UML ACTIVITY DIAGRAM-BASED TEST CASE GENERATION

[1]Santi Swarup Basa, [2]Santosh Kumar Swain, [3]Durga Prasad Mohapatra
[1]Assistant Professor, [2]Professor, [3]Professor
[1]Computer Science, [2]Computer Science & Engineering, [3]Computer Science & Engineering
[1]North Orissa University, Baripada, [2]KIIT Deemed to be University, Bhubaneswar, [3]NIT, Rourkela, [123]India

Abstract : Thorough and effective testing is necessary to produce high quality and reliable software. The primary goal of testing is to cover as many as faults in the software. Test case design is an essential task in the testing activities of software development. Further, manual testing is time-consuming and labor-intensive for large software. Automated design of test cases is very important activity in software testing process. In this paper, the UML activity diagram is used for automatic test case design. Using the UML activity diagram, we present a test case generation methodology. First, a specific activity diagram is to be converted to a graph, called ACtivity Transition Graph (ACTG) is constructed from. Then by traversing the ACTG and using different coverage criteria, the test paths are generated. Finally, test cases are generated using our proposed Generation of Test Cases from the Activity Diagram (GTCAD) algorithm from the test paths. The approach presented in this paper shows all activity flows during testing of object-oriented software and finds all activity related faults as well as hybrid coverage based synchronous faults.

IndexTerms **- Software testing, UML diagrams, Activity diagram, Coverage criteria, Test case, Test suite.**

I. INTRODUCTION

Software systems are playing an increasingly prime role in society. There is always demand for good software systems in the society. New methods, techniques, and tools are developed to support testing activities of large and complex software systems. So there is a pressure to focus on quality issues. The poor quality software is not acceptable by the society as the company faces more losses. In Software Development Life Cycle (SDLC), a sequence of steps is followed to produce efficient and qualitative software [1]. In SDLC, testing has a very significant role in large and distributed software development.

In software testing, the program or application is executed with a set of test cases with the intention of finding bugs in the application or program software testing can be done in two ways. They are Black box and White box testing. Black box testing focuses on the output generated against any input [2]. The technique is also called as functional testing. But, the internal structure of software is to be verified in case of White box testing. That is also designated as structural testing [2]. Earlier testers went for manual testing, which is a very tedious. It is also very labor-intensive and time-consuming process. To avoid all those deficiencies, testers went for automated testing in the software development process. Steps followed in automated software testing consist of (1) Construction of test cases (2) test data selection (3) Test execution, and (4) comparing the obtained result with the expected results. A test case is a sequence of inputs and corresponding outputs that satisfying specific coverage criteria. The testers are to check with the help of test cases to see whether the requirements are satisfied with the developed software system or not.

A test suite includes a set of test cases. In black box testing, requirement specifications are used for generating test cases without knowing internal details of codes, where early detection of the fault can be possible before the design phase. In code-based testing, the source code is necessary and early fault detection in the specification or design phase is not possible. Model-Based Testing (MBT) is very important in finding faults much early than code-based testing. MBT is a better testing approach than code-based testing as it detects the errors at the early stage which requires less cost to fix them. Faults being detected at later stages could be detected much earlier with less expense in MBT [16]. Unified Modeling Language (UML) is considered as an important source for Model-Based test case design since it reduces complexity of software by using increased number of diagrams. It also inspires for early testing. So, Model-Based testing using UML can reduce time, effort and cost of testing if exploited satisfactorily and at the same time improve software quality.

Unified Model Language (UML) has now become standardized for modeling language for the development of the system [3, 27]. To generate test cases, different UML diagrams are used. UML models are too abstract to support test case design. It is a very challenging area in the software testing field. The researchers has been used UML based testing for many years to design test cases [30]. Many researchers are still working on this particular domain. The Activity Diagram (AD) is one of the UML behavioral diagrams which describe the sequential and concurrent activities. Integration level test cases can be generated from the AD. Sequential and concurrent flow of activities among the objects are shown in AD. It can be used for object flow modeling and control flow of objects within a problem domain. It is easy to understand due its simple representation like a flowchart. Each Use case of Use case diagram is to be represented by an activity diagram. The degree of completeness of the test scenarios are insured by the AD. The scenarios of events for one use case are shown by this diagram. In this work, we emphasize on test case generation using the activity diagram. Activity Diagram structures are very much nearer to flow chart of the code [21]. So the low-level test data can be found from an activity diagram like the code. For generating test cases, activity and hybrid coverage criteria are used in our approach. In recent years, test case generation from UML activity diagram has gained the attention by many researchers [22-25]. In this paper, we give emphasis on UML behavioral model, like activity diagram.

In this paper, our approach focuses on activity diagram based test case generation. First, the activity diagram is constructed by using RSA tool for the specific use case and generates the XMI code of it. Then we parse the XMI code using XMI parser developed in JAVA to analyze and construct directed graph called ACTG. Then, test paths are generated from the directed graph using All Transition Path (ATP) algorithm. Lastly, coverage based test cases are generated automatically using our proposed algorithm named *Generation of Test Cases from the Activity Diagram (GTCAD)* algorithm from the test paths. In this paper, we have taken ATM Withdrawal Operation as an example case study to illustrate our approach. Results show that our approach semi-automatically generates test cases.

The remaining part of this paper is structured as follows: In section 2, we discuss the diagrams of UML, including activity diagram and all definitions associated with test case generation. Section 3 discusses the analysis of related work. In section 4, we describe our proposed framework for test case generation. We compares our approach with related work in section 5. Section 6 concludes the paper and highlights our future work.

II. BASIC CONCEPTS

2.1 UML Diagrams

The Unified Modeling Language (UML) is a modeling language which specifies and visualizes the complex system It also helps in construction and documentation of software system[27]. A collection of diagrams are used for modeling complex systems. UML defines nine models to show the software intensive system. It is also the baseline for early testing process.

UML diagrams are classified into three broad categories i.e. User's view, Structural view and Behavioral view. Use case diagram is considered as user's view diagram. Static aspects of the system are represented by stable structural diagrams. They form the main structure. Classes, interfaces, objects, components, and nodes are static parts of UML Class, Object, Component and Deployment diagrams are considered as structural diagrams.

The dynamic behavior of a system is represented by Behavioral diagrams. It contains the changing parts of a sydtem.UML has the following four types of behavioral diagrams:

- Use case diagram, Activity diagram, Sequence diagram, Collaboration diagram, Statechart diagram

In this paper, we have focused on activity diagram for generating test cases. In the next section, we briefly discuss the UML activity diagram.

2.2 Activity Diagram (AD)

An activity diagram is one of behavioral diagram of UML. Dynamic flow of system is represented by this diagram. It is similar to a flowchart that shows the flow from start to end activity. The operations of one event of system can be represented by an activity. Activity diagram shows the actions of an object or group of objects through flow of activities. The activities are the state of performing operations. AD basically defines the internal behavior of an operation and also represents the conditional or parallel activities. [23]. It gives importance on the flow of sequence and concurrent control from one activity to another activity. UML activity diagram includes activities, activity flows, signal senders, decisions, forks, joins, objects, swim lanes, and receivers. Action and activity states are represented with round-cornered boxes. The edges represent flow of control, message and signal etc. Swim lanes are represented by horizontal bars to separate the participating objects to carry out the activity as a whole. Diamonds represent decisions are shown by diamond symbol. The fork or join of concurrent activities are represented by bars. Start and End activity are represented by solid circle.

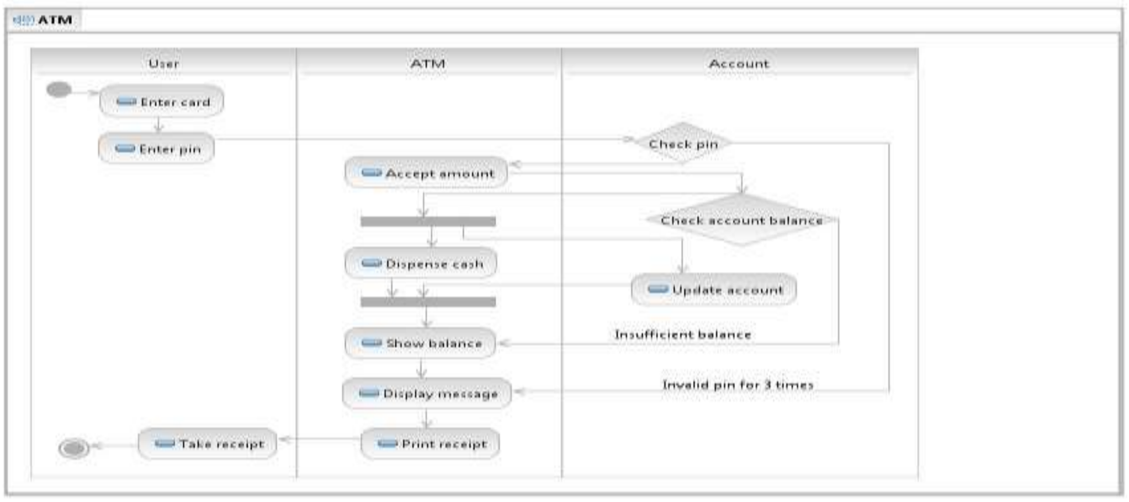


Fig.1 Activity Diagram for ATM Withdrawal Operation Example

Figure 1 represents the activity diagram for ATM Withdrawal Operation example. The activity such as Enter card, Enter pin and Accept amount etc are represented by rounded rectangles. This example contains three swimlanes separating three objects: User, ATM, and Account.

2.3 Graph Concept

A G**raph** is collection of a set of vertices and edges and represented by G = (V,E) . The set V contains the elements of vertex and E contains the elements of edges [19]. A graph can be **directed or undirected**. In a directed graph, the edges have orientation and also called as digraph. On the other hand, the edges have no orientation in case of undirected graph [4, 5]. A directed graph is acyclic if it has no cycle. A **Multigraph** is a graph in which many the edges connected to a vertex and many forms loops. The directed multigraph is known as multidigraph.

A sequence of vertex and edges from start node to end node is called a P**ath**. If vertex and edges are repeated the path is called as a **Walk**. The path is called as cyclic path if the start vertex is same as end vertex.

Graph traversal is a process by which one can move through a graph visiting every vertex at least once. Depth-First Search (DFS) and Breadth-First Search (BFS) are considered as two techniques for graph traversal. [17]. DFS is an algorithm visits the child vertices (depth-wise) before its neighbor vertices (its breadth vertices) and of each visited vertex is added to a stack (LIFO) [5]. On the other hand, BFS visits all the neighbor vertices (breadth-wise vertices) before exploring its child (its depth wise vertices). It traverses from a specified vertex to another finding shortest path using the queue (FIFO) data structure.

Definition 1 *(ACtivity Transition graph (ACTG)):* ACTG is a directed graph representing the relationships among activities in parent-child way. In ACTG, Node represents activity and Edge represents the transition between activities to activity. The ACTG contains a unique start node and one or more than one stop node.

.

Definition 2*(Activity Transition Path)*: An Activity Transition Path is a sequence of nodes from the start node to the stop node in the ACTG. The sequence of edges (transitions) and nodes (activities) obtained through Traversal of the ACTG from start node to the desired stop node is known as the activity transition path. An Activity Transition Path P from vertex n_i to n_k contains of nodes in order n_i, n_{i+1}… n_k, where for each pair of adjacent nodes ($n_{i+j,}$ n_{i+j+1}), an edge in ACTG, for 0<i<k<k-I is present.

2.4 Testing Coverage Concept

Definition 3 (*Coverage Criteria*): Coverage Criteria is the percentage covered to test the functionality of the software. It includes information about the portion of an application program executed when we execute it with the test suits. It checks the testing coverage of parts of the application program.

Definition 4 (*Activity coverage for a test case*): It covers activities with some adequate transitions. Every activity is reachable from the initial activity. The activity coverage ratio is calculated from the following:

(number of activities covered / total number of activities) * 100.

Definition 5 (*Activity Transition coverage*): Each activity-transition in the ACTG is taken at least once. Transition coverage is calculated from the following:

(covered activity-transitions with test-suits/ total number of activity-transitions in the ACTG) * 100.

Definition 6 (*Activity Transition Path coverage*): It is the sequence of activity-transition that requires that all simple activity transition paths in the activity diagram be covered.

Transition path coverage= (simple transition paths traversed/ total simple transition paths) *100

Definition 7: Cyclomatic Complexity (CC) is used to know the minimum number of independent test paths designed for each activity diagram. Cyclomatic complexity can be computed via using two notations, number of edges (E) and number of nodes (N). Equation 1 shows the formula to calculate CC [2].

$$CC=E-N+2 \qquad (1)$$

Definition 8 (Hybrid Coverage criterion): **H**ybrid Coverage can be defined by the following:

“A test suit T satisfies the hybrid coverage criterion if and only if T contains all activity transition paths, all predicates and all branches in an activity diagram and the test suit is generated by applying cyclomatic complexity technique”.

III. RELATED WORK

Meiliana et al. [28] used UML sequence diagram , activity diagram and combination graph as SYTG. They applied a modified Depth First Search algorithm on the combined UML diagrams and combination graph to generate optimized test cases. They have also presented some experimental result that shows sequence diagrams can produce better results.

Ikram et al. [29] intended to analyze and survey of various procedures of activity diagram based test case generation techniques. Semi-automated and automated techniques incorporated different testing approaches such as integration testing, model driven and grey-box for reviewed purposes. They also reviewed different techniques used in UML activity diagram for test case optimization and prioritization.

Trong [18] presented a method using class, activity and sequence diagrams. Adequacy criteria for those diagrams are also tested in their work. These criteria are not explicitly defined but presented as a general discussion. They proposed to cover all activity edge of an activity diagram using activity edge criterions.

A method for automatically generating test cases was presented by Rayadurgam et al. [6]. They used a formal approach to define structural coverage and represented how a model checker can be used for modified and decision coverage to generate a sequence of test cases. That checker is used to verify the a group of properties of UML models.

Looking towards the Offutt's state-based technique, Abdurazik et al. [7, 8] defined a test data generation method. In that method, they mentioned that test specification describes the test cases and testing should cover the requirements. They used test data generation method to generate test cases using a UML state chart diagram. They proposed TSL language and verified some elements like input and expected output. The elements are derived from preconditions and events in the UML model.

Prasanna et al. [9] proposed that test case generation is an essential activity in testing and by using UML diagram test cases can be generated. They applied genetic algorithm (GA) on the class diagram by using model based approach and DFS algorithm is used for traversal. They observed that efficiency can be improved by joining GA with testing. They also showed that their methodology is useful for early error detection in SDLC

Kansomkeat et al. [10] proposed a method test case sequences from event sequences of testing flow graph (TFG). The TFG is developed from state chart diagram. They have taken states and transition coverage criteria for the generation of sequences of test case.

Swain et al. [11] used the UML state chart diagram to automatically generate the test cases. Their generated test cases achieves, the states and transition coverage's. They used the predicates of state chart diagram for test case generation further they used function minimization technique for minimization of test cases.

Four test generation approaches were presented by Utting et al. [13]. They went for the testing process with domain model for test data generation, which described the domain of input data.The second approach focused on generation of test data from the environment model in which, it described the operation frequencies of the System Under Test (SUT). The third one is the behavioral model which described the irregularities and inconsistencies of the SUT including oracle information to the expected output. The fourth approach did not include any model but the high-level abstraction is used to describe instead of using detail implementation.

Mingsong et al. [14] proposed a technique to generate random test cases for a certain JAVA code. The code execution traces are found by running the JAVA code. Then they generated reduced set of test cases using activity diagram. Lastly they checked the consistencies between code execution test and the model behavior by using the generated test cases.

In [15], the state machine is transferred to a test flow graph and test cases are generated using various popular coverage criteria such as All Transition(AT), Round Trip Path (RTP), All Transition Path(ATP) etc. ATP was proved to be a stronger coverage criterion compared to AT and RTP in their work. Another study found that RTP to be not sufficient to detect the faults as 10% to 34% of faults is undetected. This is for a weaker structure of RTP. It is concluded that comparing with random testing, RTP is detecting 88% of faults and with random testing, it is only detecting 66% of faults. Combining RTP with, the fault detection efficiency can be improved if the Category-Partition (CP) method is combined with RTP. So, on comparing ATP with all the above criteria, it is resolved that RTP is better when it is combined with other methods and ATP is almost the same or better than RTP.

IV. PROPOSED FRAMEWORK FOR TEST CASE GENERATION

In this section, our proposed framework to generate test cases from an activity diagram is discussed. We have named our approach, *Generation of Test Cases from Activity Diagram (GTCAD)*. Our proposed framework is shown in Fig. 2. The proposed approach includes the following steps.

4.1 Construct activity diagram for the particular use case of the system.
4.2 Generate XMI Representation.
4.3 Convert the activity diagram into an activity transition graph (ACTG).
4.4 Generate activity transition paths by traversing ACTG using DFS.
4.5 Generate test cases from transition paths using transition coverage criteria.

We explain above steps in detail in the following subsections. Each step is also illustrated with a running example of ATM withdrawal operation.

4.1 Construct activity diagram for the particular use case of the system.

In UML, Use case diagram represents the user's view of the system. Each use case in use case diagram represents an event. One or more activity diagram is required to be drawn to show the scenario of each use case. Here, we construct an UML activity model that is used to represent one or more requirements of a system. We use RSA tool to draw the activity diagram. The activity diagram of ATM withdrawal operation is shown in Fig. 1.

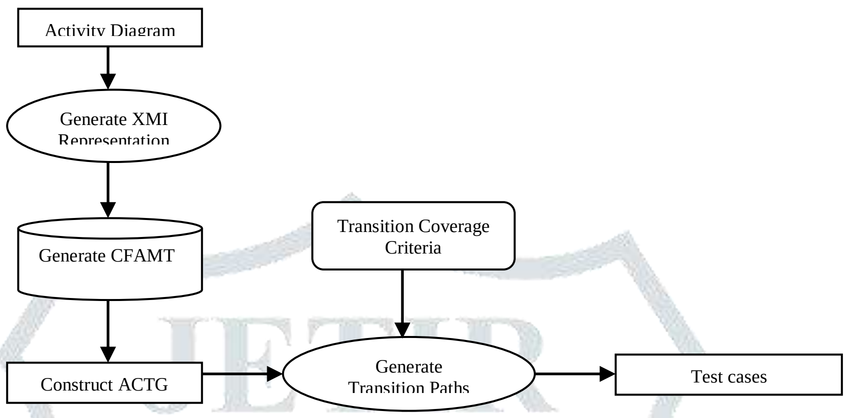
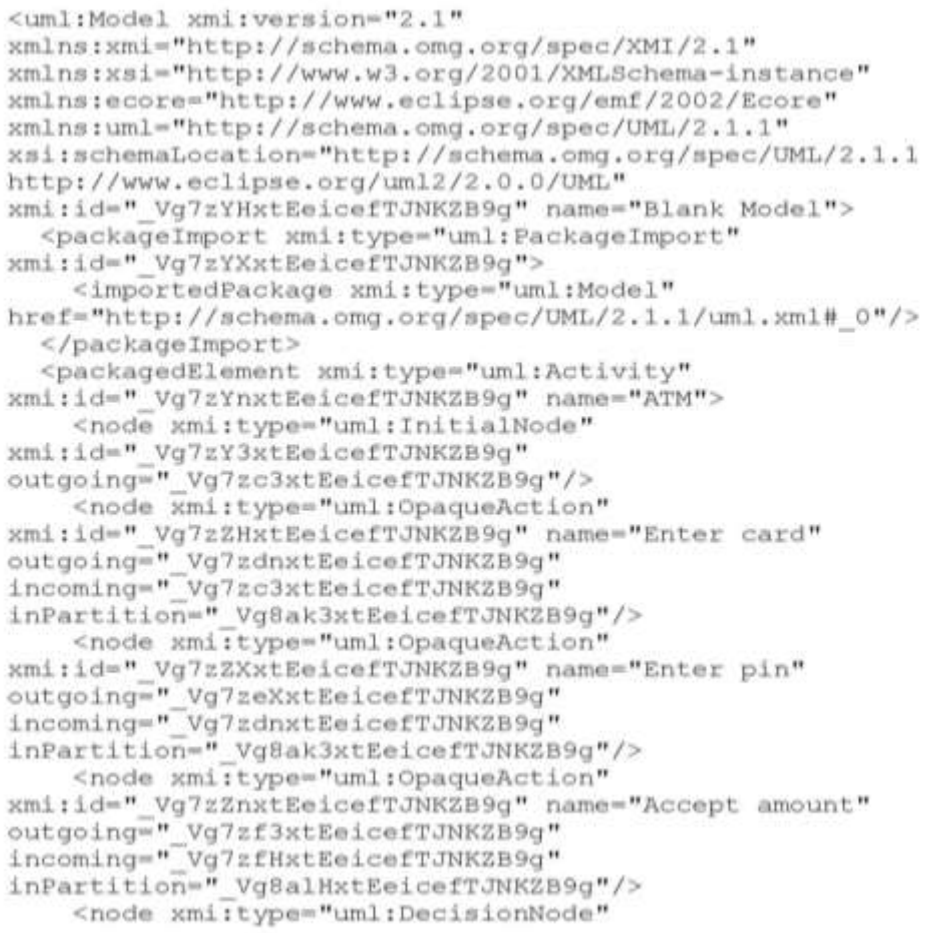

Activity Diagram
Generate XMI Representation
Generate CFAMT
Transition Coverage Criteria
Construct ACTG
Generate Transition Paths
Test cases

Fig.2 Schematic Diagram of Proposed Framework for Generating Test Cases

4.2 Generate XMI Representation.

After developing the activity diagram, we export the XMI representation of the activity diagram which is used as the input in the extractor. The XMI representation of the activity diagram of ATM withdrawal is shown in Fig. 3.



```
<uml:Model xmi:version="2.1"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:uml="http://schema.omg.org/spec/UML/2.1.1"
xsi:schemaLocation="http://schema.omg.org/spec/UML/2.1.1
http://www.eclipse.org/uml2/2.0.0/UML"
xmi:id="_Vg7zYHxtEeicefTJNKZB9g" name="Blank Model">
  <packageImport xmi:type="uml:PackageImport"
xmi:id="_Vg7zYXxtEeicefTJNKZB9g">
    <importedPackage xmi:type="uml:Model"
href="http://schema.omg.org/spec/UML/2.1.1/uml.xml#_0"/>
  </packageImport>
  <packagedElement xmi:type="uml:Activity"
xmi:id="_Vg7zYnxtEeicefTJNKZB9g" name="ATM">
    <node xmi:type="uml:InitialNode"
xmi:id="_Vg7zY3xtEeicefTJNKZB9g"
outgoing="_Vg7zc3xtEeicefTJNKZB9g"/>
    <node xmi:type="uml:OpaqueAction"
xmi:id="_Vg7zZHxtEeicefTJNKZB9g" name="Enter card"
outgoing="_Vg7zdnxtEeicefTJNKZB9g"
incoming="_Vg7zc3xtEeicefTJNKZB9g"
inPartition="_Vg8ak3xtEeicefTJNKZB9g"/>
    <node xmi:type="uml:OpaqueAction"
xmi:id="_Vg7zZXxtEeicefTJNKZB9g" name="Enter pin"
outgoing="_Vg7zeXxtEeicefTJNKZB9g"
incoming="_Vg7zdnxtEeicefTJNKZB9g"
inPartition="_Vg8ak3xtEeicefTJNKZB9g"/>
    <node xmi:type="uml:OpaqueAction"
xmi:id="_Vg7zZnxtEeicefTJNKZB9g" name="Accept amount"
outgoing="_Vg7zf3xtEeicefTJNKZB9g"
incoming="_Vg7zfHxtEeicefTJNKZB9g"
inPartition="_Vg8alHxtEeicefTJNKZB9g"/>
    <node xmi:type="uml:DecisionNode"
```

Figure 3: XMI representation of Activity diagram for ATM Withdrawal Operation

4.3 Construct ACTG.

In this subsection, we explain procedure for constructing ACTG from an activity diagram . An activity transition graph is a directed graph having the start and stop nodes. Other nodes may be of decision, fork, join and merge nodes. In addition, it contains the flow in between one activity to other activity and guard conditions. The ACTG is created by traversing the activity diagram from starting node to end node. We convert the activity diagram into a graph, called activity transition graph (ACTG) using the following steps:

- For each condition, create an entry in the Control Flow Activity Mapping Table (CFAMT). The CFAMT is constructed by the symbols by unique number to each activity of activity diagram. The CFAMT is a three columns table representing Activity Name, Symbol Name and the corresponding controlling activity which performs the particular activity. The CFAMT for Activity Diagram for ATM Withdrawal Operation given in Fig.1 is shown in Table 1.
- Then, using the CFAMT, create nodes in the ACTG.
- Loop statements are considered as conditional statements.
- One entry is made For each concurrent execution activity in CFAMT. So, ACTG represents each activity by different paths.

A directed graph called ACtivity Transition Graph (ACTG) is generated from CFMAT. The extractor i.e. Java code takes the XMI as the input and develops the intermediate graph which shows the different activities present in the activity diagram as nodes and the flow between those activities as the edges between these newly generated nodes. The Control Flow Activity Mapping Table (CFAMT) of ATM Withdrawal Operation is given in Table 1. Then using CFAMT, Activity Transition Graph (ACTG) is generated which is shown in Fig. 4.

Table 1 Control Flow Activity Mapping Table (CFAMT) of ATM Withdrawal System

Activity Name	Symbol Name	Controlling entity	Activity Name	Symbol Name	Controlling Entity
Enter Card	1	User	Update account	8	Account
Enter Pin	2	User	Merge	9	ATM
Check Pin	3	Account	Show balance	10	ATM

Accept amount	4	ATM	Display message	11	ATM
Check Account Balance	5	Account	Print receipt	12	ATM
Fork	6	ATM	Take receipt	13	User
Dispense cash	7	ATM	Exit	14	ATM

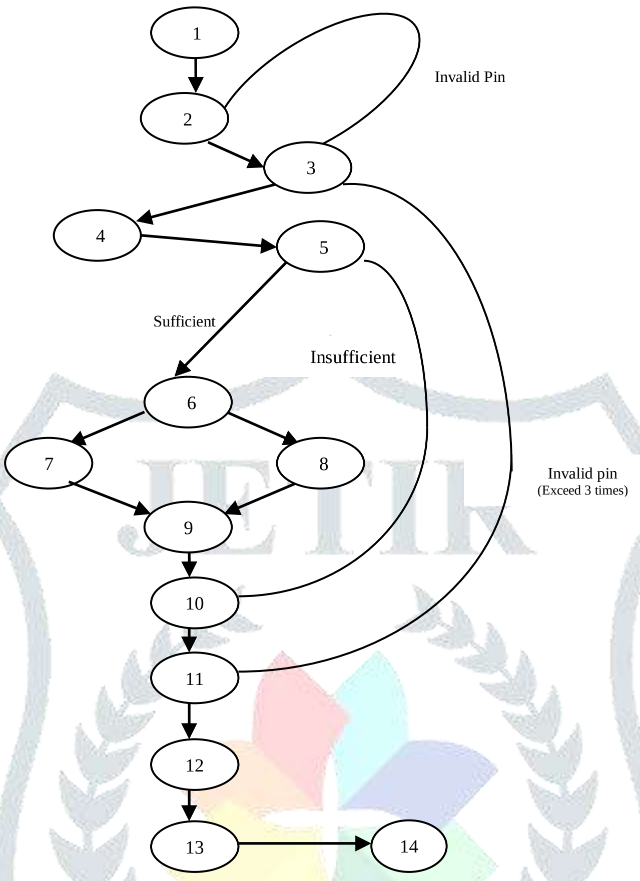


Fig.4 Activity Transition Graph (ACTG) of ATM Withdrawal Operation

4.4 Generate Activity Transition Path

All activity transition paths are generated from the ACTG. To do this, an algorithm named *All Transitions Path (ATP)* Algorithm is proposed to generate all activity transition paths. Our approach traverses the ACTG using DFS algorithm in order to extract all the possible test paths. All decision nodes of activity diagram are to be traversed to generate the activity transition paths. Then we traverse each branch of the each decision node by satisfying the branch coverage criterion. At the time of traversal, the algorithm will store the predicates values of guard conditions to generate the test cases along the path. During the traversal, the algorithm checks all the loops only once. So, all the simple transition paths are generated. These test paths are generated using transition coverage criteria. We obtain All-Transition Paths from the start node to a stop node in the ACTG. We use the path enumeration algorithm for the purpose of finding test paths. For All Transition Paths, the proposed ATP Algorithm is given below.

Algorithm1: *ATP Algorithm*

```
Input   : An ACTG
Output: Activity transition paths

1.  LFlag=0;
2.  V_node=0;
3.  Begin
4.  For each V_node during the traversal, node_count will be incremented by 1 and push the node into stack S;
5.  If  node_count = 2 then
6.      Set LFlag=1;
7.  end
```

```
8.  If  LFlag=1 and node_count = 3 then
9.      Repeat pop(S) until top(S ) = backtrack node; // Backtrack to last 'Decision Node'
10. end
11. If  type(node) = 'Fork ' then
12.     Traverse 'Fork  Node' sub tree and enter the 'Activity Node' into Q until type 'Join Activity Node'/'Final Activity
        Node' is visited for out -degree ('Fork Node' ) times; Enter 'Join Activity Node ' into queue Q;
13.     while Empty(Q) do
14.         q_node=Delete (Q);
15.         if (Type(q_node)!='Join Activity Node ') then
16.             Push(q);
17.             Explore all $d_q$ = descendents of q where q is the root with maximum depth 2 and push(S)= $d_q$ if type ( $d_q$ )
                ! = 'Activity Node' ,  'Final Activity Node ' and 'Join Activity Node';
18.         end
19.     end
20. end
21. if  Type( node) ='Final Activity Node ' then
22.     Pop(S) to an array containing the sequence of nodes representing activity path; where last node is the Top(stack),;
23.     Repeat pop(S) until top(S ) = the backtrack node; // Backtrack to type(node) =' Decision Node '
24.     Visit the child whose node_ count = 0;
25.     If empty(S) then stop else Goto Step 4 ;
26. end
    end
```

By applying the above proposed algorithm on ACTG of ATM Withdrawal Operation given in Fig,4, we generate the Activity transition paths. These Activity transition paths are used as test cases. The generated test paths are as follows:

Test Path 1:- 1--->2--->3--->2--->3--->11---> 12 --->13 --->14
Test Path 2:- 1--->2---> 3---> 4--->5--->6--->7--->9--->I0--->11--->12 --->13 --->14
Test Path 3:- 1--->2---> 3---> 4--->5--->6--->8--->9--->I0---> 11--->12---> 13---> 14
Test Path 4:- 1--->2--->3--->11--->12 --->13 --->14
Test Path 5:- 1--->2--->3--->4--->5--->I0--->11---> 12 --->13 --->14

4.5 Generate Test Cases

We generate test cases following the *activity transition path coverage* criterion. To generate test cases, the algorithm developed is named as *Generation of Test Cases from Activity Diagram (GTCAD) algorithm*. For All Transition Path, proposed *GTCAD Algorithm* is given below.

```
Algorithm: Generation of Test Cases from Activity Diagram (GTCAD)
Input     : ACTG for Activity diagram, All Transition Paths
Output    : Test suite T
//CONDpre=precondition
//In ($a_1, a_2, a_3 ... a_i$) = input value set
//Out ($d_1, d_2 ... d_m$) = output value set
//CONDpost= post condition of the method
// C_node=Current _node; F_node=Final Node
// S is the start node
Steps                    :
1.  AP [ ] = All transition paths of ACTG= (AP [1], AP [2] ….AP[n])
2.  For each path AP[i] € AP do
3.      C_node  = S
4.      $CONDpre_i$= Find PreCond(S)
5.      $t_i=\phi$
6.      while C_node ≠ F_node of  AP[i] do
7.      event C_node=Find_Event(C_node)
8.      If  E ≠ Guard condition
9.      t={ CONDpre, In($a_1, a_2, a_3, ......, a_i$), Out($d_1, d_2, ..., d_m$), CONDpost }
10.     end if
11.     if (E= = Guard condition) then
```

```
12.         E (Val) = ($C_1, C_2 \ldots C_i$) // Set of values on the path P [i]
13.           t = { CONDpre, In ($a_1, a_2, a_3 \ldots a_i$), Out ($d_1, d_2 \ldots d_m$), E (Val), CONDpost }
14.       end if
15.         $t_i = t_i \cup t$ //  test case t is added to set $t_i$
16.         C_node = node->next // Move to the next node
17.       end while
18.         Get output and CONDpost$_i$ for the F_node of AP [i]
19.       t = { CONDpre$_i$, In, Out, CONDpos$_i$}
20.       $t_i \leftarrow t_i \cup t_i$
21.       TC$\leftarrow$ TC $\cup$ $t_i$
22.     end for
23.     return(TC)
24.     end
```

Using GTCAD algorithm we generate the test cases from the test paths presented in subsection D for ATM Withdrawal Operation. Test cases are generated which covers activities in activity diagram on the basis of activity and hybrid coverage. Here five different test cases are generated. The generated test cases are explained below.

Test Case 1 {Input: Invalid Pin No, Output: Return ATM card with Invalid Pin message}
Test Case 2 {Input: Valid Pin No, Valid Amount, Output: Dispense Cash, Update Account, Print Receipt}
Test Case 3 {Input: Valid Pin No, Valid Amount, Output: Dispense Cash, Update Account, Print Receipt}
Test Case 4 {Input: Invalid Pin No(for 3 times), Output : Card blocked(for 24 hours)and Return with failure message }
Test Case 5 {Input: Valid Pin No, Valid Amount, Output : Return card with failure message 'Insufficient Funds'}

Using All Transition Path coverage criteria, the generated test suite is given by TS = (T1, T2, T3, T4, T5,)

To validate the above test cases we, have taken several test data which are given in Table 2

Table 2 Test data for ATM Withdrawal Operation

Test case ID	Input		Expected Output	Test Path Covered
	Pin	Amount		
1	*034	N.A	Invalid pin/ Unsuccessful	1
2	*234,#045,2*3#5	N.A	Invalid pin(3times)/Unsuccessful/Card blocked	4
3	1234	510	Invalid amount/Amount should be multiple of 100	5
4	1234	90	Invalid amount/Amount should be multiple of 100	5
5	1234	10000	Insufficient balance (if minimum balance amount in bank is 5000) / Unsuccessful / Show balance	5
6	1234	30000	Insufficient fund in ATM/ Unsuccessful	5
7	1234	60000	Minimum limit exceeds/ Unsuccessful / Invalid amount	5
8	1234	85000	Minimum limit exceeds(per day)/ Unsuccessful/ Invalid amount	5
9	1234	5000	Successful/Print receipt/Dispense cash/ Server problem	5
10	1234	4000	Cash Dispensed/ Balance Display/ Receipt print	2,3

V. COMPARISON WITH RELATED WORK

In our proposed work, the CFMAT is formed from ACTG, representing all the activities in the certain activity diagram. Branches of AD are covered in the generated test cases. Hence branch coverage criterion is satisfied. They also confirm all the conditions/predicates coverages and all the basic paths because all the loops checked only once by the model. The test cases generated are validated by the cyclomatic complexity techniques to satisfy the cyclomatic complexity coverage criterion. Our approach is also satisfies hybrid coverage criterion which includes the branch,, full predicate ,all basic path and the condition coverage criterion.

Many reported methods augments UML model with certain annotations to facilitate the test case generation. [29,30]. Offut et.al. [8] presented an idea of generating unit level test cases using state chart diagrams. In our research work, we focused cluster level testing which involves object interactions shown in activity diagram. We have not used any additional formalism in the process which requires additional effort in test case generation. In comparison to [8,29,30]our work is advantageous one.

Abdurazik et. al. [8, 20] used requirement specifications for generating test cases using state chart diagram. They used a like TSL and checked the elements found from the preconditions and transition events. On contrary, in our work we proposed the ACTG, formed from the CFMAT which shows the different activities present in the activity diagram as nodes and the flow between those activities as the edges between these nodes. The faults related to activity interaction among objects are detected from Branches behavioral paths. Hence, branch coverage criterion is satisfied.

In comparison to the related work [9,18], Our work has the advantage that no redundant edges are found from the test paths generated, which leads to the reduction of time, cost and testing efforts for software testing. Our proposed methodology is meant for detecting the faults like incorrect activity response , arguments passing, activities sequencing. It also uncovers faults leading to missed and inappropriate control flows among the activities. We have used the activity as well as hybrid coverage criterion to derive the test cases.

Sarma et al. [32] have integrated the intermediate graphs generated from use case and sequence diagram to form a system testing graph (STG). Then test scenarios are generated by traversing the STG. Their method was a manual while we provide a semi-automatic method. Our method is advantageous to this in the sense that using our method we explore cluster level testing. Comparing to the coverage aspects to their approach, we provide a stronger coverage of activity path coverage as well as hybrid coverage which achieve activity synchronization faults and object interaction faults.

Pechtanun et al. [33] used activity diagram for the generation of test cases. They transformed the AD into Activity Convert (AC) grammar. Test cases are generated from AC grammar. But in comparison to that, we have used a combined model which achieves message activity path coverage. Our approach is capable of reveling activity synchronization faults as well as object interaction faults.

V. CONCLUSION

Automated test case generation using UML diagrams is an approach to identify faults in the implementation and to reduce testing effort. It improves design quality and find faults in the early stage before implementation, which reduces software development time. In this paper, the focus is to generate sequence of test cases using activity diagram by implementing different coverage criteria based on activity synchronization and hybrid coverage. Using, the activity diagram, early fault detection can be made in the design phase and tester can observe the dynamic behavior of the model. This approach is about synchronous testing and finding of synchronous faults, as testers always try to find out all such errors like missing activity, missing transition faults in the software. Sometimes the process is running according to the stages but not working perfectly. For example the ATM machine, which runs from the starting state to the final state but the money, is not transacted. Testers repeatedly try to find out such errors. Our future work will include test case generation using other UML diagrams. The approach can also be applied with other soft computing techniques like Genetic Algorithm, Neural Network, and Fuzzy Logic etc. to increase the accuracy of the test cases and optimize the no. of test cases.

REFERENCES

[1] R. Pressman, *Software Engineering: A Practitioner's Approach*: Mc-GrawHill, 2005

[2] http://softwaretestingfundamentals.com/

[3] Object Management Group: The Unified Modeling Language UML 1.5 Technical Report formal/03-03-01, The Object Management Group (OMG), 2003.

[4] Jensen, J.B. and Gutin, G.Z. 2008. Digraphs: Theory, Algorithms and Applications", 2nd edition.Springer Publishing Company, Incorporated,.

[5] Boni, G. and Juan, C. D. 2011. Automated Functional Testing based on the Navigation of Web Applications. Workshop on Automated Specification and Verification of Web Systems, EPTCS 61, 49–65.

[6] Rayadurgam, S. and Heimdahl, P.E.M. 2001. Test-Sequence Generation from Formal Requirement Models", Proceedings of the 6th IEEE International Symposium on High Assurance Systems Engineering (HASE'01),

[7] Offutt, J. Liu, S. Abdurazik, A. and Ammann, P. 2003. Generating Test Data from State-based Specifications. ISE Department, George Mason University, USA,

[8] Abdurazik A. and Offutt, J. 1999. Generating Test Cases from UML Specifications.

[9] Prasanna, M. and Chandran, K.R. 2009. Automatic Test Case Generation for UML Object Diagrams Using Genetic Algorithm. Int. J. Advance. Soft comput. Appl., vol.1, no. 1, pp. 19-32, July.

[10] Kansomkeat, S. and Rivepiboon, S. 2003. Automated- Generating Test Case Using UML Statechart Diagrams ", SAICSIT.

[11] Swain, R. Panthi, V. Behera, P.K. Mahapatra, D.P. 2012. Automatic Test Case Generation Based on State Machine Diagram. International Journal of Computer Information Systems, vol.4, no.2, pp. 99-124,

[12] Weissleder, S. 2010. Simulated Satisfaction of Coverage Criteria on UML State Machines. Third International Conference on Software Testing, Verification and Validation.

[13] Utting, M. Pretschner, A. and Legeard, B. 2006. A taxonomy of model-based testing" Working Papers 2006. Department of Computer Science, the University of Waikato (New Zealand), April.

[14] Mingsong, C., Xiaokang, Q., and Xuandong, L. 2006. Automatic test case generation for UML activity diagrams. In Proceedings of the 2006 Inter-national workshop on Automation of software test, Shanghai,pp. 2–8, China.
[15] Holt, Elisabeth, N. Briand, L.C. and Torkar, R. 2014. Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study. Information and Software Technology 56.8 : 890-910,.
[16] CemKaner, JD. 2007. Test case prioritization in a specification -based testing environment. Proceedings of Florida Institute of Technology Department of Computer Sciences STAR East.
[17] Bondy, J.A. 1976.Graph Theory With Applications. Elsevier Science Ltd.
[18] Trong, T.H. 2004. A Systematic Approach to Testing Design Models. In Doctoral Symposium, 7th International Conference on the Unified Modeling Language, Lisbon, Portugal, 10-15, October
[19] Diestel, R. 2005.Graph Theory. Springer,
[20] Abdurazik, A. and Offutt, J. 2000. Using UML collaboration diagrams for static checking and test generation. In Proceedings of the 3rd International Conference on the UML. Lecture Notes in Computer Science, Springer-Verlag GmbH, vol. 1939, pp. 383 - 395, York,U.K., October.
[21] Rumbaugh, J. Jacobson, I. and Booch, G. 2001. The Unified Modeling Language reference manual, Addison-Wesley.
[22] Kundu, D. and Samanta, D. 2009. A Novel Approach to Generate Test Cases from UML activity Diagrams , Journal of Object Technology, Vol. 8, No.3, pp.65-83, May-June.
[23] Linzhang, W. Jiesong, Y. Xiaofeng, Y Jun, H. Xuandong, L. and Guoliang, Z. 2004. Generating test cases from UML activity diagram based on gray-box method , In 11th Asia-Pacific Software Engineering Conference (APSEC04), pp. 284-291.
[24] Mingsong, C. Xiaokang, Q. and Xuandong, L. 2006. Automatic test case generation for UML activity diagrams , In 2006 international workshop on Automation of software test, pp. 2-8,.
[25] Rudram, C. 2003. Generating Test Cases from UML, University of Sheffield ,technique report, available at http://www.dcs.shef.ac.uk.
[26] Rhmann, W. and Saxena, V. 2016. Optimized and Prioritized Test Paths Generation from UML Activity Diagram using Firefly Algorithm. International Journal of Computer Applications, pp.16-22, vol.145 – No.6, July.
[27] Booch,G. Rumbaugh, J. and Jacobson, I. 1999. The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, Massachusetts,.
[28] Meiliana. Septian, I. Alianto, R. S. Daniel. Gaol, F. L. 2017. Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm. In proceedings of the 2nd International Conference on Computer Science and Computational Intelligence(ICCSCI), pp.629-637, 13-14 October.
[29] Ikram, M. T. Butt, N. A. Hussain, A. and Nadeem,. A. 2015. Testing from UML Design using Activity Diagram: A Comparison of Techniques. International Journal of Computer Applications, vol.131, No.5, pp. 42-47 December.
[30] Cavarra. Crichton C. and Davies, j. 2004. A method for the automatic generation of test suites from object models. Information and Software Technology, 46(5): 309-314.
[31] Trong, T.D. 2003. Rules for Generating Code From UML Collaboration diagram and Activity Diagrams. Master's Thesis, Colorado State University, Fort Collins, Colorado.
[32] Sarma, M. and Mall, R. 2007. Automatic Test Case Generation from UML Models. In Information Technology, (ICIT 2007). 10th International Conference on , IEEE, pp. 196–201.
[33] Pechtanun, K.. and Kansomkeat, S. 2012. Generation Test Case from UML Activity Diagram Based on AC Grammar. In Computer & Information Science (ICCIS), 2012 International Conference on , vol. 2, IEEE, pp. 895– 899.