Copyright ⓒ 2024 KSII

A UML-based Approach towards Test Case Generation and Optimization

Shahid Saleem[1], Saif U. R. Malik[2], Bilal Mehboob[3], Roobaea Alroobaea[4], Sultan Algarni[5], Abdullah M. Baqasah[6], Naveed Ahmad[7] and Muhammad Hasnain[8,*]

[1,8] Faculty of Computer Science, Lahore Leads University Lahore, Punjab Pakistan
[e-mail: shahidnoon16@hotmail.com, drhasnain.it@leads.edu.pk]
[2] Cybernetica AS Tallinn Estonia [e-mail: saif.rehmanmalik@gmail.com]
[3] Department of Software Engineering, Superior University Lahore, 54000, Pakistan
[email: Bilal.mehboob@superior.edu.pk]
[4] Department of Computer Science, College of Computers and Information Technology, Taif University, P. O. Box 11099, Taif 21944, Saudi Arabia [e-mail: r.robai@tu.edu.sa]
[5] Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University (KAU), Jeddah 21589, Saudi Arabia [email : saalgarni@kau.edu.sa]
[6] Department of Information Technology, College of Computers and Information Technology, Taif University, Taif 21974, Saudi Arabia [email : a.baqasah@tu.edu.sa]
[7] Department of Software Engineering, National University of Computer and Emerging Sciences, Islamabad, P.O. Box 42230, Pakistan [email : naveed.ahmad@nu.edu.pk]
[*]Corresponding author: Muhammad Hasnain

Received November 23, 2023; revised January 27, 2024; accepted February 26, 2024; published March 31, 2024

Abstract

Software testing is an important phase as it ensures the software quality. The software testing process comprises of three steps: generation, execution, and evaluation of test cases. Literature claims the usage of single and multiple 'Unified Modeling Language' (UML) diagrams to generate test cases. Using multiple UML diagrams increases test case coverage. However, the existing approaches show limitations in test case generation from UML diagrams. Therefore, in this research study, we propose an approach to generate the test cases using UML State Chart Diagram (SCD), Activity Diagram (AD), and Sequence Diagram (SD). The proposed approach transforms UML diagrams into intermediate forms: SCD Graph, AD Graph, and SD Graph respectively. Furthermore, by integrating these three graphs, a System Testing Graph (STG) is formed. Finally, test cases are identified from STG by using a traversal algorithm such as Depth First Search (DFS) that is an optimization method. The results show that the proposed approach is better compared to existing approaches in terms of coverage and performance. Moreover, the generated test cases have the ability to detect faults at the unit level, integration, and system level testing.

Keywords: Activity Diagram, State Chart Diagram, Sequence Diagram, System Testing Graph, Model Based Testing, Test Case Generation.

1. Introduction

Software testing is an essential phase in Software Development Life Cycle (SDLC) [1]. It is the process of executing a program on a set of well-designed tests with the intent of discovering error [2]. It aims to identify flaws, bugs, defects and errors in the program. To assess the real results and compare them with the required and expected results, it needs to run a program with a number of test suites. A software product quality is based on how in-depth it is tested. Software testing is also used to test the product for other quality factors like security, efficiency, usability, maintainability, integrity, portability, compatibility and reliability [3]. It starts from the initial phase and continues throughout the SDLC to reduce cost, time and defects. The main objectives of software testing include; identifying bugs, suggesting changes and modification, checking behavior according to the specification and to ensure quality [4].

It is reported in the literature that a huge budget (50%) of SDLC is spent on testing [5] [6]. A drastic increase in the usage of a computer application enhances the size of a software system and makes it more complex, that's why testing becomes more error-prone step in software development process. Many applications are safety critical, so reliability is important. In other words, a quality application is required to meet high reliability. The most common and important methodology from existing techniques is followed to increase reliability. Therefore, software testing is an important part of software quality assurance (SQA) in SDLC.

The testing process is primarily divided into three phases i.e. generation, execution and evaluation [6]. Generation of test cases is found more problematic and error-prone step in the testing process [7]. Test case execution and evaluation are relatively easy to perform; therefore, most of the researchers focus on test cases generation (TCG). In literature, different methods have been used to generate test cases i.e. Scenario Based (SB), Code Based (CB) and Model Based (MB) testing [8]. In SB approaches, the source of test case generation is requirement specification, in CB it is the code and in MB approach uses the system models. Among these methods, MB is effective and efficient than CB because it includes both source code and specification [7]. Models are built by using requirement specification and are helpful to generate source code. The model captures the important information from the specification which is the base for implementation. MB has the ability to identify such type of errors which are not easily identified by CB. There are many advantages of MB i.e. it is easy to understand, reduces the testing time, focuses on specification coverage, and enables the early identification of faults and independence from implementation.

Therefore, an MB approach to generate test cases has the advantage of applying testing processes throughout the SDLC and widely used for TCG because it requires less testing time and effort [9]. In MB approach, various system models are used to generate test cases such as Unified Modeling Language (UML), Data Flow Diagram (DFD), Graph Transformation System (GTS), Entity Relationship Diagram (ERD) and Abstract State Machine (ASM) [7].

The Object Management Group (OMG) standard is widely used in software system industry [10]. It became an industrial de-facto standard for visualization [7]. UML language is used to design, construct, artifact and modify system specifications [11]. It is a standard language for modeling software blueprints. It designs both static and dynamic view of the system and shows different aspects [5]. Different types of models are widely used in industry. All aspects of a system are not covered by a single model, so UML defines numerous type of diagrams for possible aspects of the system [3]. With the wide acceptance of UML in the software industry, researchers started investigating how UML-based model is useful for testing? To find the answer, a number of software testing techniques have been proposed based on UML models.

Of late, TCG from UML models such as SCD, AD and SD has received attention from researchers. As discussed in [12], a collaboration diagram represents the dynamic behavior of objects unlike other diagrams. It is found that generation of test cases using individual diagrams has certain problems. As reported in [13] that generated test cases based on SCD only deal with the testing of a single object of the class. In the case of a complex system where multiple objects interact for the required system's behavior, the existing technique is inadequate. Test cases generated from AD reported in [5] [10] [14] do not give any state information. Therefore, the state of the object remains unknown during execution. A system behaves differently to same input according to its state. So information of state and object interaction is necessary for effective TCG. TCG from SD only address the interaction faults [15] [16]. To overcome these issues, researchers have proposed approaches based on combinational UML models. Authors in [6] claim that by merging multiple UML model coverage of test cases increased and helped to detect more faults.
Contribution of this study is as follows:

In this study, we proposed an approach to generate test cases using combinational UML model i.e. SCD, AD and SD. The first step is to transform SCD, AD and SD into an intermediate form State Chart Diagram Graph (SCDG), Activity Diagram Graph (ADG) and Sequence Diagram Graph (SDG), respectively. The second step is to merge the SCDG, ADG, and SDG to create a combined graph called System Testing Graph (STG). In the third step, a traversal algorithm Depth First Search (DFS) method is used to construct test paths by traversing STG. Finally, the optimized test paths are generated. A tool is developed to demonstrate the proposed methodology. The proposed approach is evaluated on two key cases: "ATM card validation" and "library book issue".

The rest of paper is organized as follows:
Section 2 briefly describes the basic concepts of testing, Section 3 presents an overview of the related work on software testing and test case generation. Section 4 presents our proposed approach followed by experimental results and finally in section 5. Section 6 concludes and summarizes the main points of the study.

2. Basic Concepts

In this section, we describe different concepts and terminology that we used in rest of paper.

2.1 Activity Diagram

The AD is a UML diagram that explains a process in a system. It is mostly useful for demonstrating the flow of activities within a system, software process and business process. The AD is a visual representation to depict the dynamic behavior of a specific task. It is often used in the initial phases of software development to visualize and understand the steps and actions involved in a particular process. It achieves this by modeling the flow of control from one activity to another activity [17] [18]. In accumulation to defining the sequence of activities, an AD also captures the representation of data interactions within the system, providing a comprehensive visualization of both the process flow and the inherent data modeling.

2.2 State Chart Diagram

A SCD is a type of UML diagram that is aimed to visualize the dynamic behavior of a system in response to external stimuli. A SCD shows different states of an object or a system and the transitions between these states. They are mainly useful for capturing the life cycle of an object or the behavior of a system over time.

SCDs are beneficial for modeling the behavior of complex systems that can be in different states and transition between them based on events and conditions. They provide a visual representation of the system's behavior over time.
A SCD is a graphical tool that demonstrates the lifecycle of an object, portraying its various states and transitions to expressive the vibrant behavior of a software system. This type of diagram is particularly beneficial for modeling objects that respond to external stimuli or specific events [18].

2.3 Sequence Diagram

Sequence diagram (SD) captures the interaction between objects and order of interaction [3]. It is most common interaction diagram, which mainly focuses on the interaction between two objects through exchanging the messages in lifelines.
The SD provides a sequential representation of interactions, a valuable instrument for understanding the flow of interaction among objects in a system. They are normally used during the design phase to capture and communicate the dynamic aspects of a system.

2.4 Coverage Criteria

Numerous approaches exist in literature to generate test cases based on UML models and these approaches have their own pros and cons. For the evaluation of these approaches, some evaluation parameters including intermediate representation, coverage criteria, and automation were identified in [19]. Coverage criteria are most commonly used parameter for evaluation. Various path coverage criterion are used for evaluation of the testing approaches such as message path coverage, activity coverage, path coverage, branch coverage and transition coverage. In this work, we used path coverage. A path shows a sequence of nodes from initial to leaf node while counting each node at least once. To calculate path coverage in percentage, we used the following formula [15],

$$\frac{\text{nodes covered in path}}{\text{total no. of nodes in Model}} \times 100\%$$

3. Related Work

Techniques to generate test cases can be categorized as sequence based testing (SBT), model-based testing (MBT) and coverage base testing (CBT) [9]. Mainly test suites are structured from the source code [2]. The implementation phase started after the completion of the design phase of SDLC. So it's challenging to generate test cases during an earlier phase of the SDLC. It is appropriate to generate test cases at early stages without time-consuming. The MB approach is more systematic and effective than CB [2]. A review of literature outlines that MB techniques are widely used because these techniques require less testing time and effort [7]. In MB, test cases are generated from different system models like UML, DFD, GTS and ERD etc. In literature, UML-based models are comprehensively used [7]. Many of existing studies are in favor of using TCG from UML diagrams. Numerous studies have been proposed approaches and methodologies for the TCG from UML models [3]-[6], [8], [20]-[21]. It has been found that studies use individual models and combination of models for TCG.
In literature, it is also found that generation of test cases using individual diagrams has certain problems. As reported in [12] that test cases based on SCD only deal with the testing of a single object of the class. In the case of a complex system where multiple objects interact for required system behavior, and existing technique is inadequate. Test cases generated from AD

as reported in [5] [10] [22] do not give any state information. Therefore, the state of the object remains unknown during the execution of test cases. A system behaves differently to same input according to its state. So information of state and object interaction is necessary for the effective TCG. Test cases generated from SD only address the interaction faults [15] [16]. To overcome these issues researchers have proposed approaches based on combinational UML models. Authors of a study [6] claim that by merging multiple UML model coverage of test cases increased and helped to detect more faults.

Studies [3] [8] [21] [23] [24] [25] used combination of UML models such as AD and SD to generate test cases. They transform UML models into an intermediate form known as a graph. Then integrate both graphs to generate a combined graph. After the generation of combine graph, they traverse it by using different traversing algorithms such as DFS and Breadth First Search (BFS). Different UML models have distinct abilities to detect the various type of faults. A research study [6] presented a technique to generate TC using combinational UML models i.e. SCD and SD. They convert models into intermediate form graph then integrate both graphs into single graph known as system testing graph. After that resultant graph traversed to identify test sequence known as test cases.

A recent approach [26] is focused on combinational testing to generate TC. Authors in this study used SD for generating combinatorial test cases. Once the information has been created, the optimization algorithms are used to generate TC. As state-based testing has been known as a challenging task in software testing, authors in [12] proposed an approach using various coverage criteria derived from SCD. It has been observed from previous studies [27] that TC are generated by minimizing time and cost. However, the challenge is to use such techniques proposed from UML diagrams and consider the concurrent states for generating TC.

Artificial intelligence (AI) models, such as machine learning techniques (MLT), have been widely used in the literature to perform complex tasks for improving software performance. A recent study [28] reports the use of MLT for TC reduction and TC prioritization. These techniques can be trained on data to learn the patterns and predict or decide about new data. However, the present study is being undertaken to propose a TCG strategy based on multiple diagrams, and recent advancements in AI cannot comprehend this task until sufficient data is available to train the models.

4. Proposed Approach

In this study, we have proposed an approach to generate test cases using combinational UML model i.e. SCD, AD and SD. There are four main steps in the proposed methodology. The first step is to transform SCD, AD and SD into an intermediate form: State Chart Diagram Graph (SCDG), Activity Diagram Graph (ADG) and Sequence Diagram Graph (SDG), respectively. The second step is to merge the SCDG, ADG, and SDG to create a combined graph called System Testing Graph (STG). In the third step, a traversal algorithm DFS method is used to generate test paths by traversing STG. Finally, the optimized test paths are generated. In the following section, we explain the proposed approach.

4.1 Transformation into Intermediate Form

The first part of the proposed methodology is the transformation of UML model into intermediate representation i.e. SCD, AD, and SD into SCDG, ADG, and SDG, respectively.

4.1.1 Transformation of SCD into SCDG

In this part, we have clearly defined SCD and SCDG. After that, present a technique to transform SCD into SCDG. An SCD designs the dynamic behavior of a class in response to internal or external stimuli. Specifically, it describes the behavior of a single object in response to multiple events.

4.1.2 Definition of SCDG

The SCDG was used by [6], [23] and [29] is defined in Eq. (1)

$$SCDG = \left\{ \sum N_{SCDG} , \quad \sum E_{SCDG} , \sum C_{SCDG} , \; N_i , \; N_f \right\} \tag{1}$$

$\sum N_{SCDG}$ is the set of nodes in SCD in which an object behaves same in response to stimuli, $\sum E_{SCDG}$ is the set of all transitions shows that change from one node to another, whereas, $\sum C_{SCDG}$ is the set of conditions that enables the belonging transition to lead a different transition. The initial node, N_i, shows the initial state of all objects while N_f is the final node that shows end of the object existence.

Next, we examine the transformation technique of SCD into SCDG. Each state in SCD is mapped as a node and an edge from node N_i to N_j is used to show sequential dependency of N_i on N_j . **Fig. 1** shows the SCD for ATM card validation transaction and its corresponding SCDG is represented in **Fig. 2**.

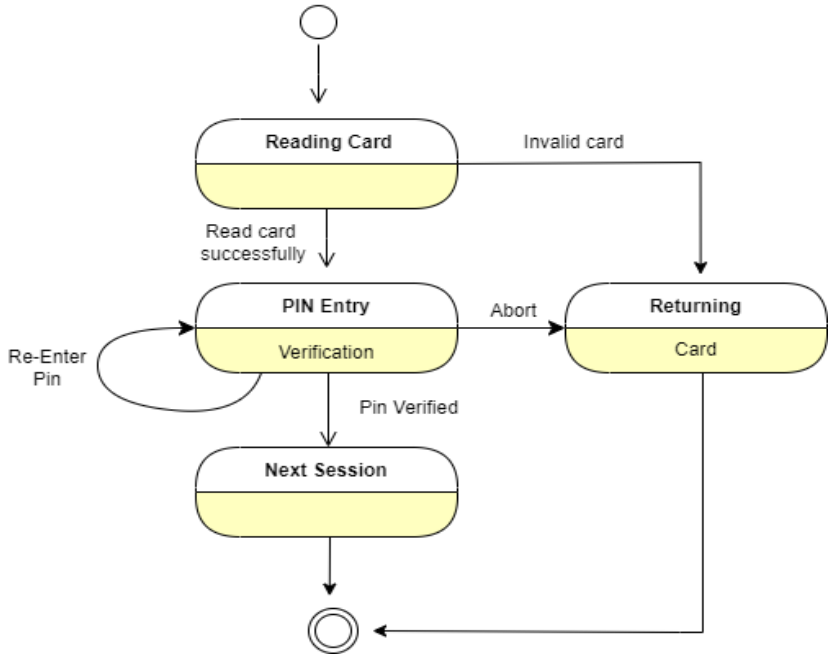

Fig. 1. State chart diagram for ATM

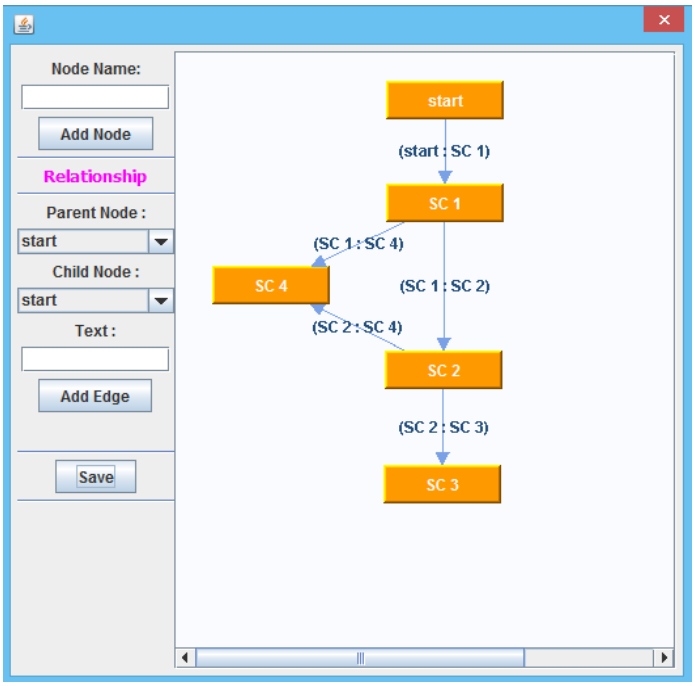

Fig. 2. State chart diagram graph

4.1.3 Transformation of AD into ADG

After defining AD and ADG at the start of this part, we have discussed the approach to covert AD into ADG. An AD is used to design flow-of-control from one activity to another activity. An activity can be defined as the operation of the system.

4.1.4 Definition of ADG

The ADG was used by [6] and [21] is defined in Eq. (2)

$$ADG = \left\{ \sum N_{ADG} , \quad \sum E_{ADG} , \sum C_{ADG} , \; N_i , \; N_f \right\} \tag{2}$$

$\sum N_{ADG}$ is a set of all activity states, $\sum E_{ADG}$ is a set of all transitions between two states and $\sum C_{ADG}$ is a set of all conditions, whereas, Ci is corresponding transition E_i that leads to the next transition. $N_i \in \sum N_{ADG}$ represents the initial activity state while $N_f \in \sum N_{ADG}$ represents the final activity state.

Here, we explain the conversion of AD into ADG. Each activity in AD is mapped as a node and an edge between two nodes shows the sequential dependency between them. **Fig. 3** show the AD for card validation scenario of ATM and ADG as represented in **Fig. 4**.

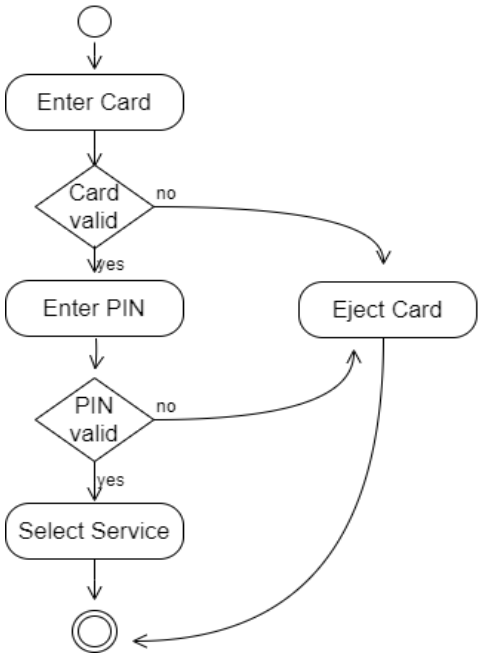

Fig. 3. Activity diagram for ATM

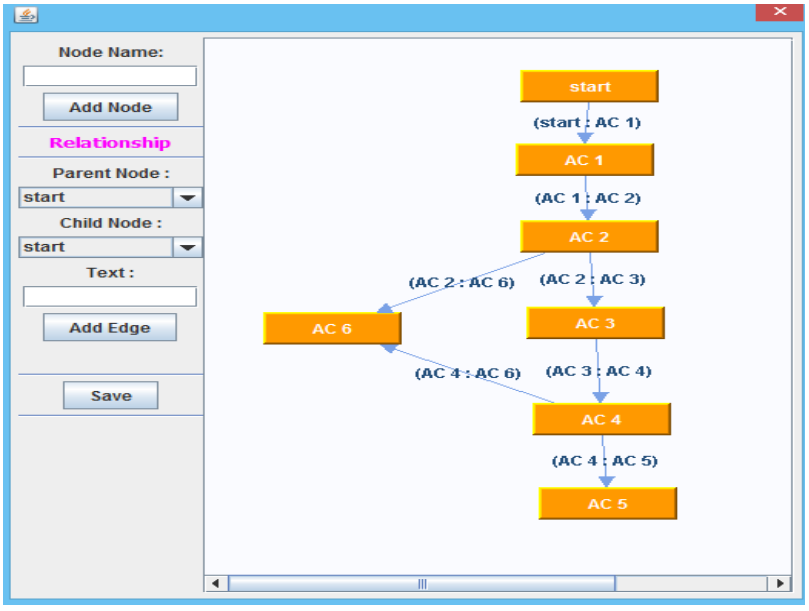

Fig. 4. Activity diagram graph

4.1.5 Transformation of SD into SDG

After defining SD and SDG at the start of this part, we have discussed the approach to transform SD into SDG. A sequence diagram is used to design interaction between two objects and order of interaction.

4.1.6 Definition of SDG

The SDG was used by [6] and [23] is defined in Eq. (3)

$$SDG = \left\{ \sum N_{SDG} , \quad \sum E_{SDG} , \; N_i , \; N_f \right\} \tag{3}$$

Where, $\sum N_{SDG}$ is the set of nodes showing numerous states in a scenario while $\sum E_{SDG}$ is the set of all edges showing transition between various states. N_i is the initial node expressing the first state, whereas, N_f is final node which represent final state of an object.
In order to devise a method, we specify a scenario as follows:

$$Scn = < Scn\ ID\ ; StartState\ ; MessageSet\ ; Successful/Unsuccessful >$$

Where $Scn\ ID$ (scenario identity) is an identity number used for each scenario, $StartState$ is the beginning point of each scenario or starting state of a scenario. Similarly, $MessageSet$ represent all events that take place in a scenario while $Successful/Unsuccessful$ is a state that conveys final output of a system. The success of the system totally depends upon a user selection.
An event in a scenario can be defined as:

$$Event = < msg\ Id\ ; from\ Object\ ; to\ Object\ ; \ [/Constraint] \ >$$

Where $msg\ Id$ is a unique identity number which is assign to every message to represent its identity, $from\ Object$ belongs to an object that has sent a message, $to\ Object$ represents the object that receives a message. $Constraint$ represents the condition/s having concern to an event occurrence. An event that has an iterative process specify through a steric (*).

Fig. 5 present the SD of ATM card validation, **Fig. 6** shows the various scenarios in SD and corresponding SDG as presented in **Fig. 7**.

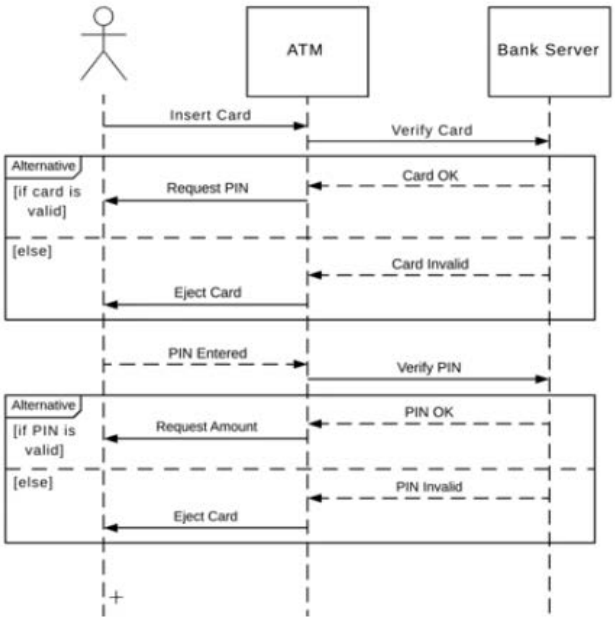

Fig. 5. Sequence diagram for ATM

<Scn1
State X
S1(m1, a, b)
S2(m2, b, c)
S3(m3, b, a)
Unsuccessful>

<Scn2
State X
S1(m1, a, b)
S2(m2, b, c)
S4(m4, a, b)
S5(m5, b, c)
S6(m3, b, a)
Unsuccessful>

<Scn3
State X
S1(m1, a, b)
S2(m2, b, c)
S4(m4, a, b)
S5(m5, b, c)
S7(m6, b, a)
Successful>

Fig. 6. Scenario Triplet

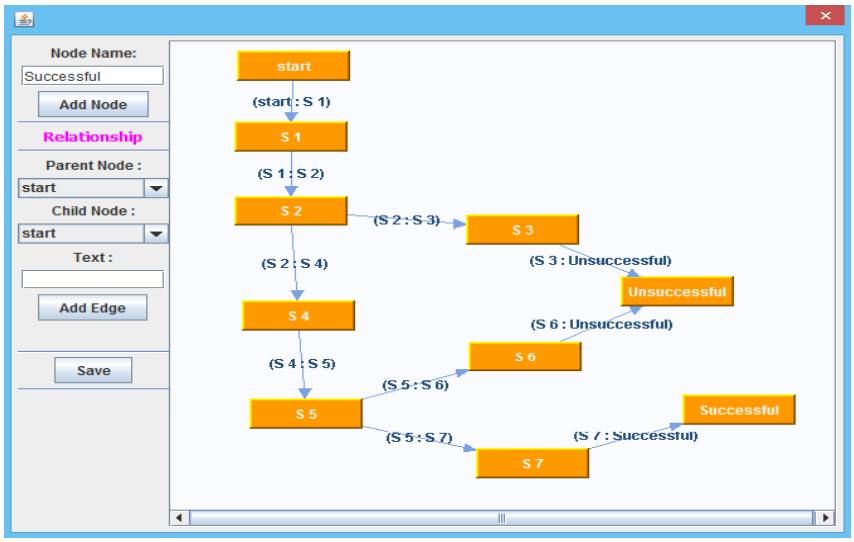

Fig. 7. Sequence diagram graph

4.2 Integrating SCDG, SDG, and SDG into STG

Subsequent to generating SCDG, ADG, and SDG, the three graphs are merged as presented in **Fig. 2, Fig. 4** and **Fig. 7** into a system testing graph, which is a combined graph.

4.2.1 Definition of STG

The STG is defined in Eq. (4)

$$STG = \left\{ \sum N_{STG} , \quad \sum E_{STG} , \; N_i , \; N_f \right\} \tag{4}$$

Where, $\sum N_{STG} = \left\{ \sum N_{SCDG} \right\} \cup \left\{ \sum N_{ADG} \right\} \cup \left\{ \sum N_{SDG} \right\}$ is a set of all states of SCD, AD and SD while, $\sum E_{STG}$ is a set of all transition between different states of SCD, AD and SD. N_i is Starting node of SCDG whereas, the final set of nodes of STG is N_f .

For integration of three graphs as per definition of STG, we proposed an algorithm. **Fig. 8** shows the STG after combining three graphs SCDG, ADG, and SDG. Detailed description of merging graphs is explained in Algorithm 1.

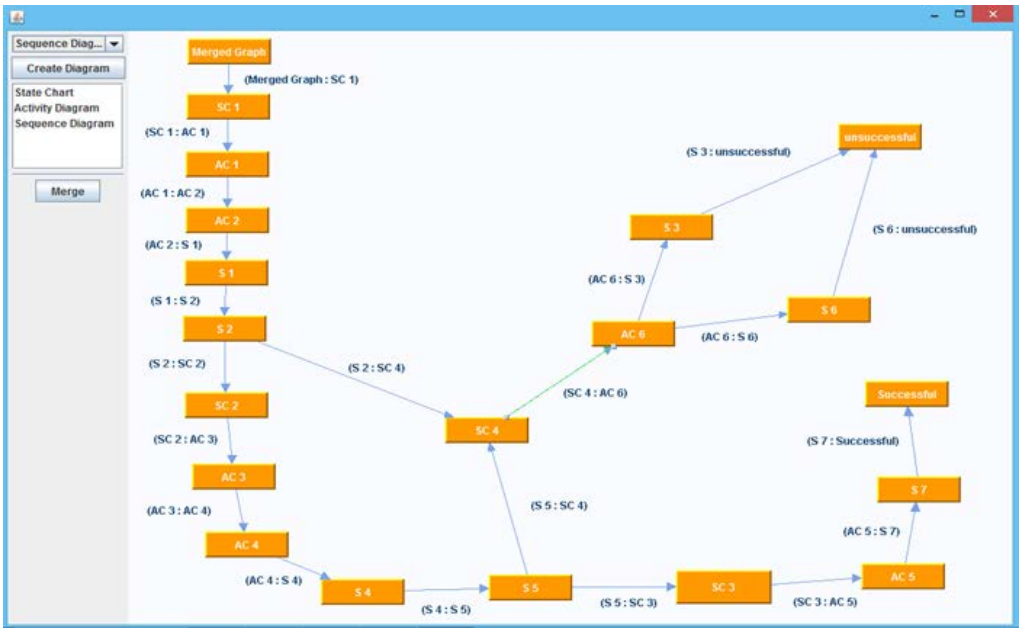

Fig. 8. System testing graph

4.2.2 Algorithm

An algorithm is proposed from merging three graphs. It starts traversing from SCDG graph. While traversing a graph whenever a conditional node occurs with the multiple paths, it draws an edge from that node to next graph a flow such as; SCDG to ADG, ADG to SDG and SDG to SCDG. It repeated till all vertices of three graphs are visited.

4.2.3 Algorithm 1: Generate STG

Input: SCDG, ADG, and SDG.
Output: STG.

1. $P_1 = N_{SCDG}$ // current node of SCDG.
2. $P_2 = N_{ADG}$ // current node of ADG.
3. $P_3 = N_{SDG}$ // current node of SDG.
4. $M_G = N_m$ // current node of merged graph.
5. $merge\ (P_1, P_2, P_3, M_G)$
    a. **While P_1 ! null**
        i. If P_1 ! child M_G
            1. Add node in child of merged graph.
            2. $M_G = P_1$
        ii. End if
        iii. If P_1 ! conditional node
            1. P_1 = child of P_1
            2. Repeat step i
        iv. Else
            1. Break the loop
        v. End if
    b. End while.
    c. **While P_2 ! null**

i. If P_2 ! child M_G
   1. Add node to the child of a merged graph.$M_G = P_2$
ii. End if
iii. If P_2 ! conditional node
   1. P_2 = child of P_2
   2. Repeat step i
iv. Else
   1. Break the loop
v. End if

d. End while.
e. While P_3 ! null
i. If P_3 ! child M_G
   1. Add node to the child of a merged graph.
   2. $M_G = P_3$
ii. End if
iii. If P_3 ! conditional node
   1. P_3 = child of P_3
   2. Repeat step i
iv. Else
   1. Break the loop
v. End if

f. End while.

6. End merge
7. If P_1, P_2 and P_3 have conditional node
   a. Repeat step 5 for each path.

4.3 Test Case Generation

In the third part of the proposed methodology, all paths of STG from start node to leaf node are enumerated. Our approach traverses STG using traversal algorithm depth-first search (DFS) method that guarantees all possible vertices are visited. All possible test paths STG are depicted in **Fig. 9**.

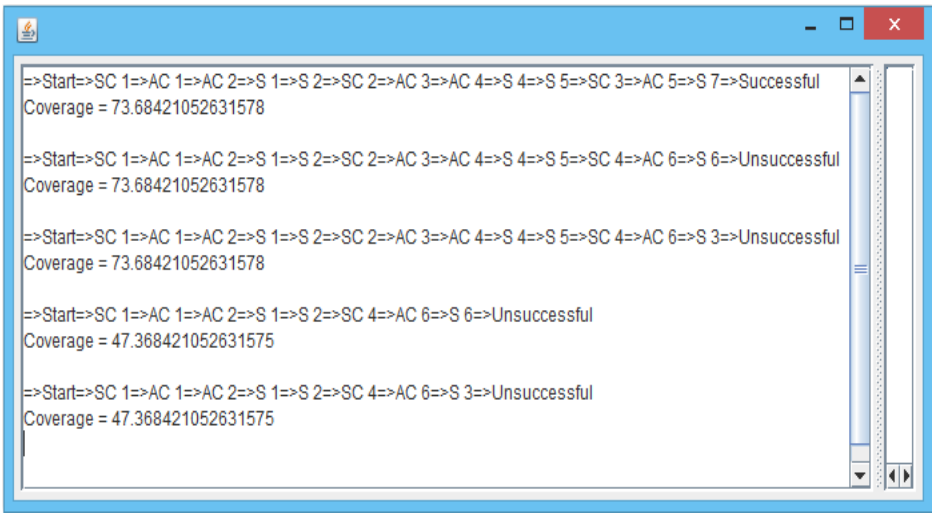

Fig. 9. Lists of possible test paths

Each path of STG considers as a test case. TCG process is discussed in Algorithm 2.

4.3.1 Algorithm 2: Test Case Generation

Input: STG
Output: Test suit (T)

1. Detect all possible paths $P = \{ \{P_1\}, \{P_2\}, \{P_3\}, \dots\dots, \{P_n\} \}$ //from starting node to leaf node in the STG
2. For every path $P_i \in P$ do
    a. $N_j = N_x$ // current node; initialize with start node.
    b. $t_i \leftarrow \emptyset$ // for path P_i initially test case is empty
    c. Do
        i. If N_j = State node
            1. Select T
            2. t = $\{preC\ ,\ postC\}$
        ii. End if
        iii. If N_j = Activity node
            1. t = {testing step}
        iv. End if
        v. If N_j = Sequence node
            1. t = $\{I\ (a_1, a_2, a_3, \dots, a_n)\ ,\ O(d_1, d_2, d_3, \dots, d_m)\ \}$
            where,
            $I\ (a_1, a_2, a_3, \dots, a_n)$ = set of all inputs for method m (…)
            $O\ (d_1, d_2, d_3, \dots, d_m)$ = set of all resultant values when m (…) is executed.
        vi. End if
        vii. $t_i = t_i \cup t$ // add t into test set.
        viii. $N_j = N_k$ // the next node of path P_i.
        ix. $T = T \cup t_i$
    d. While $\left(N_j \neq N_z\right)$ // N_z is final node of path P_i.
    e. End while
    f. Determine the final output, input, post C and pre-C of scenario.
    g. $t = \{preC_i\ ,\ I_i\ ,\ O_i\ ,\ postC_i\}$
    h. $T \leftarrow T \cup t$
3. End for
4. Return (T)
5. Stop

TCG algorithm starts from determining all possible paths in the STG. Steps 5 to 14 identify numerous pre-conditions, inputs, output and post condition for every path. Step 2 to 15 is repeated for all paths. And finally, step 16 gives us test cases.

4.4 Test Case Optimization

Test case optimization is important to reduce time and cost. It reduces the size of test case without effecting quality factor. Those test cases are considered in the testing process, which are appropriate to the specified coverage. An optimized concept is a selection of best test case for test case execution. In optimization, test case reduction can be performed in two ways either at the time of TCG or after the TCG [4]. In this research, test case reduction is done at

the time of TCG. We reduce all redundant test suites at the time of integrating graphs into system testing graph. We also used a heuristic algorithm graph traversal DFS technique for optimization. It traverses graph in depth and identifies maximum coverage path.

5. Experimental Result and Discussion

In this section, the results of our approach are compared with the existing proposed approach to show that how effectively optimal test cases are generated with the maximum coverage. We have to implement the methods proposed in [6], [23] on two examples i.e. ATM card validation and library book issue to demonstrate results. In literature, most researchers demonstrated their approach through the example of ATM card validation. It is a well-known example to all; therefore, we also use it for the evaluation of our approach.

5.1 ATM Card Validation

The results of calculated path coverage of ATM card validation example are shown in **Fig. 10** and **Fig. 11**.

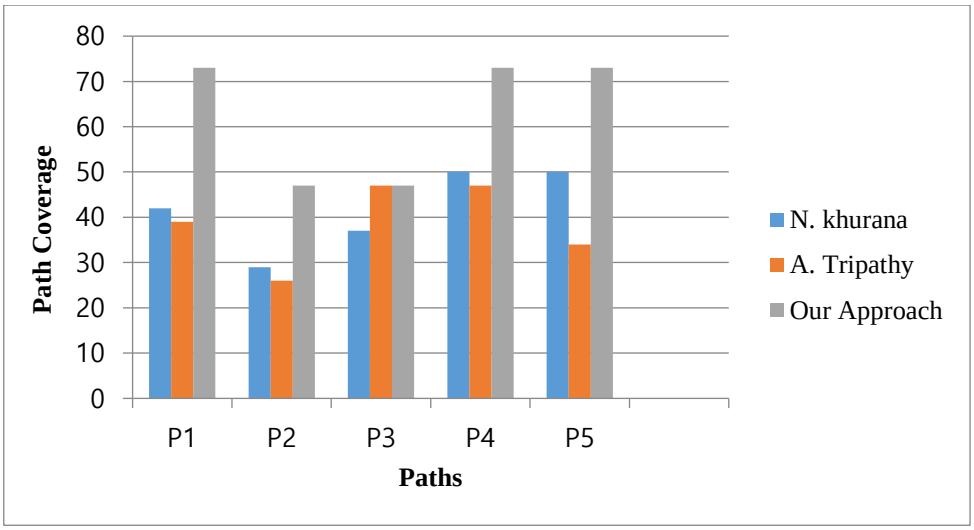

Fig. 10. Comparative analysis of ATM card validation

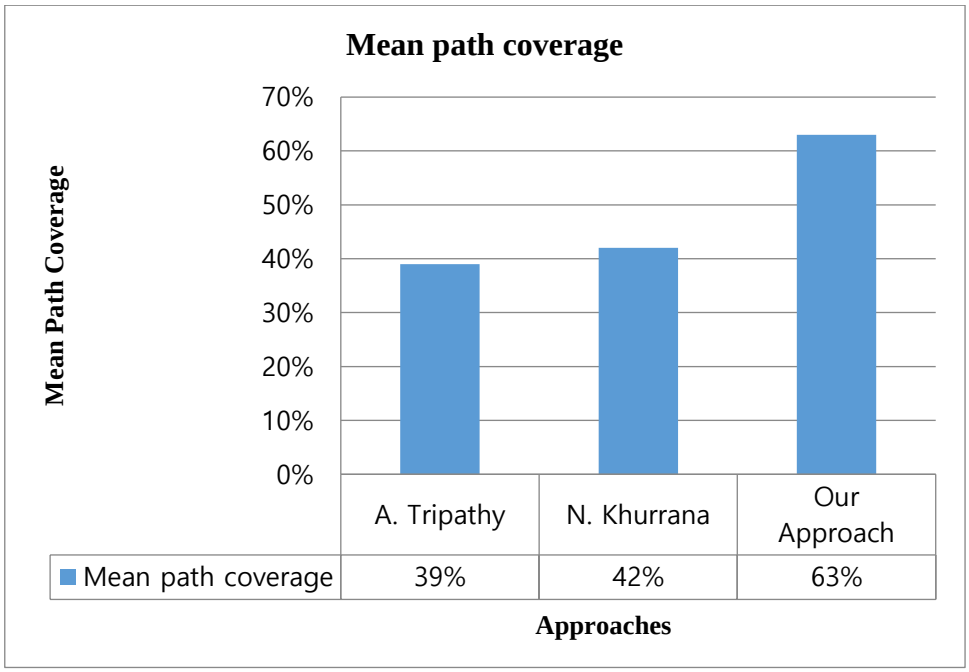

Fig. 11. Mean path coverage of ATM card validation

5.2 Library Book Issue

Calculated path coverage of Library book issue example is shown in **Fig. 12** and **Fig. 13**.

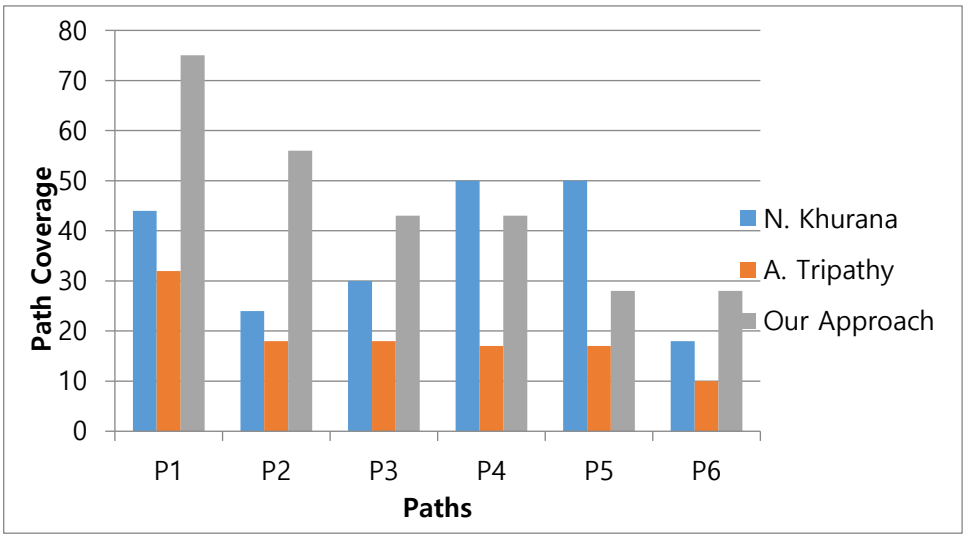

Fig. 12. Comparative analysis of library book issue

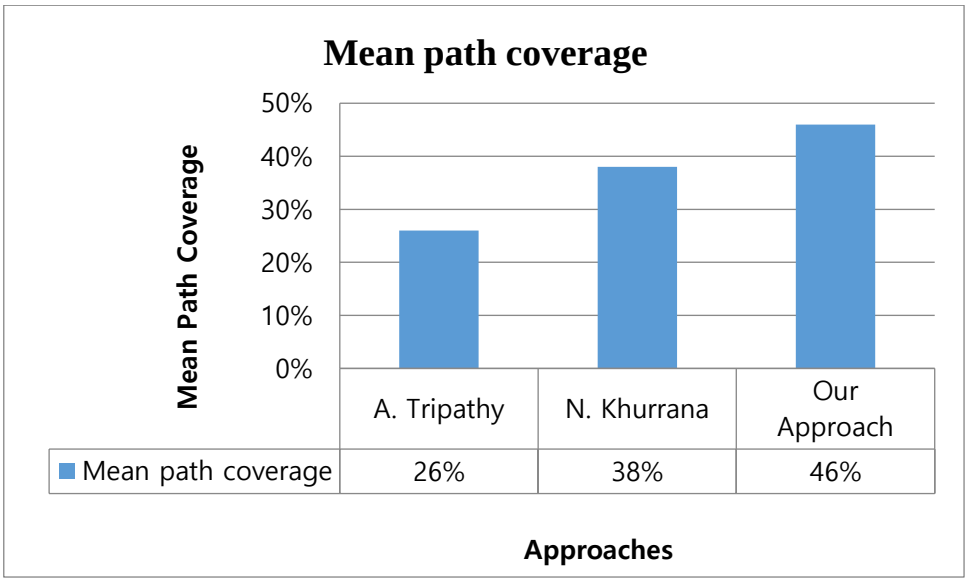

Fig. 13. Mean path coverage of library book issue

The results depicted in **Fig. 10**, **Fig. 11**, **Fig. 12** and **Fig. 13** show that our approach is superior in path coverage as compared to existing approaches. Through the comparative analysis containing three approaches (two existing and one proposed in this research study) were implanted by using above-mentioned examples to justify the proposed approach.

Meaningfully, the outcomes expose a significant improvement in path coverage with the combination of an additional UML diagram in the perspective of ATM card validation. The path coverage increased from 50% to 73%, while the mean coverage increased from 42% to 63% within the same case study. This considerable improvement underscores the efficiency of our approach in enhancing the carefulness of testing scenarios.

For the second case of Library Book issue, our approach sustained to surpasses the existing methods. The path coverage experienced a significant boost from 50% to an inspiring 75%. Furthermore, the mean coverage in the library book issue case increased from 38% to 46%. These results strengthen the superiority of our approach, representing its capacity to consistently attain more broad coverage in various scenarios.

In brief, the comparative results explicitly support the claim that our approach is preferred when compared with the existing methodologies. The combination of UML diagrams seems to be a key contributing aspect, considerably enhancing both path and mean coverage in various testing scenarios. These outcomes emphasize the potential impact of our approach on improving the efficiency of path coverage in software testing.

6. Conclusion and Future Work

In this paper, we have proposed an approach to generate optimal test cases with the maximum coverage. The proposed approach model-based enables us to generate test case. Three UML diagrams i.e. SCD, AD and SD were used to generate test cases from the proposed model. The proposed approach transformed UML diagrams into intermediate form by generating three graphs. A system testing graph was generated by integrating these three graphs. This paper used the depth first method to generate optimized test cases. By integrating three UML models, it covers the maximum test cases. Moreover, test cases generated from the approach resolve a

problem like integration, scenario, pre-post and operational. To evaluate the proposed approach we used two key cases such as “ATM card validation” and “library book issue”. Results depict that the proposed approach has maximum coverage and optimal results. In future works, we would formalize and verify the proposed approach on other cases to examine its generalization in real world cases of software systems.

Acknowledgment

This research work was funded by Institutional Fund Projects under grant no. (**IFPIP:378-611-1443**). The authors gratefully acknowledge technical and financial support provided by the Ministry of Education and King Abdulaziz University, DSR, Jeddah, Saudi Arabia

References

[1] P. N. Boghdady, N. L. Badr, M. A. Hashim, and M. F. Tolba, “An enhanced test case generation technique based on activity diagrams,” in *Proc. of ICCES’2011 2011 Int. Conf. Comput. Eng. Syst.*, no. June, pp. 289–294, 2011. Article (CrossRef Link).

[2] O. Oluwagbemi and H. Asmuni, “An approach for automatic generation of test cases from UML diagrams,” *Int. J. Softw. Eng*. its Appl., vol. 9, no. 8, pp. 87–106, 2015. Article (CrossRef Link).

[3] V. M. Sumalatha, G.S.V.P.Raju, “UML based Automated Test Case Generation technique using Activity-Sequence diagram,” *The International Journal of Computer Science & Applications (TIJCSA)*, vol. 1, no. 9, pp. 58–71, 2012. Article (CrossRef Link)

[4] B. N. Biswal, “Test Case Generation and Optimization of Object-Oriented Software using UML Behavioral Models,” 2010.

[5] A. Hettab, E. Kerkouche, and A. Chaoui, “A Graph Transformation Approach for Automatic Test Cases Generation from UML Activity Diagrams,” in *Proc. of the Eighth International C* Conference on Computer Science & Software Engineering*, pp. 88–97, 2015. Article (CrossRef Link)

[6] N. Khurana and R. S. Chillar, “Test Case Generation and Optimization using UML Models and Genetic Algorithm,” *Procedia Comput. Sci.*, vol. 57, pp. 996–1004, 2015. Article (CrossRef Link)

[7] D. Arora and B. Hazela, “Testing And Verification of Software Model Through Formal Semantics : A Systematic Review,” *Int. J. Res. Eng. Technol*., vol. 03, no. 10, pp. 78–82, 2014.

[8] M. Khandai, A. A. Acharya, and D. P. Mohapatra, “Test case generation for concurrent system using UML combinational diagram,” *Int. J. Comput. Sci. Inf. Technol. IJCSIT*, vol. 2, no. 5, pp. 97–102, 2011.

[9] N. Khurana and R. S. Chillar, “Literature Review of Test Case Generation Techniques for Object Oriented System,” *Int. J. Comput. Appl*., vol. 105, no. 15, 2014.

[10] A. K. Jena, S. K. Swain, and D. P. Mohapatra, “A novel approach for test case generation from UML activity diagram,” in *Proc. of Issues and Challenges in Intelligent Computing Techniques (ICICT) International Conference on*, pp. 621–629, 2014. Article (CrossRef Link)

[11] A. Hettab, A. Chaoui, and A. Aldahoud, “Automatic test cases generation from uml activity diagrams using graph transformation,” in *Proc. of The 6th International Conference on Information Technology*, vol. 8, pp. 1–12, 2013.

[12] Kaur and V. Vig, "Automatic test case generation through collaboration diagram: a case study," *International Journal of System Assurance Engineering and Management*, vol. 9, pp. 362-376, 2018. Article (CrossRef Link)

[13] S. Kansomkeat and W. Rivepiboon, “Automated-generating test case using UML statechart diagrams,” in *Proc. of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pp. 296–300, 2003.

[14] S. Kamonsantiroj, L. Pipanmaekaporn, and S. Lorpunmanee, "A memorization approach for test case generation in concurrent UML activity diagram," in *Proc. of the 2019 2nd International Conference on Geoinformatics and Data Analysis*, pp. 20-25, 2019. Article (CrossRef Link)
[15] N. K. Bahrin and R. Mohamad, "TCG algorithm approach for UML sequence diagram," in *Proc. of 2015 9th Malaysian Software Engineering Conference (MySEC)*, pp. 43–48, 2015. Article (CrossRef Link)
[16] Y. Seo, E. Y. Cheon, J.-A. Kim, and H. S. Kim, "Techniques to generate UTP-based test cases from sequence diagrams using M2M (Model-to-Model) transformation," in *Proc. of 2016 IEEE/ACIS 15th Int. Conf. Comput. Inf. Sci.*, pp. 1–6, 2016. Article (CrossRef Link).
[17] A. A. Acharya, P. Mahali, and D. P. Mohapatra, "Automated Test Case Generation Using Uml Use Case Diagram And Activity Diagram," *J. Theor. Appl. Inf. Technol.*, vol. 70, no. 3, pp. 399-412, 2014. Article (CrossRef Link)
[18] A. Abdurazik, J. Offutt, and A. Baldini, "A controlled experimental evaluation of test cases generated from UML diagrams," 2004.
[19] M. Touseef, N. Anwer, A. Hussain, and A. Nadeem, "Testing from UML Design using Activity Diagram: A Comparison of Techniques," *Int. J. Comput. Appl.*, vol. 131, no. 5, pp. 41–47, 2015. Article (CrossRef Link)
[20] M. Khandai, A. A. Acharya, and D. P. Mohapatra, "Test case generation for concurrent system using UML combinational diagram," *Int. J. Comput. Sci. Inf. Technol*. vol. 2, 2011.
[21] S. K. Swain and D. P. Mohapatra, "Test case generation from Behavioral UML Models," *Int. J. Comput. Appl.*, vol. 6, no. 8, pp. 5–11, 2010. Article (CrossRef Link)
[22] D. Kundu and D. Samanta, "A Novel Approach to Generate Test Cases from UML Activity Diagrams," *J. Object Technol.*, vol. 8, no. 3, pp. 65–83, 2009. Article (CrossRef Link)
[23] A. Tripathy and A. Mitra, "Test case generation using activity diagram and sequence diagram," in *Proc. of Int. Conf. Adv. Comput. ICAdC 2012*, pp. 121–129, 2013. Article (CrossRef Link).
[24] S. Dalai, A. A. Acharya, and D. P. Mohapatra, "Test Case Generation For Concurrent Object-Oriented Systems Using Combinational Uml Models," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 5, 2011. Article (CrossRef Link)
[25] R. Gupta and V. Jaglan, "Test case generation for UML behavioral diagram by traversal algorithm," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 10, pp. 3262–3266, Aug. 2019. Article (CrossRef Link).
[26] S. Tatale and V. C. Prakash, "Automatic Generation and Optimization of Combinatorial Test Cases from UML Activity Diagram Using Particle Swarm Optimization," *Ingénierie des Systèmes d'Information*, vol. 27, no. 1, pp. 49-59, 2022. Article (CrossRef Link)
[27] 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*, vol. 34, no. 3, pp. 993-1002, 2022. Article (CrossRef Link)
[28] A. Singh, "Taxonomy of Machine Learning Techniques in Test Case Generation," in *Proc. of 2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 474-481, 2023. Article (CrossRef Link)
[29] Y. D. Salman, N. L. Hashim, M. M. Rejab, R. Romli, and H. Mohd, "Coverage criteria for test case generation using UML state chart diagram," in *Proc. of AIP Conference Proceedings*, vol. 1891, 2017. Article (CrossRef Link).

SHAHID SALEEM is a lecturer at Lahore Leads University, Lahore. He earned his MSSE degree from COMSATS University Islamabad in 2017. With a keen interest in software testing and artificial intelligence, Shahid is dedicated to advancing research in these areas. He has contributed to various publications and projects, focusing on innovative solutions to real-world software engineering challenges.

SAIF U. R. MALIK is currently working as a Senior Researcher at Cybernetica AS, Estonia. He did his Ph. D. from North Dakota State University, USA in 2014. Previously, he has worked as an Assistant Professor at COMSATS University Islamabad Pakistan. Dr. Saif research interest includes Formal Methods and its application in and Large-Scale Computing Systems, Distributed Computing Systems, Fault-tolerance in Communication Networks, Data Centers, and Formal Analysis of Security and Privacy of Data.

BILAL MEHBOOB received the PhD degree from Monash University in Software Engineering and the M.S. degree in software engineering from the National University of Science and Technology (NUST), Pakistan. He is currently working as assistant professor in Superior University, Pakistan. His current research interests include Software engineering, Machine learning, Data mining, AI, Blockchain and Cybersecurity.

ROOBAEA ALROOBAEA received the bachelor's degree (Hons.) in computer science from King Abdulaziz University (KAU), Saudi Arabia, in 2008, and the master's degree in information system and the Ph.D. degree in computer science from the University of East Anglia, U.K., in 2012 and 2016, respectively. He is currently a Professor with the College of Computers and Information Technology, Taif University, Saudi Arabia. His research interests include human computer interaction, cloud computing, and machine learning.

SULTAN ALGARNI received the bachelor's degree (Hons.) in computer science from King Abdulaziz University (KAU), Saudi Arabia, in 2008, and the master's degree in information technology from University of New South Wales (UNSW), Australia, in 2014 and the Ph.D. degree in computer science from King Abdulaziz University (KAU), Saudi Arabia, in 2022. He is currently an assistant professor with the Faculty of Computing and Information Technology, King Abdulaziz University, Saudi Arabia. His research interests include information security, networking, IoT, SDN, and artificial intelligence.

ABDULLAH M. BAQASAH received the bachelor's degree in computer science from King Abdulaziz University (KAU), Saudi Arabia, in 2003, and the master's degree in information technology and the Ph.D. degree in computer science from La Trobe University, Australia, in 2009 and 2015, respectively. He is currently an Assistant Professor with the College of Computers and Information Technology, Taif University, Saudi Arabia. His research interests include semantic web, software engineering, the Internet of Things, cloud computing, artificial intelligence, and machine learning.

NAVEED AHMAD is a seasoned academic and R&D consultant specializing in BPM (Business Process Management), AI, NLP (Natural Language Processing), and Audio Analytics. Currently serving as Professor and Head of the Software Engineering Department at FAST-NU, he plays a crucial role in driving research and academic excellence in these domains. Ahmad holds a Ph.D. in process management from the University of Cambridge, where his research focused on software engineering, particularly process management and simulation. Throughout his academic journey, he has conducted extensive studies and utilized advanced techniques to tackle complex problems.

MUHAMMAD HASNAIN holds PhD from Monash University Australia. He is currently working as head of the computer science department Lahore Leads University, Pakistan. He is playing a crucial role to excel the students in research work. He has published several research articles in the area of Software testing, artificial intelligence, blockchain. Currently he is highly involved in the research on large language models (LLMs) and their applications in healthcare.