Proceedings of the 3rd International Conference on Computing and Informatics, ICOCI 2011,8-9 June, 2011 Bandung, Indonesia
Paper No. 021

AN IMPROVED ALGORITHM IN TEST CASE GENERATION FROM UML ACTIVITY DIAGRAM USING ACTIVITY PATH

Nor Laily Hashim[1], Yasir D. Salman[2]
[1]Universiti Utara Malaysia, Malaysia, laily@uum.edu.my
[2]Universiti Utara Malaysia, Malaysia, s820400@uum.edu.my

ABSTRACT. This paper proposed an improved algorithm to automatically generate test cases directly from UML activity diagram using an activity graph. This algorithm has been implemented as a prototype using UML activity diagrams as inputs to generate test cases. These generated test cases are generated automatically are compared to test cases that are generated manually in order to evaluate the algorithm's usability and reliability. The result shows that the test cases generated by the developed test case generator program are the same as the one manually derived.

Keywords: test case generation, automated generation, test case

INTRODUCTION

Test case generation is one of the most important elements for the testing efforts for programs and applications (Linzhang et al., 2004). It is especially complicated when a system contains simultaneously executing participants, since a system like that can show different responses depending on the simultaneous occurrence conditions. A Unified Modelling Language (UML) activity diagram is a suitable modelling language for describing interactions between system objects given that an activity diagram can be conveniently used to capture business processes, workflows and interaction scenarios (Kim et al., 2007).

Many have generated test cases from UML activity diagrams by generating them manually to test the system (Linzhang et al., 2004; Kundu & Samanta, 2009; Kim et al., 2007). Many generate test cases automatically using different methods (Chen et al., 2009; Javed, Strooper & Watson, 2009), however none of these studies has revealed the algorithm that they have used.

Kundu & Samanth (2009) have developed an algorithm to generate test cases using activity paths. Their algorithm has never been implemented and this algorithm needs an activity graph that has been generated manually from the activity diagram. Their algorithm also needs to enter the type of each node manually and apply some rules to get the result paths.

This paper focuses on extending Kundu & Samanth (2009)'s algorithm used in generating test cases using UML activity diagram by applying activity path approach, in which the test cases are generated automatically generated. The activity path is a method to calculate all possible paths from the activity diagrams, converted each number in the path to its original data, and from there will generate the test cases.

BACKGROUND

The following are the main concepts applied in this study.

UML Activity Diagram

UML is a standard that contains a large set of modelling concepts that are related in complex ways. Since UML became a standard of OMG in 1997, UML models have become main class of artefacts in software development processes. UML provides a number of diagrams to describe different aspects of software artefacts. UML activity diagrams illustrate the sequential control flows of activities. They can be used to represent control flow of an operation or model the dynamic aspects of a group of objects, which form a kind of design specifications for programs (Chen at al., 2009).

In order to directly reuse the activity diagrams modeling of an operation as a test model to generate test cases, it is necessary to follow the testability requirement. An activity diagram should only have one initial activity state, pair of branches and merges, pair of forks and joins. The owner object of each activity state should be labeled by swim lanes, or be labeled in the name of activity state. Every node other than the initial node and final node has at least one outgoing edge and one incoming edge, which means all nodes are reachable. Any fork node only has two exit edges. Concurrent activity states will not access the same object and only execute asynchronously (Linzhang et al., 2004).

Activity graph

An activity graph is a directed graph while its construct is represented by each node in the activity graph (initial node, decision node, flow final node, guard condition, join node, fork node, etc.), and from that each border of the activity graph symbolizes the stream in the activity diagram. Furthermore, the activity graph as shown in Figure 1 summarizes assemble of an activity diagram in an organized way which can be used for further automation (Kundu & Samanta, 2009). These nodes are used for test case generation due to their capability for detecting more faults in the synchronization and loop faults than the other approaches. In addition the ability to identify location of the faults will help to reduce testing afford and having model-based test case generation to improve and develop design quality. Furthermore, it has a possibility to built automatically prototype for the activity graph. The algorithm by Kundu & Samanta (2009) has taken the activity graph as an input and the output from it will be the activity path. Furthermore, the generated paths will need some of rules to apply on the result paths to get all the possible paths that will be needed to generate the test cases.

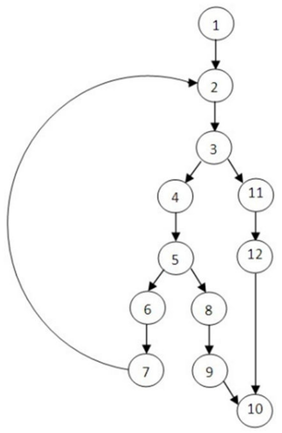


Figure 1. Activity Graph for login screen

Test Case Generation

Test case generation has attracted more and more research attention (Cartaxo, Neto & Machado, 2007; Chen at al., 2009; Heumann, 2001; Javed, Strooper, & Watson, 2007; Kim etal., 2007; Kundu & Samanta, 2009; Linzhang at al., 2004; Mingsong, Xiaokang, & Xuandong, 2006). Large number of them focused on UML activity diagram as a based to generate the test case (Chen, at al, 2009; Kundu & Samanta, 2009; Kim, Kang, Baik, & Ko, 2007; Linzhang at al., 2004; Mingsong, Xiaokang, & Xuandong, 2006), and none of them reveal their proposed approaches or algorithms, or share their testing implementation conducted during the testing.

A test case specifies a set of test inputs, expected results, and execution conditions developed to verify observance with a specific requirement or implement a particular program path (Lilly & G, 2010).

PROPOSED ALGORITHM FOR GENERATING TEST CASES

The following algorithm will generate all the possible paths from the activity diagram, and from the result paths, test cases will be generated. This algorithm is evaluated in terms of its usability and reliability by comparing manual test cases with test cases automatically generated from the prototype of this algorithm. The manual test cases are obtained from Alshammari (2010), where four test cases for student functionalities from Universiti Utara Malaysia learning management system called, LearningZone are used. The functions are login, download assignment, change password, and forum search. Due to space limitation, only one Login function is used and presented in this paper.

Algorithm 1: Generate Test Case

Part A: variables used in the Algorithm

Input: an activity diagram
Output: test case
To find the number of paths, the following are the variables used:
TDN-the number of node which has two directions.
RNN- the number of return node.
TDNI - the number of two directions inside each return node.
RNNI- the number of return node inside each return node.
Basic paths with the return paths number (BPR) = TDN +1
Basic paths number (BP) = BPR - RNN
Number of paths after return (PAR) = TDNI- RNNI
The number of all paths (AP) = BP + PAR(s)

Part B : Pseudo code of methods supported in the algorithm

1. To find the basic paths with the return paths, the following are the variables used:
    a. Node - to put the nodes
    b. LoopFlag - to keep track of the loop
    c. N1 - for the first next of the nodes
    d. N2 - for the second next of the nodes
    e. Nodeflag – for keeping track of the visited nodes
    f. End – for know the end of the path

```
Begin
LoopFlag = TDN, Node = first node, Nodeflag=0, End= true
While LoopFlag is not empty do
  While End is true do
```

```
        If (it's the first node) then put the first node in the path;
            Put the node first next in N1 and its second next in N2;
        If (N1 = 0) then put end to true;
          If (NodeFlag = 0 and N2 = 0) then put N1 for the next node in the path;
                Else if (NodeFlag = 0 and N2! = 0) then put N1 for the next node
              in the path and put NodeFlag = 1, put NodeFlag = NodeFlag – the
              number of two direction node before it;
                    Else if (NodeFlag = 1 and N2! = 0) then put N2 for the
                  next node in the path and put NodeFlag = 2;
        Put the next node of the path as the current node;
      End
      Put the path in array;
    End
```

2. To found all the paths (AP)
   Duplicate the paths that have without End to the (TDNI+ RNNI) number;
   Put the return paths inside it to continue it to the number of paths inside the return path;
   Finish each one from the size basic paths;
   That will generate the possible paths.

3. From the save information of each node, print out the details of each path to the last one, what will give us the test case.

Automatic generate for the test case from the login activity diagram

First: Use the activity diagram, as shown in Figure 2 to generate the test case from it.

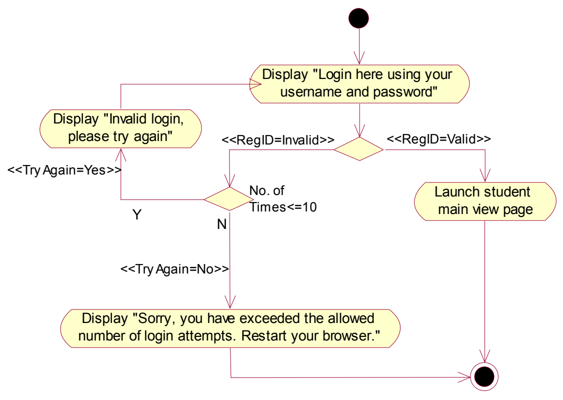
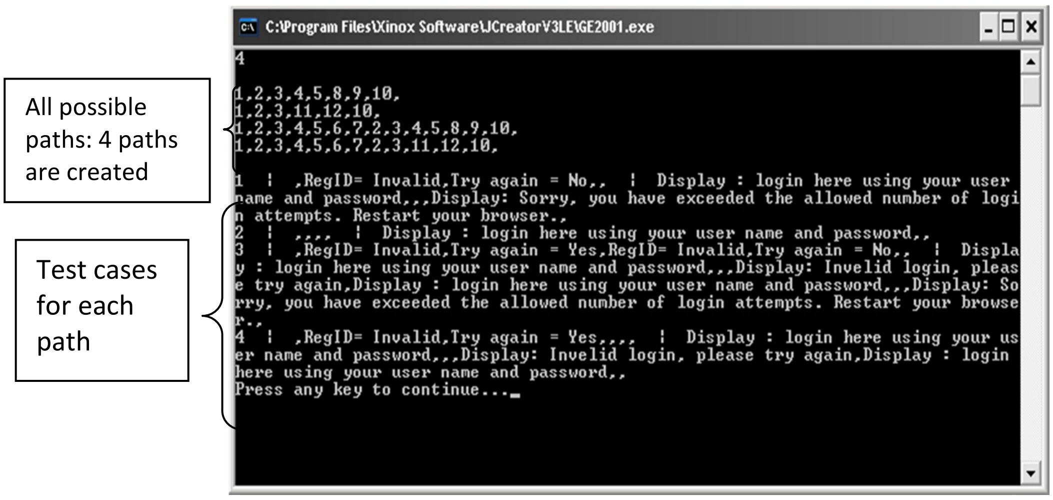

Display "Login here using your username and password"
Display "Invalid login, please try again"
<<RegID=Invalid>>
<<RegID=Valid>>
<<Try Again=Yes>>
No. of Times<=10
Y
N
Launch student main view page
<<Try Again=No>>
Display "Sorry, you have exceeded the allowed number of login attempts. Restart your browser."

Figure 2. Activity Diagram for "Login"

Using the algorithm explained earlier, the prototype will generate the following result as shown in Figure 3


All possible paths: 4 paths are created
Test cases for each path
C:\Program Files\Xinox Software\JCreatorV3LE\GE2001.exe

```
4

1,2,3,4,5,8,9,10,
1,2,3,11,12,10,
1,2,3,4,5,6,7,2,3,4,5,8,9,10,
1,2,3,4,5,6,7,2,3,11,12,10,

1  ¦  ,RegID= Invalid,Try again = No,,  ¦  Display : login here using your user
name and password,,,Display: Sorry, you have exceeded the allowed number of logi
n attempts. Restart your browser.,
2  ¦  ,,,,  ¦  Display : login here using your user name and password,,
3  ¦  ,RegID= Invalid,Try again = Yes,RegID= Invalid,Try again = No,,  ¦  Displa
y : login here using your user name and password,,,Display: Invelid login, pleas
e try again,Display : login here using your user name and password,,,Display: So
rry, you have exceeded the allowed number of login attempts. Restart your browse
r.,
4  ¦  ,RegID= Invalid,Try again = Yes,,,,  ¦  Display : login here using your us
er name and password,,,Display: Invelid login, please try again,Display : login
here using your user name and password,,
Press any key to continue..._
```

Figure 3. The program result for the "login" activity diagram

This result contains all possible paths that are generated from the algorithm based on given activity diagram as shown in Figure 3. The detail for each path together with saved nodes is used to generate the test cases. The test cases generated for Login activity diagram can be seen in Table 1 in the last column. Column two shows the branch conditions that each path follows.

Table 1. The generated test cases for "Login"

Test case	Sequence of Branch Conditions	Activity Sequence
1	RegID=Invalid Try again=No	Display: Login here using your user name and password , Display: Sorry, you have exceeded the allowed number of login attempts. Restart your browser.
2	RegID=Valid	Display: Login here using your user name and password , Launch student main view page.
3	RegID=Invalid Try again=Yes RegID=Invalid Try again=No	Display: Login here using your user name and password, Display: Invalid login, please try again. Display: Login here using your user name and password, Display: Sorry, you have exceeded the allowed number of login attempts. Restart your browser.

4	RegID=Invalid
	Try again=Yes
	RegID=Valid
		Display: Login here using your user name and password,
		Display: Invalid login, please try again,
		Display: Login here using your user name and password,
		Launch student main view page.

Test Result

Four student functionalities have been automatically generated using the proposed test case generator prototype. The generated test cases are compared with the test cases manually generated by Alshammari (2010). From this comparison, it shows that the test cases generated by the developed test case generator program are the same as the one manually derived. Therefore, it can be concluded that the proposed algorithm is usable and reliable.

FUTURE WORK

In the future, this algorithm can be generalized so that it can accommodate various test coverage criteria within the same test derivation framework. This prototype can be improved by developing the ability to build directed graph and parse test cases from it. The UML activity diagrams that have been used in this research were using UML 1.0 version. For the future work, the use of UML 2.0 notation is recommended. Additionally, the UML activity diagrams were generated manually in the work; therefore, the use of a tool that is able to automatically generate these diagrams is suggested.

CONCLUSION

The use of active graph is a good method to ensure that the generation of the test cases automatically using the proposed algorithm and its implementation, and using the UML activity diagram as a base to guarantee the ability of automatic implementation. The reusability of this research work can be applied to complete full automatic analysis. This test will open new opportunity to discover new techniques or methods for the testing using UML diagrams and automatic test cases generation.

REFERENCES

Alshammari, S. A. (2010). Generating Test Cases for LearningZone. *Thesis*, UUM .

Cartaxo, E., Neto, F., & Machado, P. (2007). Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems. *IEEE*, 1292-1297.

Chen, M., Qiu, X., Xu, W., Wang, L., Zhao, J., & Li, X. (2009). UML Activity Diagram-Based Automatic Test Case Generation For Java Programs. *The Computer Journal*, 52 (5), 545-556 .

Heumann, J. (2001). *Generating Test Cases From Use Cases.* Retrieved 5 July, 2010, from: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jun01/GeneratingTestCasesFromUseCasesJune01.pdf

Javed, A., Strooper, P., & Watson, G. (2007). Automated Generation of Test Cases Using Model-Driven Architecture. *IEEE*, 3-9.

Kim, H., Kang, S., Baik, J., & Ko, I. (2007). Test Cases Generation from UML Activity Diagrams. *IEEE*, 556-561.

Kundu, D., & Samanta, D. (2009). A Novel Approach to Generate Test Cases from UML Activity Diagrams. *Journal of Object Technology*, 8 (3), 65-83.

Lilly, R., & G, U. (2010). Reliable Mining of Automatically Generated Test Cases from Software Requirements Specification. *International Journal of Computer Science Issues*, 87-91.

Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., & Guoliang, Z. (2004). Generating Test Cases from UML Activity Diagram based on Gray-Box Method. *IEEE*, 284-291.

Mingsong, C., Xiaokang, Q., & Xuandong, L. (2006). Automatic Test Case Generation for UML Activity Diagrams. *ACM*, 2 - 8 .