International Journal of Engineering & Technology, 7 (4.31) (2018) 58-63

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

An Automated Test Case Generating Tool Using UML Activity Diagram

Md Abdul Monim[1], Rozi Nor Haizan Nor[2*]

[1,2] Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Selangor, Malaysia
**Corresponding author E-mail: rozinor@upm.edu.my*

Abstract

Software or application testing is a process of executing a program with the goal of finding defect to make better system. In software testing phase, writing test cases is one of the important activities. Manually writing test cases approach is lengthy of time period and need more effort to accomplish the process. On the other hand, automated test case generation technique is the way to solve this issue and model-based test case generation approach would be the appropriate for this automation process. Usually, a model is required in model-based testing approach to generate the test cases and UML activity diagram is the model in our research. In this paper, we explained our proposed model-based test case generating approach and we successfully developed a tool based on our proposed technique that could generate test cases automatically using UML activity diagram as an input. Finally, we conducted an experiment on a real life simple application of a system using the tool and successfully able to show that our tool can produce same test cases as manually writing test cases of the same system but this tool can save a lot of time and effort as well.

Keywords: *Model-based testing (MBT); Model transformation; Test automation; Test case generation; UML activity diagram*.

1. Introduction

Today we are living in the software economy era, where every business is now software related. While this is exciting, but developing software is also have an array of strenuous challenges. Software and Applications are developed to enhancing our daily life. Software is called good when its performance and efficiency shows some good sign. Making a good quality software is the main goal for all the developer and they follow a standard rule to gain that quality. A software system or a web application had to go through into a development life cycle and testing is an important phase of this software/application development life cycle.
Software testing can be done by manually or automatically. In manual testing, the testing requires to do all the process by manually includes input, analysis, writing and managing the test cases. Manual testing deals with human interaction with all the process from beginning to end of the process and it is a time-consuming process. A human can get tired of doing all the process continuously. On the other hand, in automated testing, most of the testing processes are automated. Generating test cases, executing the test cases and producing the test result are done by automatically. Though, one of the software testing fundamental is hundred percent automation is not possible in the field of software testing. Some task still needs the intervention of human. Automated testing process needs some of the tools to support that automation and there are a huge amount of tools are available in the market. Different kind of tools was developed for different types of language and methods. The scientists and academies have focused more attention on this automated testing field to developing different techniques, methods, approaches, and tools.
If the software testing stage can be finished early in software development life cycle then the total development process will be shortened and the software product is possible to deliver early. So, the main concern is reducing the testing time by applying any technique or approach and test case generation is mostly related part to consume the length. Manual test case writing approach is lengthy of time period and need more effort to accomplish the process. Manually have to write all test cases from the requirement and then execute the test cases are also done by manually. Writing test cases from the requirement is a very long process, boring and error-prone. Hence, automated test case generation is the way to solve this issue.
In this paper we adopted an approach for test case generation technique. The model-based testing (MBT) approach was adopted for test case generation in this research. We proposed a model to automate the test case generating techniques for generate test case and we have developed a tool that can generate test cases automatically using UML activity diagram based on our proposed model. In case study we have applied our developed tool in a system and presented an overall process with result and discussion.

2. Related Work

A lot of previous research has done based on model-based test case generation from different UML diagram using different techniques.
Vikas Suhag and Rajesh Bhatia [1] proposed a model-based testing technique based on UML Web diagram and UML Sequence diagram for a web application. They used UML Web diagram that has the element like client page, form, frameset and links to model the web application with the help of association, generalization and dependency between web pages. They used sequence diagram to illustrate the change of the state of that web application in response to each operation specified in web diagram during the web page design. For the model interchange XML has used for this research.

Sagarkumar et al. [2] describes an approach for test case generation from combination of UML activity diagram and use case diagram. At first, activity diagram are converted into activity graph. Then extracting concurrent control flow path (CCFP) from activity graph. For CCFP they used depth-first search (DFS) algorithm. Finally they use a combinational approach using use-case and activity diagram to generate test case. For exchanging UML model they used XML.

Syed Asad et al. [3] shows in their paper, an automated model-based test case generation process using combination of UML sequence diagram and UML class diagram. They used formal modeling approach and for constructing those models Visual Paradigm was used. C# programming language was used for developing their system and XML as a model interchange.

Anbunathan R & Anirban Basu [4] proposed a method to generate test cases from UML activity diagram using genetic algorithm. In their method, Activity Diagram was created to capture input scenarios. Activity diagram was converted to XMI then it parsed to extract model information. SAX parser was used for parsing technique. A control flow graph (CFG) is derived from edges by sorting the edges using depth-first search (DFS) algorithm. A recursive algorithm is developed to obtain test path to generate test cases from CFG. They were generated executable test case so that it could be execute into test scripts, they used Robotium API database using Genetic Algorithm.

Supriya S. Patil & Pramod A Jadhav [5] proposed a functional test case generation based on model-driven testing using UML activity diagram and Finite State Machine (FSM). They have done a research to generate test case using individual approach where path was calculated using activity diagram and finite state machine separately and again another path was calculated using combine of activity diagram and finite state machine. Then they removed redundancy before generating test cases using combined result.

Fernando & Glaucia [6] presents an automatic approach named EasyTest to generate test cases from UML activity diagrams using gray-box techniques. The EasyTest approach comprises three phases, 1) importing activity diagrams in XMI; 2) test cases generation; and 3) applying test cases. The study used activity dependency graph and activity dependency tree for sequencing the test path. They developed a Java tool called "EasyTest tool" that provides automatic support for the Phases 2 and 3.

2.1. Comparison with Some Existing Tools

There were several model-based testing tools available in the market that can generate test cases using different techniques and different types of input format. Some of them are discussed below,

1. **Tool name:** BPN-Xchange
Input format: BPMN, UML
Type: Commercial
Description: The tool BPM-Xchange can create test cases from business process modeling (BPM) using criteria (statement, branch, path, and condition).This tool can able to import models from different types of modeling tools and able to export test cases in Excel file. It also supports HP Quality Center for test case management.

2. **Tool name:** GraphWalker
Input format: FSM
Type: Open source
Description: This tool generates test cases from finite-state machines (FSM) using A* search algorithm or random search algorithm with a limitation for various coverage criteria like state, edge, requirement. The approach for generating test cases is also model-based testing (MBT).

3. **Tool name**: MISTA
Input format: Petri net
Type: Academic
Description: MISTA is a test case generating tool based on high-level Petri nets (a mathematical modeling language). It can generate executable test code for different types of platform like JUnit, NUnit, Selenium using mapping algorithm. Usually MISTA is used for functional testing or security testing.

3. Model-Based Testing Approach

"Model-based testing (MBT) is a testing technique where the runtime behavior of an implementation under test is checked against predictions made by a formal specification or model" [7].

Model-based testing actually is a black box testing approach which the internal design of the system is not taken into account during the test case generation. The process of MBT starts with constructing functional test models based on the software or system requirements then this model is used for generating the test cases. The final test execution can be done either manual or automated [8].

Model-based testing (MBT) has two approaches, one is online MBT another is offline MBT. Online MBT is an approach where tests are generated dynamically during execution. That means test case generation and execution in a same motion. On the other hand, offline MBT is an approach where test cases are generated from a test model and execute it later on. It generating finite set of test cases and execute it later and it allows automatically test execution in third party test execution environment. Our proposed model followed offline MBT approach. For test case generation it is based on the information provided by the test models so that the models should need to include the system behavior that the tester wishes to test. Modeling of a system is a very suitable way to represent any system. Models can be as simple as a graph, flowchart, or diagram. A model of a software or system is a portrait of its behavior where the behavior can be described in terms of the input sequences that accepted by the system, set of actions, conditions and the flow of data through the system's module.

In modeling there have also two types; informal model and formal model. Informal model can be drawing directly from requirements and usually no need to follow any modeling rules. All the notation of models used for testing does not have to comply with any modeling rules. The only rule is followed during creation of these model is that the test model is created only for understand or read able for that person or party who involved in testing. This type of model can be used for test case generation but the tester should be able to understand all the information that the model keep inside. These models can be created using any modeling tools or just hand drawing. While formal model is a model that should be designed by following some specific modeling rules. All the notation of model have some formal rules [9]. As an example UML is a formal modeling language. The main advantage of using formal model is the possibility of generating test cases automatically. A tool is needed to interpret with that model to read all the formal notation of that test model. Many open source and commercial modeling tools is available in the market for developing UML.

Moreover, the model-based testing process can be divided into three phases (see figure 1), those are i) Modeling, ii) Test case generation and iii) Test execution (This research was not involved to test execution). The modeling deals with the designing of system behavior, where the test case generator generates test cases from these models. Lastly, the test executor conducts the test and usually it needs a third party test environment.

4. UML Activity Diagram

One of the most popular formal modeling language is Unified Modeling Language (UML). UML diagrams can be fragmented into several interconnected interaction diagrams. Activity diagram is one of the most used UML behavioral diagram. An activity diagram is suitable for generating the test cases to decide the different components.

Activity diagram allows modeling a process as an activity that consists of a collection of nodes connected by edges. An activity can be attached to any modeling element for the purpose of modeling its behavior. The element provides the context for the activity, and the activity may refer to features of its context [2].

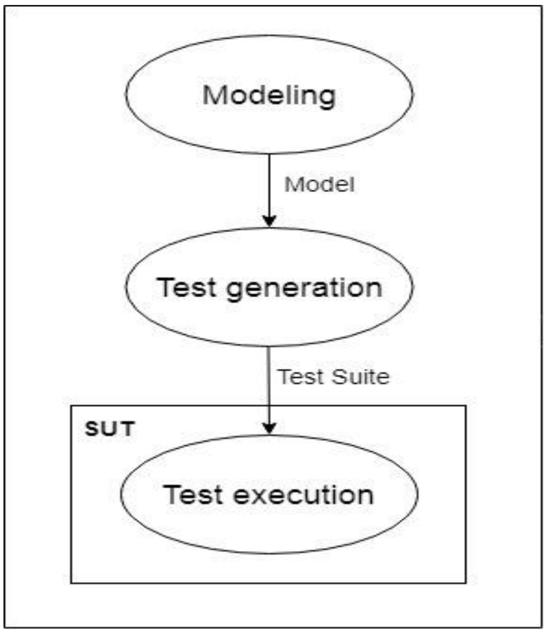


Fig. 1: Model-based testing approach.

In activity diagram have two major elements: nodes and edges. Each node is an action node, control node, or can be object node. Nodes in activity diagram are connected by edges which include control flow edges or object flow edges.
More elements of an activity diagram, elaborated in [10].
1) **Initial node:** The filled in circle is the starting point of the diagram.
2) **Activity final node:** The filled circle with a border is the ending point. An activity diagram can have zero one or more activity final nodes.
3) **Activity:** The rounded rectangles represent activities that occur.
4) **Edge:** The arrows on the diagram.
5) **Decision node:** A diamond with one flow entering and several leaving.
6) **Condition:** Text such as correct or incorrect, yes or no, pass or fail on a flow, defining a guard which must evaluate to true in order to traverse the node.
7) **Fork node:** A black bar with one flow going into it and several leaving it. This denotes the beginning of parallel activity.
8) **Join node:** A black bar with several flows entering it and one leaving it. All flows going into the join must reach it before processing may continue. This denotes the end of parallel processing.
9) **Merge node:** A diamond with several flows entering and one leaving. The implication is that one or more incoming flows must reach this point until processing continues, based on any guards on the outgoing flow.
10) **Flow final node:** The circle with the X through it. This indicates that the process stops at this point.

5. Proposed Model

A model has been proposed for the overall development process to generate test cases from UML activity diagram. The proposed model has some process starts from developing a model (UML activity diagram) for system under test (SUT), this model should be a formal model. A formal model is a model that should be designed by following some specific modeling rules. This formal model must developed by a modeling software or tool that can interpret with that model to read all formal notation and transform the model as an intermediate modeling form. The next process is model reading method, here have a predefined method (will be discussed at the next section) for reading the model so that the programming language can read or parse the model to take the necessary attributes. After that there have a test case generation technique (the technique will be elaborated at next section), this activity follows the given technique for generating test cases from the model. The final activity is test cases that come through a test case generator. The test cases will be generated at this step as an output. The proposed model is given below,

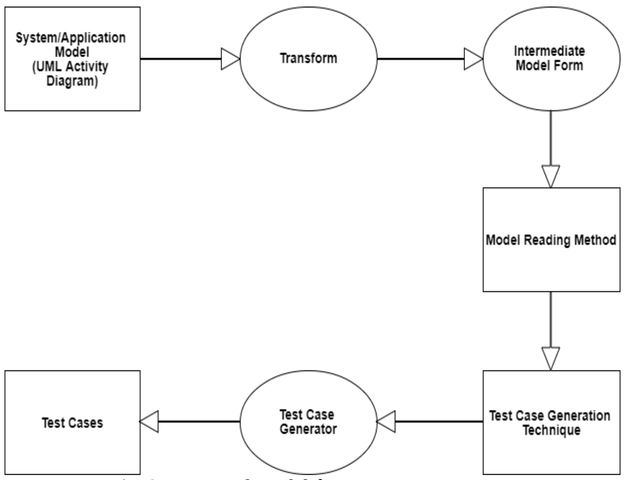


Fig. 2: Proposed model for test case generation.

6. Implementation

6.1. Modeling for System under Test (SUT)

In model-based testing (MBT) approach required a model for generating test cases and these models developed from software requirements and design specifications. The important thing is, most of the requirements and high-level design of software are documented in the form of model and these models can be used for generating test cases by automatically. All the models represent the externally visible behavior of the software or system. The notation of the models can be graphical, textual or mixed, where the notation can be varied.
So, in this phase required a model for the system under test (SUT) and this research already selected UML activity diagram as a model. We have mentioned earlier that in model-based testing approach usually use two different types of model, i) Formal model, ii) Informal model. This project used formal modeling approach. A formal model is a model that describes all the functionality of the SUT by following some formal modeling rules. So, here UML activity diagram is our formal model. And any high level programming language need a read able description of the functionality of the SUT, it usually required a software or tool for developing or creating that formal model. Now need to develop/create a UML activity diagram as a model for the system under test using any modeling tools that can develop the diagram then save/export the diagram in (.uml) extension format as an intermediate form of model. Many open source and commercial modeling tools are available in the market for developing UML diagram.

6.2. Test Case Generation

In total test case generation process, the task has divided into two sections 1) Model conversion, 2) Generating test cases from the model. And each section has two steps.
First section is model conversion; this section has two steps to preparing the activity diagram for generating test cases.

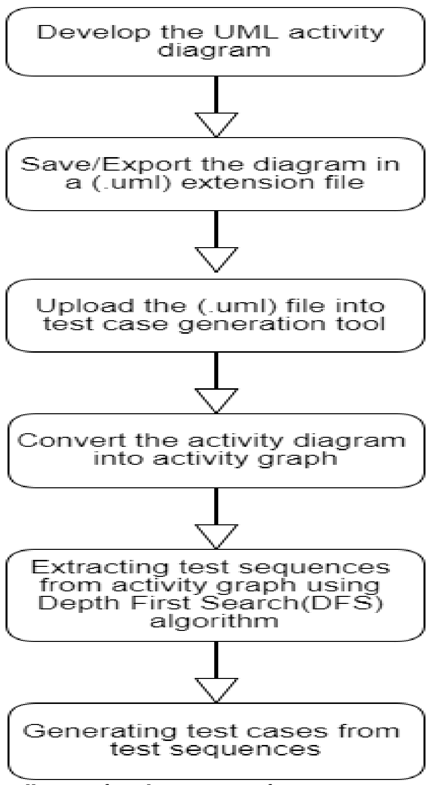
1) Model Conversion
Step 1: Read the model (UML activity diagram) that already developed:
After successfully read the UML activity diagram then it will take all the necessary attribute/notation for the further step. For reading the diagram (.uml) file, here applied “Java DOM Parser” to make a read able Java code for further use.
Step 2: Convert the model (UML activity diagram) into an activity graph based on given technique:
To generate test cases from an activity diagram, first need to convert the activity diagram (.uml) file to an intermediate diagram known as activity graph. Activity graph is a directed graph that contains start node, end node, and the edges. The edges represent the flow in the activity diagram. In this activity graph, the vertices are the action nodes and the dependency edges are the edges between the activity action nodes. The fork, join, and merge nodes are ignored while constructing the activity graph but the flow of edges kept remain same. This activity graph conversion technique has been proposed by Jain et al. [2]. They proposed a set of rules for mapping an activity diagram into activity graph.
This step will execute when the diagram (.uml) file already uploaded into the developed tool.
2) Generating test cases from the model
In generating test cases from the model section, there are two steps have to follow to generate test cases from activity graph:
Step 1: Producing test sequences from activity graph:
To producing test sequences the set of basis paths are obtained from the activity graph by applying depth-first search (DFS) algorithm. A basis path is a path in which every loop is executed zero or one times. This ensures that all nodes in the activity graph are traversed at least one or twice to ensure that both true and false values of the loop are executed at least once. To derive basis path, depth-first search is used. The algorithm used to generate basis path, adopted from [2]:

```
Input: Activity graph
Output: Test Sequence
Let TS be Test Sequence.
Let PD be Parameterized Dependencies.
Step 1: Traverse the activity graph using Depth    First Search.
Step 2: If loop present
Step 3: Make one iteration.
Step 4: End if.
Step 5: for each pair (A, B) of instantiated sequence in PD.
Step 6: TS= Merge (A, B)
Step 7: Store all the Test Sequence TS.
Step 8: End.
```

Test sequences will be printed based on basis path.
Step 2: Generating test cases according to test sequences:
After successfully producing the test sequences from activity graph, the tool is ready to generate the test cases. Once the user executes “Generate test cases” the tool will generate test cases according to test sequences in a given specific format. The test cases will be saved in a text (.txt) file after completion of test case generation. In test cases, there have a test case ID that indicates the unique number of test path and all the test steps (activity) that indicates the test sequences of this test path.

7. Case Study

In this section we used our developed tool to make a comparison with manually created test cases and performed an experiment on a real life simple application of a system using that tool. We have considered for our experiment in an automated teller machine (ATM) cash withdraw system for the case study. An activity diagram of ATM cash withdraw system have used in our developed tool to generate the test cases. This formal UML activity diagram of ATM cash withdraw system is developed by Eclipse modeling tools with Papyrus plug-in. In this activity diagram has total seven activity node, three decision node, one fork node, one join node, one initial node and one activity final node.


Develop the UML activity diagram
Save/Export the diagram in a (.uml) extension file
Upload the (.uml) file into test case generation tool
Convert the activity diagram into activity graph
Extracting test sequences from activity graph using Depth First Search(DFS) algorithm
Generating test cases from test sequences
Fig. 3: Overall steps of implementation of test case generation process.

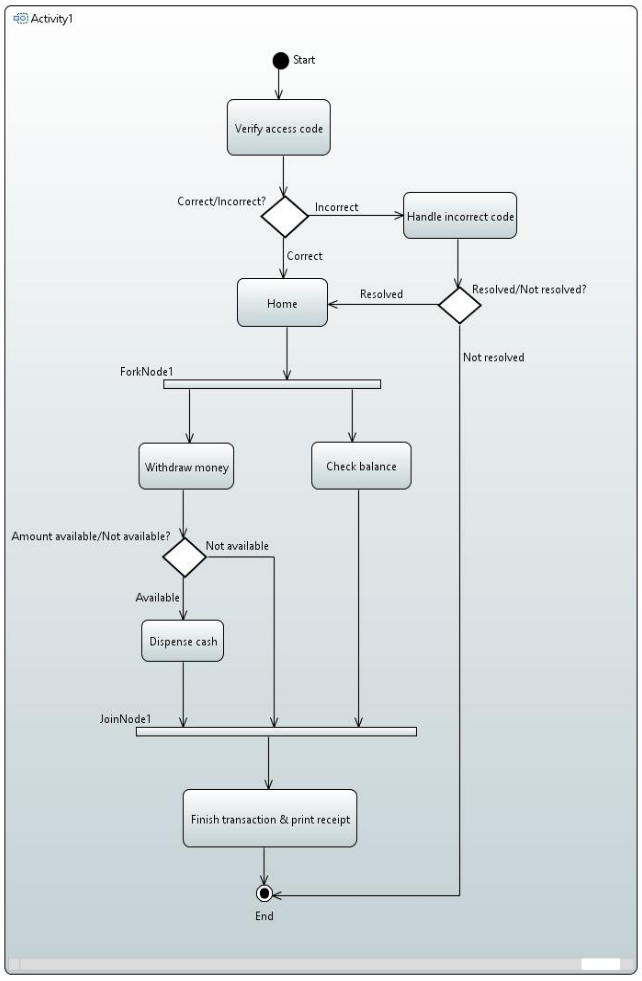


Fig. 4: UML activity diagram of ATM cash withdraw system.

At first we have used manual process to create the test cases using this ATM cash withdraw system to compare with the result that was generated using our developed automated test case generating tool. So when we use manually test case generation technique using our proposed technique then the activity diagram need to be converted into an activity graph. The activity graph is given below,

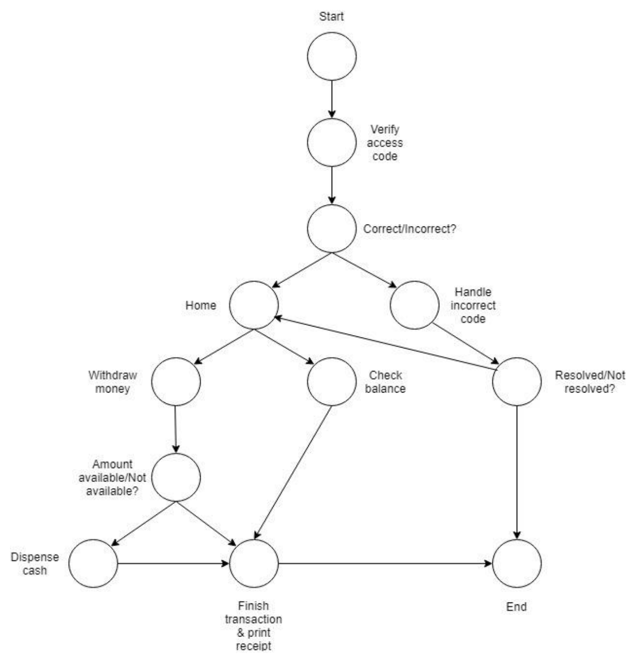


Fig. 5: Activity graph of ATM cash withdraw system.

As we described before, in this activity graph, the vertices are the action nodes and the dependency edges are the edges between the activity action nodes. The fork, join, and merge nodes are ignored while constructing the activity graph but the flow of edges kept remain same.

After we get the activity graph then we need to produce test sequence from this activity graph using depth-first search (DFS) algorithm. Test sequences is presented based on basis path. Produced test sequences are presents below,

Basis path: 1

Test sequences: Start - Verify access code - Correct/Incorrect? - Handle incorrect code - Resolved/Not resolved? - Home - Withdraw money - Amount available/Not available? - Dispense cash - Finish transaction & print receipt - End

Basis path: 2

Test sequences: Start - Verify access code - Correct/Incorrect? - Handle incorrect code - Resolved/Not resolved? - Home - Withdraw money - Amount available/Not available? - Finish transaction & print receipt - End

Basis path: 3

Test sequences: Start - Verify access code - Correct/Incorrect? - Handle incorrect code - Resolved/Not resolved? - Home - Check balance - Finish transaction & print receipt - End

Basis path: 4

Test sequences: Start - Verify access code - Correct/Incorrect? - Handle incorrect code - Resolved/Not resolved? - End

Basis path: 5

Test sequences: Start - Verify access code - Correct/Incorrect? - Home - Withdraw money - Amount available/Not available? - Dispense cash - Finish transaction & print receipt - End

Basis path: 6

Test sequences: Start - Verify access code - Correct/Incorrect? - Home - Withdraw money - Amount available/Not available? - Finish transaction & print receipt - End

Basis path: 7

Test sequences: Start - Verify access code - Correct/Incorrect? - Home - Check balance - Finish transaction & print receipt - End

Now we used our developed automated test case generating tool to generate test cases using same diagram that we have constructed using Eclipse modeling tools. When the activity diagram was uploaded into the tool, the tool has ready to generate test cases, just need one click on the button "Generate test cases". We used Java programming language to develop the tool. Screenshots of the interface of the tool are given below,

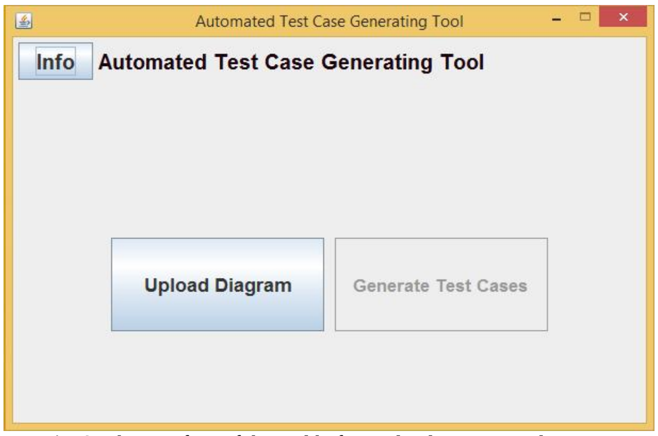

Automated Test Case Generating Tool
Info
Automated Test Case Generating Tool
Upload Diagram
Generate Test Cases

Fig. 6: The interface of the tool before upload an activity diagram.

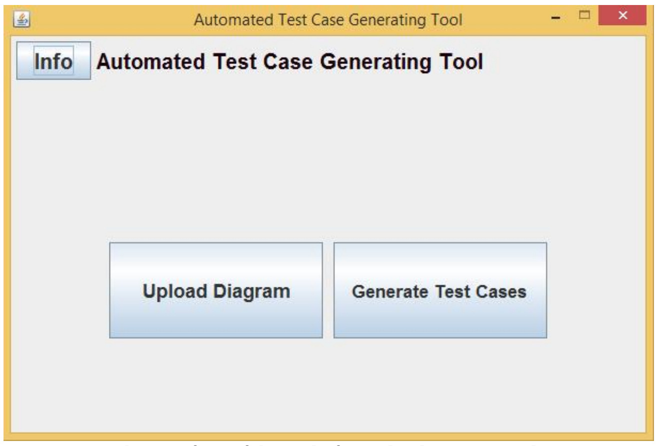
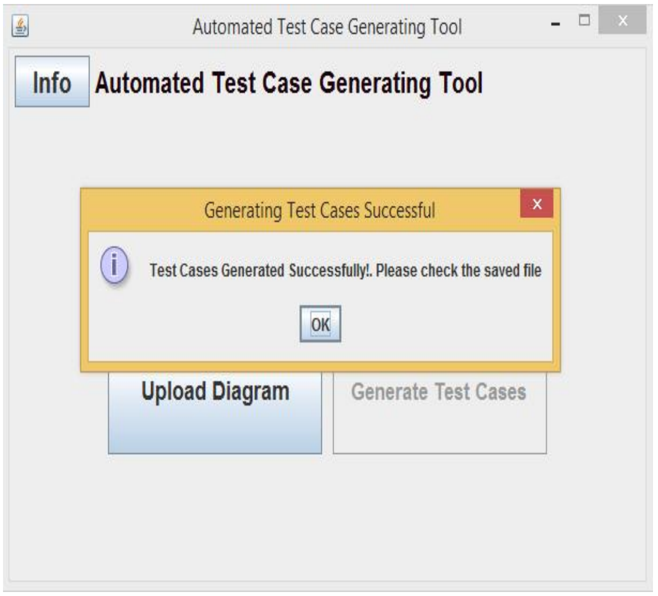

Automated Test Case Generating Tool
Info
Automated Test Case Generating Tool
Upload Diagram
Generate Test Cases

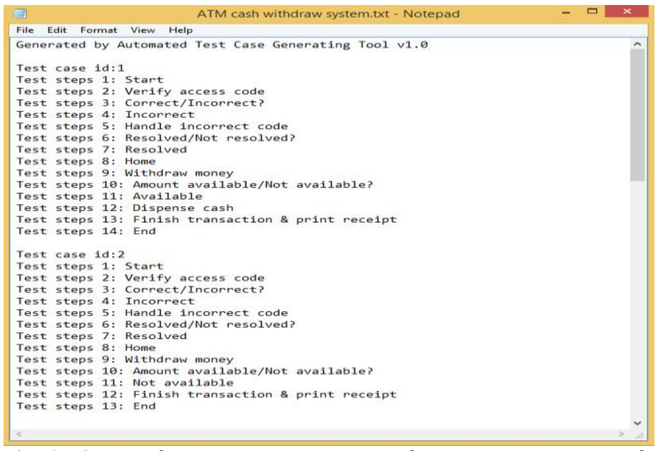
Fig. 7: The interface of the tool after upload an activity diagram.


Automated Test Case Generating Tool
Info
Automated Test Case Generating Tool
Generating Test Cases Successful
OK
Upload Diagram
Generate Test Cases

Test Cases Generated Successfully!. Please check the saved file

Fig. 8: The interface of the tool after click the button "Generate test cases".

After generate test cases successfully the tool saved all the generated test cases in a text (.txt) file. The result of automatically generated test cases of ATM cash withdraw system are provided below as a screenshot of (.txt) file.

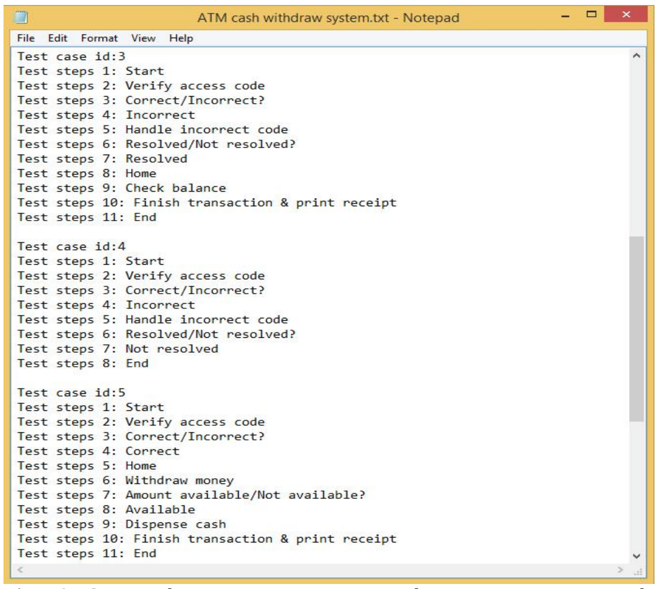


```
Generated by Automated Test Case Generating Tool v1.0

Test case id:1
Test steps 1: Start
Test steps 2: Verify access code
Test steps 3: Correct/Incorrect?
Test steps 4: Incorrect
Test steps 5: Handle incorrect code
Test steps 6: Resolved/Not resolved?
Test steps 7: Resolved
Test steps 8: Home
Test steps 9: Withdraw money
Test steps 10: Amount available/Not available?
Test steps 11: Available
Test steps 12: Dispense cash
Test steps 13: Finish transaction & print receipt
Test steps 14: End

Test case id:2
Test steps 1: Start
Test steps 2: Verify access code
Test steps 3: Correct/Incorrect?
Test steps 4: Incorrect
Test steps 5: Handle incorrect code
Test steps 6: Resolved/Not resolved?
Test steps 7: Resolved
Test steps 8: Home
Test steps 9: Withdraw money
Test steps 10: Amount available/Not available?
Test steps 11: Not available
Test steps 12: Finish transaction & print receipt
Test steps 13: End
```

Fig. 9: Generated test cases using automated test case generating tool screenshot 1/3.

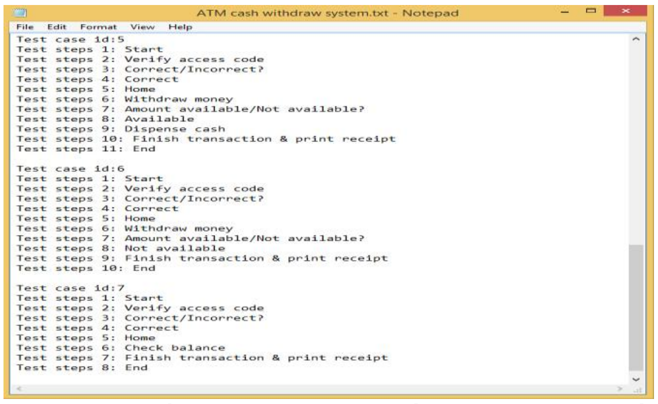


```
Test case id:3
Test steps 1: Start
Test steps 2: Verify access code
Test steps 3: Correct/Incorrect?
Test steps 4: Incorrect
Test steps 5: Handle incorrect code
Test steps 6: Resolved/Not resolved?
Test steps 7: Resolved
Test steps 8: Home
Test steps 9: Check balance
Test steps 10: Finish transaction & print receipt
Test steps 11: End

Test case id:4
Test steps 1: Start
Test steps 2: Verify access code
Test steps 3: Correct/Incorrect?
Test steps 4: Incorrect
Test steps 5: Handle incorrect code
Test steps 6: Resolved/Not resolved?
Test steps 7: Not resolved
Test steps 8: End

Test case id:5
Test steps 1: Start
Test steps 2: Verify access code
Test steps 3: Correct/Incorrect?
Test steps 4: Correct
Test steps 5: Home
Test steps 6: Withdraw money
Test steps 7: Amount available/Not available?
Test steps 8: Available
Test steps 9: Dispense cash
Test steps 10: Finish transaction & print receipt
Test steps 11: End
```

Fig. 10: Generated test cases using automated test case generating tool screenshot 2/3.



```
Test case id:5
Test steps 1: Start
Test steps 2: Verify access code
Test steps 3: Correct/Incorrect?
Test steps 4: Correct
Test steps 5: Home
Test steps 6: Withdraw money
Test steps 7: Amount available/Not available?
Test steps 8: Available
Test steps 9: Dispense cash
Test steps 10: Finish transaction & print receipt
Test steps 11: End

Test case id:6
Test steps 1: Start
Test steps 2: Verify access code
Test steps 3: Correct/Incorrect?
Test steps 4: Correct
Test steps 5: Home
Test steps 6: Withdraw money
Test steps 7: Amount available/Not available?
Test steps 8: Not available
Test steps 9: Finish transaction & print receipt
Test steps 10: End

Test case id:7
Test steps 1: Start
Test steps 2: Verify access code
Test steps 3: Correct/Incorrect?
Test steps 4: Correct
Test steps 5: Home
Test steps 6: Check balance
Test steps 7: Finish transaction & print receipt
Test steps 8: End
```

Fig. 11: Generated test cases using automated test case generating tool screenshot 3/3.

Actually it's a single text file, but due to long output we had to take screenshot three times to provide the full result. In test cases, there have a test case ID that indicates the unique number of test path and all the test steps (activity) that indicates the test sequences of this test path. If we see the activity diagram of ATM cash withdraw system there have some condition between decision node and activity node. When we generate the test cases from activity diagram, we put that condition massage too inside the test steps for better understand. So, the tester can easily get which activity flow is going through after which activity.

8. Conclusion

This paper presented automated test case generation using UML activity diagram. Model-based testing (MBT) approach is used for the whole process. We have been proposed a model to extract, utilize, and preparing the data from diagram to test case then we developed an automated test case generating tool based on our techniques. In the case study we took a real life system for comparison with our developed automated test case generating tool and manually test case generating techniques. We successfully able to show the result that indicates there is no difference between manually writing test cases and automatically generated test cases. But in manual process it is very time consuming, labor intensive task and may have some error prone during writing the test cases. On the other hand, automatically generated test cases can save lots of time, reduce human intervention and error free output. Our developed tool can able to read only UML file that extension is (.uml). The tool is unable to read other extension like (.xml) or (.xmi). In future, we also have intension to make some upgrade in our tool that could be apply test case minimization techniques and priority based test case generation techniques.

Acknowledgement

This research article is funded by MOSTI research grant, project no: 08-01-15-1725FR.
I would like to giving thank to MOSTI for financial support to submit and present this research article at the conference.

References

[1] Vikas suhag and Rajesh Bhatia, "Model based Test Cases Generation for Web Applications," *International Journal of Computer Applications (0975 – 8887)* Volume 92 – No.3, April 2014.
[2] Sagarkumar P. Jain, Khushboo S. Lalwani, Nikita K. Mahajan, and Bhagyashree J. Gadekar Automatic, "Test Case Generation Using Uml Models," *International Journal of Advanced Computational Engineering and Networking*, Volume-2, Issue-6, June-2014.
[3] Syed Asad Ali Shah, Raja Khaim Shahzad, Syed Shafique Ali Bukhari and Mamoona Humayun, "Automated Test Case Generation Using UML Class & Sequence Diagram," *British Journal of Applied Science & Technology*, Article no. BJAST. 24860, 15(3): 1-12, 2016.
[4] Anbunathan R and Anirban Basu, "Executable Test Generation from UML Activity Diagram Using Genetic Algorithm," IRACST - *International Journal of Computer Science and Information Technology & Security (IJCSITS)*, Vol.7, No.3, May-June 2017.
[5] Supriya S. Patil and Pramod A Jadhav, "Functional Test Case Generation based on Model Driven Testing using FSM and UML Activity Diagram," *International Journal of Advanced Research in Computer Science*, Volume 8, No. 5, May-June 2017.
[6] Fernando Augusto Diniz Teixeira and Glaucia Bragae Silva, "EasyTest: An approach for automatic test cases generation from UML Activity Diagrams," in *Information Technology: New Generations (ITNG 2017), 14th International Conference on ITNG, 2017.*
[7] Hitesh Kumar Sharma, Sanjeev Kumar Singh and Prashant Ahlawat, "Model-Based Testing: The New Revolution in Software Testing," *Database Systems Journal*, vol. 5(1), pages 26-31, May 2014.
[8] Ikya Jupudy, Neha Saraf, and Prof. Manjula R, "Comparative Analysis of Model Based and Formal Based Software Testing Methods," *International Journal of Advanced Research in Computer Science and Software Engineering*, Volume 6, Issue 3, March 2016.
[9] Elise Greveraars, Atos Origin vision on MBT, "Model Based Testing (White paper)."
[10] V. Mary Sumalatha and G.S.V.P. Raju, "Model Based Test Case Generation from UML Activity Diagrams," *The International Journal of Computer Science & Applications (TIJCSA)*, Volume 2, No. 10, December 2013.