A Proposed Test Path Generation Algorithm based on UML Activity Diagram

Aye AyeKyaw, MyatMyat Min
University of Computer Studies, Mandalay
ayeayekyaw2009@gmail.com,myatiimin@gmail.com

Abstract

Software testing is an activity of finding defects during execution time for a program so that non-defect software can be got. Software testing plays a vital role in developing software that is free from bugs and defects. Manual test is a cost and time consuming process although it may find many defects in a software application. If the testing process could be automated, the cost of developing software could be reduced obviously within a minimum amount of time. The most critical part of the testing process is generation of test paths. The paper proposes an approach for automatic Model-Based best test path generation. In this approach, all possible test paths are directly generated from XMI (XML Metadata Interchange)filebased on the activity diagram and a test path from these generated test paths is optimized. The proposed system will reduce the processing time, and choose the best test path without dependence on others.

1. Introduction

Software testing is an essential part of the software development life cycle (SDLC). Software testing is an activity that should be done throughout the whole development process. Software organizations spend nearly half of the total development time and cost of the software in testing related activities. Software testing should be performed efficiently and effectively, within the budgetary and scheduled limits.Testing can be performed on requirements; design and implementation phase in the software development life cycle. The most of disseminating errors can be eliminated and prevented before to the next phase if testing is performed in the initial phase in SDLC.The efficient and effective approaches are needed still although there are many existing automatic test case generation approaches.

The testing process consists of three parts: test case generation, test execution, and test evaluation. Comparing with the other two parts, test case generation is more challenging and difficult [2]. Generating test data from the high level design notations has several advantages over code-based test case design.Testing based on design models has the advantage that the test cases remain valid even when the code changes a little bit [8].

Several heuristic methods are available for effectivegeneration of test cases. The heuristics can be mainly dividedinto two categories: Black box testing and White box testing.There are classified the testing techniques generally as shown in figure 1.

The rest of the paper is organized as follows: The next section discussesabout Model based testing. The third section describes analysisof related works. Section 4 shows the proposed model for Model-Based test path generation. Section 5 presents a case study to demonstrate the use of our methodology with the Login Screen System and Online Banking System. The paperconcludes with section 6.

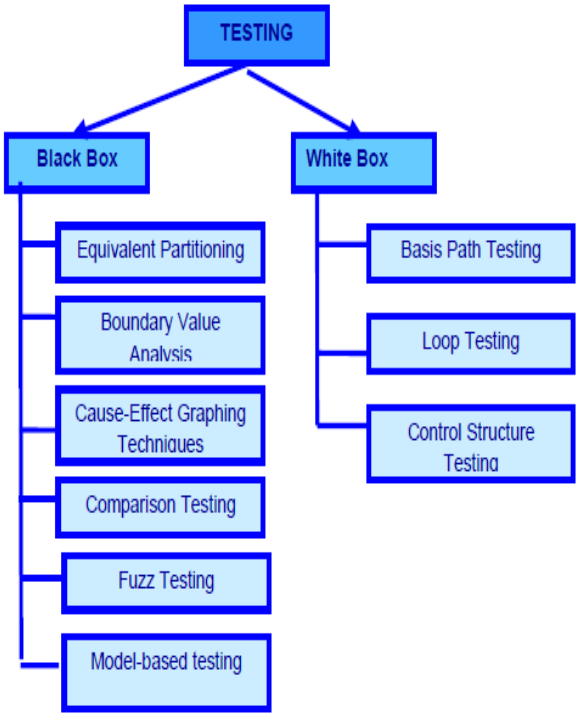

Figure 1. General Classification of Test Techniques

2. Model-Based Testing

There are many testing strategy and among them, some are Model-Based Testing (MBT) that depends on extracting test cases from different models (requirements models, usage models, and models constructed from source code), Specification-based testing (or black-box testing) that depends on requirements models, and Program-based testing (or white-box testing) that uses source code as the underlying model [7].

In recent trend, Model-Based Testing becomes a more popular approach among many researchers. Model-Based Testing (MBT) is a type of testing strategy that depends on extracting test cases from different models (Requirement model, usage model and model constructed from source code) [4]. MBT promises (1)the detection of faults at an early stage in the development cycleand (2)reduces the maintenance effort of test scripts as the test cases human and cost effort are minimized. MBT is seen usually as one form of black-box testing. MBT is suited perfectly being used in system, acceptance, and regression testing [5]. MBT have three main key technologies:

• Notation used for the model

• The test generation Diagram

•The tools that generate supporting infrastructure for the tests [6].

In current, Model-Based test case attracts many researchers by using some data mining concept to produce an automated optimal test case.

Most of researchers have used the following architecture for generating the test paths as shown in Figure 2.

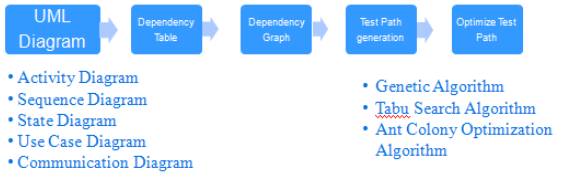

Figure 2. System Flow of Model Based Test Path Generation

One or more UML diagrams are used as an input, construct the dependency table based on these diagrams, and then create dependency graph according to the dependency table. Finally, generate all possible test paths. Some use data mining algorithm to optimize test path [3].

3. Related Works

In [1], Shantghi and Mohan Kumar proposed an approach for generating test cases for object oriented software that uses Tabu Search technique to design and derive test cases. The Tabu search technique was applied to generate optimal test case.

SaurabhSrivastava, Sarvesh Kumar and Ajeet Kumar Verma proposed a technique for optimal path sequencing in basis path testing. This approach is used to generate optimal paths using control flow graph (CFG) and Cyclomatic Complexity for finding the number of feasible paths. This paper applied ant colony optimization algorithm to prevent from selecting infeasible path and also prioritize the feasible paths i.e. which path is to be executed first to optimize time and complexity. With great ease of the ant colony optimization algorithm good results can be got and thus basis path testing can be improved, but there are areas where it can be extended such as if repetition of states or nodes can be controlled which have multiple paths. After execution of this algorithm, it automatically selects the best path sequence [9].

Model Based Test Case Generation Technique Using Genetic Algorithms presented by Mary Sumalatha and Raju based on UML activity diagram. Finding all paths genetic algorithm operations are applied to obtain the best test case [11]. In case study of their paper, the expected results should be accurately. As the above generated new individuals are not part of the generated paths, Path 1 is the best node that is selected. By really Path 4 that is covered the whole of system should be the best path.

Shanthi and Mohan Kumar [2] proposed a new model based approach for automated test cases for UML activity diagrams using genetic algorithm. With the help of ADT test path are generated, by applying the GA most prioritized test case are generated. For each node, the number of incoming nodes and the number of outgoing nodes are calculated and then evaluate fitness values. Next, the initial test data is selected by randomly. Lastly, the Prioritized test path is generated. The generated test cases apply the branch coverage criteria and the CyclomaticComplexity coverage.

4. The Proposed Model

In this section, figure 3 illustrates the flow of the automatic best test path generation by using the activity diagram.

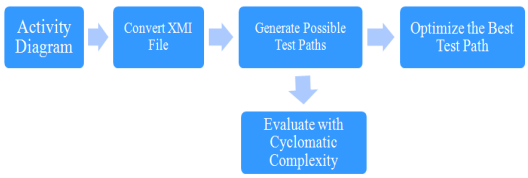

Figure 3. Architecture of the Proposed System

The proposed system uses the activity diagram as an input for the automated algorithm of generating test paths. The Modelio Software has the option of exporting the UML diagram to XMI file. Figure 4 shows the proposed algorithm for automatic test path generation.

```
Input: XMI file for Activity Diagram.
Output: all possible test paths and the best test path.
begin
  k:=0;          j:=0;          countJoinIn:=0;
  countForkOut:=0;countDecision:=0;
  totalDecision:=total decision node of the AD;
  TP_kj++ := source node of InitialEdge.
  myNode= target node of InitialEdge.
  SearchEdge( myNode, InitialEdge);
  OptimalTestPath().
end
```

Figure 4. Algorithm for Automatic Best Test Path Generation

Figure 5 and Figure 6 describe the functions that are used from the proposed algorithm. By using this algorithm, the proposed system generates all possible test paths based on the extracted information from XMI file according to figure 5. And then, the system optimizes the best test path among from these generated test paths by using the *OptimalTestPath* function as shown in figure 6. Finally, the generated test paths validate with Cyclomatic Complexity.

```
SearchEdge(Node s, Edge ee)
 begin
   nodeType := type of s node;
   If (nodeType != FinalNode&&
                    (k<=totalDecision*2))
    For each edge e_i Є E          // i=1,2,…,n
     sNode := source node of edge e_i;
     tNode := target node of edge e_i;
     if (sNode== s )
      if (nodeType==Action ||
                    nodeType==Merge)
          Add s to TP_kj++ ;
      else if (nodeType==Fork)
          countForkOut : = s.getCountNode();
          countForkOut++;
                    s.setCountNode(countForkOut);
          if (countForkOut ==1)
              Add s to TP_kj++ ;
          endif
      else if (nodeType==Join)
          countJoinIn := s.getCountNode();
          countJoinIn++;
          s.setCountNode(countJoinIn);
          if (countJoinIn == s.getCountIn())
              Add s to TP_kj++ ;
          endif
      else
          if(countDecision<=totalDecision*2)
              countDecision++;
               Add s to TP_kj++ ;
          endif
      endif
      SearchEdge(tNode, e_i);
     endif
    endfor
   else
     Add s to TP_kj++ ; k++;
     countDecision := 0;
     Set count for Fork node and Join node with 0;
   endif
 end
```

Figure 5. Algorithm of function *SearchEdge*

```
OptimalTestPath()
begin
 maxControlNode:= total control node of TP_0;
 maxTotalNode  := total node of TP_0 ;
 BestPath  := TP_0 ;
 for  each test path TP_i Є TP       // i=1,2,…n.
 curControlNode := total control node of TP_i ;
 curTotalNode  := total node of TP_i  ;
 if (curControlNode>maxControlNode )
      if (curTotalNode>= maxTotalNode)
 BestPath := TP_i ;
 maxControlNode  := curControlNode ;
 maxTotalNode  := curTotalNode ;
 else
 BestPath := TP_i ;
 maxControlNode  := curControlNode ;
 endif
 endif
 if (curControlNode  == maxControlNode )
 if ( curTotalNode>= maxTotalNode)
 BestPath := TP_i ;
 maxControlNode  := curControlNode ;
 maxTotalNode  := curTotalNode ;
 endif
 endif
 endfor
 Display BestPath as the optimal test path;
end
```

Figure 6. Algorithm of function *OptimalTestPath*

5. Case Study

This section presents the case study of Login Screen System and Online Banking System. Problem statement for Login Screen System: Login screen initially displays a message “Login here using your Username and Password”. The user enters theusername and password and the system checks if the entered username and password are valid or not. If they arevalid it displays the main page if wrongly entered it checks the number of times login has been tried. If thenumber of times the login is less than the number of times feed in the system it displays “Invalid Login PleaseTry again” or it displays “Sorry you have exceeded the allowed number of login attempts”.The activity diagram for the above problem is as shown in Figure 7 which is created by modelio software.

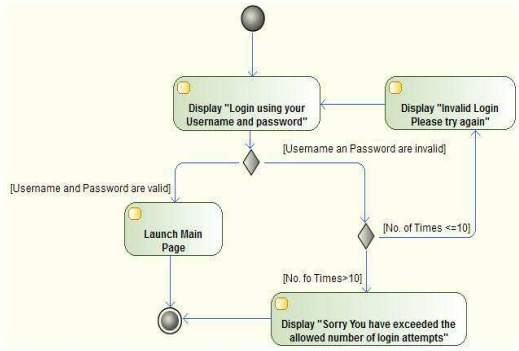

Figure 7. Activity Diagram for Login Screen

In figure 8, the nodes and edges information of the activity diagram is extracted from converted XMI file of the activity diagram.

```
<?xml version="1.0" encoding="UTF-8"?>

<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML" xmi:id="_vwWXgJMjEeOHcPtZRqSzDw"
name="login screen">

 <packagedElement xmi:type="uml:Activity" xmi:id="_vwWXh5MjEeOHcPtZRqSzDw" name="Activity">

  <node xmi:type="uml:InitialNode" xmi:id="_vwWXjJMjEeOHcPtZRqSzDw" name="Initial Node"
outgoing="_vwWXlJMjEeOHcPtZRqSzDw"/>

  <node xmi:type="uml:OpaqueAction" xmi:id="_vwWXjZMjEeOHcPtZRqSzDw" name="Display
&amp;quot;Login using your Username and password&amp;quot;" outgoing="_vwWXlpMjEeOHcPtZRqSzDw"
incoming="_vwWXlJMjEeOHcPtZRqSzDw _vwWXmJMjEeOHcPtZRqSzDw">

   <body></body>

  </node>

  <edge xmi:type="uml:ControlFlow" xmi:id="_vwWXlJMjEeOHcPtZRqSzDw" name="ControlFlow"
source="_vwWXjJMjEeOHcPtZRqSzDw" target="_vwWXjZMjEeOHcPtZRqSzDw">

   <weight xmi:type="uml:LiteralInteger" xmi:id="_vwWXlZMjEeOHcPtZRqSzDw" value="1"/>

  </edge>
```

Figure 8. Converted XMI file from Activity Diagram

Each node has node type, identity of node, node name, identity of incoming edges, and identity of outgoing edges. Each edge has edge type, identity of edge, edge name, identity of source node, and identity of target node. The number of node and edge, and detailed information are shown in Figure 9.

No of Node=8

Id	Name	Incoming	Outgoing
_vwWXjJMjEeOHcPtZRqSzDw	Initial Node		_vwWXlJMjEeOHcPtZRqSzDw
_vwWXjZMjEeOHcPtZRqSzDw	Display "Login using your Usema...	_vwWXlJMjEeOHcPtZRqSzDw_vw...	_vwWXlpMjEeOHcPtZRqSzDw
_vwWXlpMjEeOHcPtZRqSzDw	Display "Invalid Login Please try ag...	_vwWXmJMjEeOHcPtZRqSzDw	_vwWXmJMjEeOHcPtZRqSzDw
_vwWXl5MjEeOHcPtZRqSzDw	Decision-Merge	_vwWXlpMjEeOHcPtZRqSzDw	_vwWXmpMjEeOHcPtZRqSzDw_v...
_vwWXkJMjEeOHcPtZRqSzDw	Decision-Merge1	_vwWXnZMjEeOHcPtZRqSzDw	_vwWXqJMjEeOHcPtZRqSzDw_vw...
_vwWXkZMjEeOHcPtZRqSzDw	Launch Main Page	_vwWXmpMjEeOHcPtZRqSzDw	_vwWXppMjEeOHcPtZRqSzDw
_vwWXqpMjEeOHcPtZRqSzDw	Display "Sorry You have exceeded t...	_vwWXo5MjEeOHcPtZRqSzDw	_vwWXqJMjEeOHcPtZRqSzDw
_vwWXk5MjEeOHcPtZRqSzDw	Activity Final Node	_vwWXppMjEeOHcPtZRqSzDw_vw...	

No of Edge=9

Id	Name	Source Id	Source Name	Target Id	Target Name
_vwWXlJMjEeOHcPtZ...	ControlFlow	_vwWXjJMjEeOHcPtZ...	Initial Node	_vwWXjZMjEeOHcPtZ...	Display "Login using y...
_vwWXlpMjEeOHcPtZ...	ControlFlow	_vwWXjZMjEeOHcPtZ...	Display "Login using y...	_vwWXl5MjEeOHcPtZ...	Decision-Merge
_vwWXmJMjEeOHcPt...	ControlFlow	_vwWXlpMjEeOHcPtZ...	Display "Invalid Login ...	_vwWXjZMjEeOHcPtZ...	Display "Login using y...
_vwWXmpMjEeOHcPt...	ControlFlow	_vwWXl5MjEeOHcPtZ...	Decision-Merge	_vwWXqJMjEeOHcPtZ...	Launch Main Page
_vwWXnZMjEeOHcPtZ...	ControlFlow	_vwWXl5MjEeOHcPtZ...	Decision-Merge	_vwWXkJMjEeOHcPtZ...	Decision-Merge1
_vwWXoJMjEeOHcPtZ...	ControlFlow	_vwWXkJMjEeOHcPtZ...	Decision-Merge1	_vwWXpMjEeOHcPtZ...	Display "Invalid Login ...
_vwWXo5MjEeOHcPtZ...	ControlFlow	_vwWXkJMjEeOHcPtZ...	Decision-Merge1	_vwWXqpMjEeOHcPtZ...	Display "Sorry You hav...
_vwWXppMjEeOHcPtZ...	ControlFlow	_vwWXkZMjEeOHcPtZ...	Launch Main Page	_vwWXk5MjEeOHcPtZ...	Activity Final Node
_vwWXqJMjEeOHcPtZ...	ControlFlow	_vwWXkpMjEeOHcPtZ...	Display "Sorry You hav...	_vwWXk5MjEeOHcPtZ...	Activity Final Node

Figure 9. Extracted the information details of Activity Diagram

Based on these extracted information, all possible test paths are generated directly from XMI file by using the proposed algorithm. These generated test paths are shown in Figure 10.

P1: Initial Node → Display ″Login using your Username and password″ → Username and Password are valid? → Launch Main Page → Activity Final Node

P2: Initial Node → Display ″Login using your Username and password″ → Username and Password are valid? → No. of Times<=10 → Display″Sorry You have exceeded the allowed number of login attempts″ → Activity Final Node

P3: Initial Node → Display ″Login using your Username and password″ → Username and Password are valid? → No. of Times<=10 → Display″Sorry You have exceeded the allowed number of login attempts″ → Display″Invalid Login Please try again″ → Display ″Login using your Username and password″ → Username and Password are valid? → Launch Main Page → Activity Final Node

P4: Initial Node → Display″Login using your Username and password″ → Username and Password are valid? → No. of Times<=10 → Display″Sorry You have exceeded the allowed number of login attempts″ → Display″Invalid Login Please try again″ → Display ″Login using your Username and password″ → Username and Password are valid? → No. of Times<=10 → Display″Sorry You have exceeded the allowed number of login attempts″ → Activity Final Node

Figure 10. All Possible Test Paths

And then the best test path is optimized depend on number of control nodes and number of total nodes of each test path according to the propose algorithm. The best test path is shown in Figure 11.

P4: Initial Node → Display″Login using your Username and password″ → Username and Password are valid? → No. of Times<=10 → Display″Sorry You have exceeded the allowed number of login attempts″ → Display″Invalid Login Please try again″ → Display ″Login using your Username and password″ → Username and Password are valid? → No. of Times<=10 → Display″Sorry You have exceeded the allowed number of login attempts″ → Activity Final Node

Figure 11. The Best Test Path

We can see that the number of predicates are three,edges E are 9 and nodes N are 8, then the will be asfollows:

$$V = E - N + 2 = 9 - 8 + 2 = 3 \quad testpaths$$

Then the upper bound of test paths that ensures the fullactivity path coverage is 4 test paths which were produced bythe proposed model. Since,

Branch coverage<=Cyclomatic complexity<=no. of paths

Therefore, the branch coveragecriteria, and the Cyclomatic complexity coverage are applied by the generated test cases.

The proposed approach is also evaluated by the Activity diagram of Online Banking Systemshown in the appendix A. When the information is extracted from XMI file, the total node of the activity diagram is 51 nodes and the total edge is 59 edges. Finally, all possible test paths that total test paths are 31 test paths; and the best test path are generated.

The proposed system generates all test paths directly from the input file without redundant test paths. Compare the system of Suppandeep Sandhu and Amardeep Singh, their system generate all possible test paths via the final state transition table that created from the input file for tax calculator activity diagram [10]. The proposed system spends less nearly threefold in execution time than the system of [10] for the same input activity diagram.

As the proposed approach can generate all possible test paths and the best test path for not only small system but also large complex system, the proposed approach is efficient and effective for any system.However, in concurrent activities, the proposed system also eliminates the redundant activities from each test path.

6. Conclusion

Quality is the main focus of any software engineering project.Software testing is an activity that is performed for evaluating software quality and also for improving it.Good software testers cannot avoid models. MBT has emerged as a useful and efficient testing method for realizing adequate test coverage of systems.The proposed system based on MBT develops more accurate all possible test paths due to generate directly from XMI file instead of using the dependency table and dependency graph. The generated test paths are validated with Cyclomatic Complexity. The proposed system saves time in choosing the best test path because other optimization algorithms spend time to calculate weight but the proposed approach does not need to evaluate weight. The proposed system is more efficient and effective in the software development, because the human effort in finding bugs and errors,and the steps at the evolution of generated test paths are reduced.

References

[1] A.V.K. Shanthi, G. MohanKumar, “A Novel Approach for Automated Test Path Generation using TABU Search Algorithm”, International Journal of Computer Applications (0975 – 888)Volume 48– No.13, June 2012.

[2] A.V.K. Shanthi, G. MohanKumar, “A Heuristic Technique for Automated Test Cases Generation from UML Activity Diagram”, Journal of Computer Science and Applications. Volume 4, Number 2 (2012), pp. 75-86.

[3] Aye Aye Kyaw; Myat Myat Min. “Model-Based Automatic Optimal Test Path Generation via Search Optimization Techniques: A Critical Review”. In Proceedings of the 12th International Conference on Computer Applications 2014.

[4] Aye Aye Kyaw and Myat Myat Min, "An Efficient Approach for Model Based Test Path Generation," International Journal of Information and Education Technology vol. 5, no. 10, pp. 763-767, 2015.

[5] “Certified Tester Foundation Level Syllabus”, Released Version 2011, International Software Testing Qualifications Board.

[6] Chanda Chouhan Vivek Shrivastava Parminder S Sodhi, “Test Case Generation based on Activity Diagram for Mobile Application”, International Journal of Computer Applications (0975 – 8887)Volume 57– No.23, November 2012.

[7] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba, “A Proposed Test Case Generation Technique Based on Activity Diagrams”, International Journal of Engineering & Technology IJET-IJENS Vol: 11 No: 03.

[8] Santosh Kumar Swain; Durga Prasad Mohapatra. “Test Case Generation from Behavioral UML Models”, International Journal of Computer Applications (0975 – 8887) Volume 6– No.8, September 2010.

[9] Saurabh Srivastava, Sarvesh Kumar and Ajeet Kumar Verma, “OPTIMAL PATH SEQUENCING IN BASIS PATH TESTING”, International Journal of Advanced Computational Engineering and Networking, ISSN (PRINT): 2320-2106, Volume – 1, Issue – 1, 2013.

[10]Suppandeep Sandhu, Amardeep Singh, “A Systematic Approach for Software Test Cases Generation using Gray Box Testing with UML Activity Diagrams” , IJCST Vol. 2, Iss ue 4, Oct . - Dec. 2011.

[11] V.Mary Sumalatha and Dr G.S.V.P.Raju, “An Model Based Test Case Generation Technique Using Genetic Algorithms”, The International Journal of Computer Science & Applications (TIJCSA) Volume 1, No. 9, November 2012 ISSN – 2278-1080.

APPENDIX A

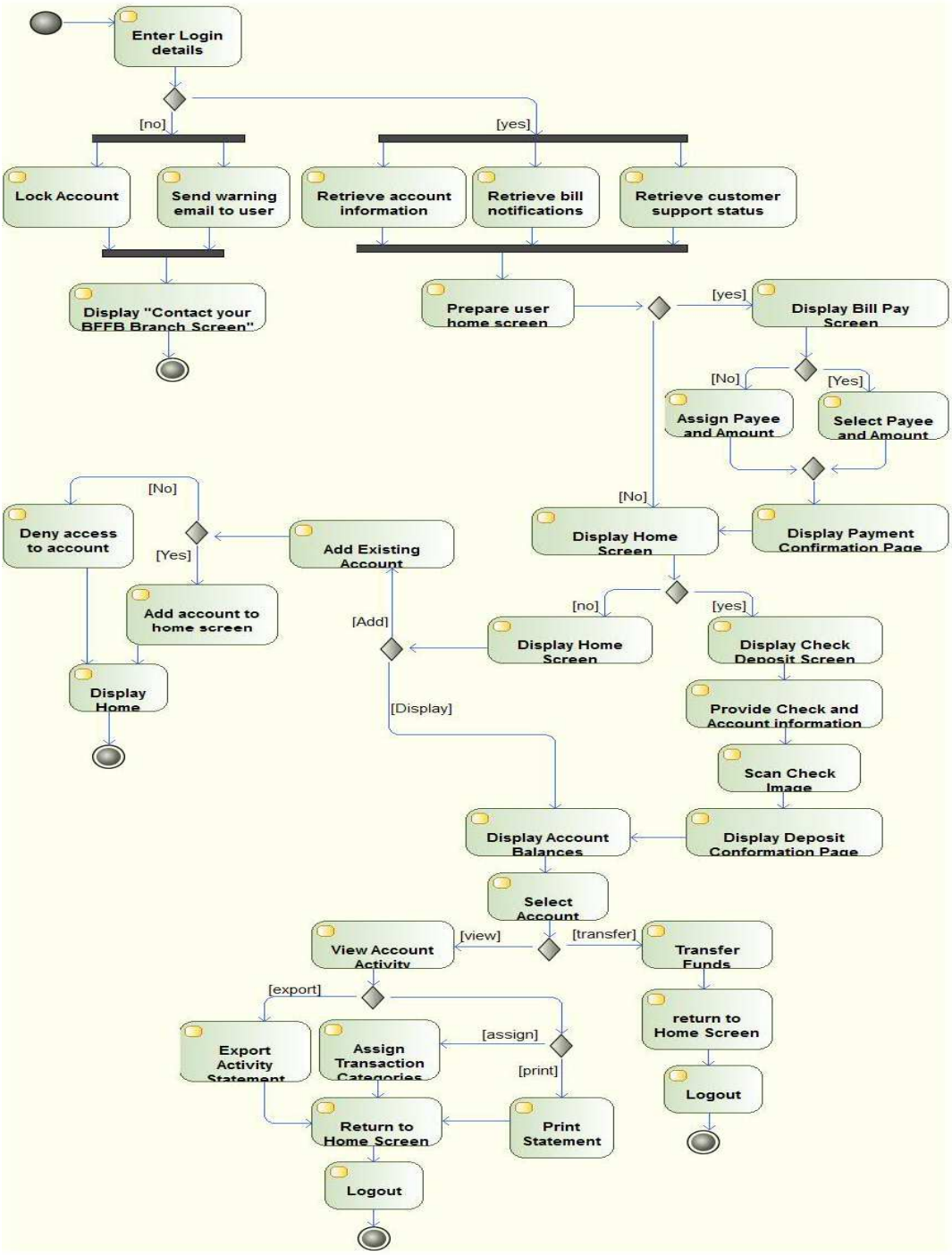

Figure 12. Activity Diagram for Online Banking System