. M. Hue, D. D. Hanh, and N. N. Binh, "A transformation-based method for test case automatic generation from use cases," in *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, 2018: IEEE, pp. 252-257.

[22] X. Wang, X. Jiang, and H. Shi, "Prioritization of test scenarios using hybrid genetic

algorithm based on UML activity diagram," in *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2015: IEEE, pp. 854-857.

[23] M. Dhineshkumar, "An approach to generate test cases from sequence diagram," in *2014 International Conference on Intelligent Computing Applications*, 2014: IEEE, pp. 345-349.

[24] R. Anbunathan and A. Basu, "Dataflow test case generation from UML Class diagrams," in *2013 IEEE International Conference on Computational Intelligence and Computing Research*, 2013: IEEE, pp. 1-9.

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

[26] S. S. Panigrahi, S. Shaurya, P. Das, A. K. Swain, and A. K. Jena, "Test Scenarios Generation Using UML Sequence Diagram," in *2018 International Conference on Information Technology (ICIT)*, 2018: IEEE, pp. 50-56.

[27] V. Panthi, A. Gardizy, R. K. Mohapatra, and D. P. Mohapatra, "Functionality testing of object-oriented software using UML state machine diagram," in *2018 International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET)*, 2018: IEEE, pp. 1-7.

[28] P. E. Patel and N. N. Patil, "Testcases formation using UML activity diagram," in *2013 International Conference on Communication Systems and Network Technologies*, 2013: IEEE, pp. 884-889.

[29] Z. A. Hamza and M. Hammad, "Generating test sequences from UML use-case diagrams," in *2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, 2019: IEEE, pp. 1-6.

[30] C.-S. Wu, C.-H. Huang, and Y.-T. Lee, "The test path generation from state-based polymorphic interaction graph for object-oriented software," in *2013 10th International Conference on Information Technology: New Generations*, 2013: IEEE, pp. 323-330.

[31] Y. Li and L. Jiang, "The research on test case generation technology of UML sequence diagram," in *2014 9th International Conference on Computer Science & Education*, 2014: IEEE, pp. 1067-1069.

[32] M. Elallaoui, K. Nafil, and R. Touahni, "Automatic generation of TestNG tests cases from UML sequence diagrams in Scrum process," in *2016 4th IEEE International*

Colloquium on Information Science and Technology (CiSt), 2016: IEEE, pp. 65-70.

[33] P. Boghdady, N. Badr, M. Hashem, and M. Tolba, "An enhanced technique for generating hybrid coverage test cases using activity diagrams," in *2012 8th International Conference on Informatics and Systems (INFOS)*, 2012: IEEE, pp. SE-20-SE-28.

[34] S. Dalal and S. Hooda, "A novel technique for testing an aspect oriented software system using genetic and fuzzy clustering algorithm," in *2017 International Conference on Computer and Applications (ICCA)*, 2017: IEEE, pp. 90-96.

[35] Y. Amannejad, V. Garousi, R. Irving, and Z. Sahaf, "A search-based approach for cost-effective software test automation decision support and an industrial case study," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, 2014: IEEE, pp. 302-311.

[36] A. K. Sultania, "Developing software product and test automation software using Agile methodology," in *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*, 2015: IEEE, pp. 1-4.

[37] N. Ye, X. Chen, P. Jiang, W. Ding, and X. Li, "Automatic regression test selection based on activity diagrams," in *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement-Companion*, 2011: IEEE, pp. 166-171.

[38] R. Ahmadi, K. Jahed, and J. Dingel, "mCUTE: A model-level concolic unit testing engine for UML state machines," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019: IEEE, pp. 1182-1185.

[39] A. Khatun and K. Sakib, "An automatic test suite regeneration technique ensuring state model coverage using UML diagrams and source syntax," in *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*, 2016: IEEE, pp. 88-93.

[40] R. Ramler and C. Klammer, "Enhancing acceptance test-driven development with model-based test generation," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2019: IEEE, pp. 503-504.

[41] M. Boucher and G. Mussbacher, "Transforming workflow models into automated end-to-end acceptance test cases," in *2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering (MiSE)*, 2017: IEEE, pp. 68-74.

[42] F. T. Rodríguez, F. Lonetti, A. Bertolino, M. P. Usaola, and B. P. Lamancha, "Extending UML testing profile towards non-functional test modeling," in *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2014: IEEE, pp. 488-497.

[43] S.-Y. Wang, J.-Z. Sun, and J. Zhang, "The Method of Generating Web Link Security Testing Scenario Based on UML Diagram," in *2016 IEEE Trustcom/BigDataSE/ISPA*, 2016: IEEE, pp. 1831-1838.

[44] R. Moraes, H. Waeselynck, and J. Guiochet, "UML-based modeling of robustness testing," in *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, 2014: IEEE, pp. 168-175.

[45] A. Nayak and D. Samanta, "Synthesis of test scenarios using UML activity diagrams," *Software & Systems Modeling*, vol. 10, no. 1, pp. 63-89, 2011.

[46] A. K. Jena, S. K. Swain, and D. P. Mohapatra, "Test case creation from UML sequence diagram: a soft computing approach," in *Intelligent Computing, Communication and Devices*: Springer, 2015, pp. 117-126.

[47] S. Dhir and D. Kumar, "Automation software testing on web-based application," in *Software Engineering*: Springer, 2019, pp. 691-698.

[48] A. Tripathy and A. Mitra, "Test case generation using activity diagram and sequence diagram," in *Proceedings of International Conference on Advances in Computing*, 2013: Springer, pp. 121-129.

[49] S. K. Swain, D. P. Mohapatra, and R. Mall, "Test case generation based on use case and sequence diagram," *International Journal of Software Engineering*, vol. 3, no. 2, pp. 21-52, 2010.

[50] F. A. D. Teixeira and G. B. e Silva, "EasyTest: An approach for automatic test cases generation from UML Activity Diagrams," in *Information Technology-New Generations*: Springer, 2018, pp. 411-417.

[51] D. Barisas, E. Bareiša, and Š. Packevičius, "Automated method for software integration testing based on UML behavioral models," in *International Conference on Information and Software Technologies*, 2013: Springer, pp. 272-284.

[52] A. Kaur and V. Vig, "Automatic test case generation through collaboration diagram: a case study," *International Journal of System Assurance Engineering and Management*, vol. 9, no. 2, pp. 362-376, 2018.

[53] P.-L. Vincent, L. Badri, and M. Badri, "Regression testing of object-oriented software: a technique based on use cases and associated tool," in *Computer*

Applications for Software Engineering, Disaster Recovery, and Business Continuity: Springer, 2012, pp. 96-106.

[54] M. Shirole and R. Kumar, "Constrained permutation-based test scenario generation from concurrent activity diagrams," *Innovations in Systems and Software Engineering*, pp. 1-11, 2021.

[55] P. K. Arora and R. Bhatia, "A systematic review of agent-based test case generation for regression testing," *Arabian Journal for Science and Engineering*, vol. 43, no. 2, pp. 447-470, 2018.

[56] A. Hettab, E. Kerkouche, and A. Chaoui, "A graph transformation approach for automatic test cases generation from UML activity diagrams," in *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering*, 2015, pp. 88-97.

[57] M. Shirole, A. Suthar, and R. Kumar, "Generation of improved test cases from UML state diagram using genetic algorithm," in *Proceedings of the 4th India Software Engineering Conference*, 2011, pp. 125-134.

[58] S. Yimman, S. Kamonsantiroj, and L. Pipanmaekaporn, "Concurrent test case generation from UML activity diagram based on dynamic programming," in *Proceedings of the 6th International Conference on Software and Computer Applications*, 2017, pp. 33-38.

[59] R. Elghondakly, S. Moussa, and N. Badr, "A comprehensive study for software testing and test cases generation paradigms," in *Proceedings of the International Conference on Internet of things and Cloud Computing*, 2016, pp. 1-7.

[60] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 2015, pp. 385-396.

[61] S. Kamonsantiroj, L. Pipanmaekaporn, and S. Lorpunmanee, "A memorization approach for test case generation in concurrent uml activity diagram," in *Proceedings of the 2019 2nd International Conference on Geoinformatics and Data Analysis*, 2019, pp. 20-25.

[62] F. Häser, M. Felderer, and R. Breu, "Software paradigms, assessment types and non-functional requirements in model-based integration testing: a systematic literature review," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1-10.

[63] M. Bernardino, E. M. Rodrigues, and A. F. Zorzo, "Performance Testing Modeling:

an empirical evaluation of DSL and UML-based approaches," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 1660-1665.

[64] M. Shirole and R. Kumar, "UML behavioral model based test case generation: a survey," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 4, pp. 1-13, 2013.

[65] P. Gulia and R. Chillar, "A new approach to generate and optimize test cases for UML state diagram using genetic algorithm: http://doi. acm. org/10.1145/180921.2180933," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 3, pp. 1-5, 2012.

[66] M. Shirole and R. Kumar, "Testing for concurrency in UML diagrams," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 5, pp. 1-8, 2012.

[67] T. E. Vos, I. Prasetya, S. Eldh, S. Getir, A. Parsai, and P. Aho, "Automating TEST Case Design, Selection and Evaluation Report on 10 Editions of A-TESTWorkshop," *ACM SIGSOFT Software Engineering Notes*, vol. 45, no. 1, pp. 21-24, 2020.

[68] N. Khurana and R. Chillar, "Test case generation and optimization using UML models and genetic algorithm," *Procedia Computer Science*, vol. 57, pp. 996-1004, 2015.

[69] P. K. Arora and R. Bhatia, "Agent-based regression test case generation using class diagram, use cases and activity diagram," *Procedia Computer Science*, vol. 125, pp. 747-753, 2018.

[70] I. Septian, R. S. Alianto, and F. L. Gaol, "Automated test case generation from UML activity diagram and sequence diagram using depth first search algorithm," *Procedia computer science*, vol. 116, pp. 629-637, 2017.

[71] 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, pp. 1351-1359, 2019.

[72] T. Ahmad, J. Iqbal, A. Ashraf, D. Truscan, and I. Porres, "Model-based testing using UML activity diagrams: A systematic mapping study," *Computer Science Review*, vol. 33, pp. 98-112, 2019.

[73] S. Pradhan, M. Ray, and S. K. Swain, "Transition coverage based test case generation from state chart diagram," *Journal of King Saud University-Computer and Information Sciences*, 2019.

[74] V. Garousi, A. B. Keleş, Y. Balaman, Z. Ö. Güler, and A. Arcuri, "Model-based testing in practice: An experience report from the web applications domain," *arXiv preprint arXiv:2104.02152*, 2021.

[75] P. Sapna and A. Balakrishnan, "An approach for generating minimal test cases for regression testing," *Procedia computer science,* vol. 47, pp. 188-196, 2015.

[76] K. Lano, K. Jin, and S. Tyagi, "Model-based Testing and Monitoring using AgileUML," *Procedia Computer Science,* vol. 184, pp. 773-778, 2021.

[77] Y. Shin, Y. Choi, and W. J. Lee, "Integration testing through reusing representative unit test cases for high-confidence medical software," *Computers in biology and medicine,* vol. 43, no. 5, pp. 434-443, 2013.

[78] L. Burgueño, A. Vallecillo, and M. Gogolla, "Teaching UML and OCL models and their validation to software engineering students: an experience report," *Computer Science Education,* vol. 28, no. 1, pp. 23-41, 2018.

[79] M. Abdelgawad, S. McLeod, A. Andrews, and J. Xiao, "Model-based testing of a real-time adaptive motion planning system," *Advanced Robotics,* vol. 31, no. 22, pp. 1159-1176, 2017.

[80] N. M. Minhas, S. Masood, K. Petersen, and A. Nadeem, "A systematic mapping of test case generation techniques using UML interaction diagrams," *Journal of Software: Evolution and Process,* vol. 32, no. 6, p. e2235, 2020.

[81] J. P. Faria, A. C. Paiva, and Z. Yang, "Test generation from UML sequence diagrams," in *2012 Eighth International Conference on the Quality of Information and Communications Technology*, 2012: IEEE, pp. 245-250.

[82] L. Haldurai, T. Madhubala, and R. Rajalakshmi, "A study on genetic algorithm and its applications," *International Journal of Computer Sciences and Engineering,* vol. 4, no. 10, p. 139, 2016.

[83] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proceedings of the world congress on engineering,* 2011, vol. 2, no. 1: International Association of Engineers Hong Kong, China, pp. 1-6.

[84] P. Kora and P. Yadlapalli, "Crossover operators in genetic algorithms: A review," *International Journal of Computer Applications,* vol. 162, no. 10, 2017.

[85] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, "Crossover and mutation operators of genetic algorithms," *International journal of machine learning and computing,* vol. 7, no. 1, pp. 9-12, 2017.

[86] V. Prakash, S. Tatale, V. Kondhalkar, and L. Bewoor, "A critical review on au-tomated test case generation for conducting combinatorial testing using particle swarm opti-mization," *Int. J. Eng. Technol,* vol. 7, no. 3, p. 22, 2018.