Journal of Scientific Research & Reports
11(5): 1-10, 2016; Article no.JSRR.26872
ISSN: 2320-0227

SCIENCEDOMAIN *international*
www.sciencedomain.org

Test Cases Generation for Object-oriented Software from Decision Slicing of UML Activity Diagram

Wasiur Rhmann[1*] and Vipin Saxena[1]

[1]Department of Computer Science, B. B. Ambedkar University (A Central University), Lucknow, U.P., India.

Authors' contributions

This work was carried out in collaboration between both authors. Both authors read and approved the final manuscript.

Article Information

DOI: 10.9734/JSRR/2016/26872
Editor(s):
(1) Vito Di Maio, Institute of Cybernetics "E. Caianiello" CNR, C / O Complex Olivetti, Via Campi Flegrei, 34, Italy.
(2) Luigi Rodino, Professor of Mathematical Analysis, Dipartimento di Matematica, Università di Torino, Italy.
Reviewers:
(1) Boni García, Universidad Rey Juan Carlos, Spain.
(2) Yulia Kartika Gunadi, Petra Christian University, Indonesia.
(3) Mukesh Mann, Gautam Budda University, India.
Complete Peer review History: http://www.sciencedomain.org/review-history/15675

Original Research Article

Received 7th May 2016
Accepted 22nd July 2016
Published 5th August 2016

ABSTRACT

Software testing is an integral part of the software development life cycle. Design of good test cases is a key challenge in software testing. Test cases can be designed from different artifacts like requirements, design and software code. In Software engineering, different UML diagrams are used for designing and analysis of the software systems. The main contribution of this work is to propose a novel technique of test cases generation from UML activity diagram using an iterative method. Iterative methods are used in numerical analysis for generation of solution of equation iteratively. In the present work a test cases generation technique from decision slicing of UML activity diagram is presented. Decision slices for each decision nodes are derived from the Activity Flow Graph (AFG) of the activity diagram. Test cases are generated for each activity path of the activity flow graph. Decision nodes at each activity path are used to generate system of equations and these equations are solved by an iterative method to generate test data for each activity path. A case study of ticket purchasing from ticket vending machine using UML activity diagram is presented.

**Corresponding author: E-mail: wasiurrhmann786@gmail.com;*

Keywords: Software testing; program slicing; activity diagram; iterative method; test cases.

1. INTRODUCTION

Software testing is a creative and challenging part of the software development life cycle. Quality and reliability of the developed software products is ensured with the help of software testing. Software testing cost generally depends on the how efficiently testing is performed. There are mainly three activities in test process: test cases generation, test case execution and test evaluation. Generation of test cases is most critical activity. A test case is the triplet [I, S, O] where I is input data, S is the state of the system and O is expected output [1-2].

There are mainly two approaches of software testing namely: white box and black box. In white box testing, there is need to know internal details of programs and in black box testing test cases are generated with the requirement specification documents and internal structure of program is not considered. In black box testing, test cases are designed with consideration of the functionality of the software system. Model based testing is a kind of black box testing which is used to find bugs earlier in the software development life cycle. In model based testing expected models which capture requirements of the software under test are created and a test tool is used to automatically derive the test cases from the designed models [3].

Large systems are more complex to test. To reduce the development and testing costs of modern software, software engineers usually advocate the use of object-oriented analysis and design paradigm [4]. Testing object oriented system with the code is tedious and complex task. Object Management Group (OMG) [5] introduced the standardized Modeling Language called Unified Modeling Language [6] to facilitate the use of standardized notation for object-oriented analysis and design. Design models are used by tester to test the object oriented software. Activity diagram is designed with higher level of abstraction so it contains less information in comparison to other UML diagrams like sequence diagram, class diagram. Software developers are heavily using models in the development of the software products [7]. Information required for testing can be generated by simple processing of these models. Testing an executable form of software artifacts involve three phases [8] namely: design of test cases; execute the test cases with the executable software artifacts; analysis the behavior of the results. Test adequacy criteria are proposed by researchers to check up to how much extent a property must be tested [9].

Unified Modeling Language is used for creating the software blueprints. It consists of different notations to support design and analysis of the object-oriented software development. It facilitates to use different diagrams to present static and dynamic views of the systems. Activity diagram focuses on the flow of behavior of a system [10]. Different UML diagrams like class diagram, use case diagram and activity diagram are used to represent system at different level of abstraction. Activity diagram is like flow chart but it is more expressive than flow chart. Sequential or concurrent control flow can be represented by an activity diagram. UML activity diagram is used by several researchers as initial specification of the system for generation of test cases [11-13].

Large and complex software can be decomposed into manageable pieces of code to easily handle the complex software and there are accurate dependency exists among different pieces of the software. Program slicing techniques are used to handle the complexity of programs which arises due to large size of programs. Program slices extracts are those statements of the programs which are relevant to a particular computation [14]. Dynamic slices are often smaller size than static slices as it includes only those statements of program which are executed for particular input data.

Program slicing techniques are used in different aspects of the software development like debugging [15], software maintenance [16], software measurement [17] and software testing [18]. In the present work, authors used a new slice method of UML activity diagram called decision slice and generated test cases for each activity path. Test data corresponding each activity path is generated using an iterative method of numerical analysis. Iterative method is used in [19] for generating tests data in the context of program code. In our approach iterative method is used at design level for test cases generation. The rest of the paper is organized as follows: Section 2 of the paper describes the UML activity diagram and iterative method used for test cases generation. Section 3 presents the methodology used for test cases generation. A case study of automatic ticket vending machine is presented in section 4. In section 5 a comparative study of proposed technique with other techniques in literature is given and finally in section 6 authors concluded the work.

2. BACKGROUND

2.1 UML Activity Diagram

In the current work, UML activity diagram is used to represent the functionality of the system. Activity diagram contains mainly two types of nodes namely action nodes and control nodes. Action nodes contains Activity, Action, SendSignal and AcceptEvent while Control nodes contains InitialNode, FinalNode, Decision, Merge, Fork and Join. There are several levels of activity modeling: Basic, Intermediate, Complete and Structured Complete and extra structure activities in UML 2.1 superstructure [20]. Test scenarios are used to check whether all business process related to the software are tested end to end. Test cases and test scenarios are often used by Software tester synonymously [21]. An activity diagram(AD) as defined in [22] is based on seven tuple described below:

AD= (A, T, F, J, R, a_i, a_f)

where

A is finite set of activities a_0, a_1,… .,a_n, representing nodes, T is finite set of transitions t_0, t_1,…..,t_n representing (edges), F is finite set of forks f_0,f_1,…,f_n. , J is finite set of join j_0, j_1,….,j_n. and R $\subseteq$(A X T) $\bigcup$ (T X A) is the flow relation, a_0 is initial activity node and a_f is final activity node.

2.2 Stationary Iterative Method

In numerical analysis iterative methods are used to generate improved approximate solution for a class of problems [23]. There is a termination criteria defined to solve the problem with iterative method. Initial data is guessed to find out the solution of the system of equations and successively approximate solutions which are closer to the results are generated.

These methods are also called relaxation methods. These methods use the error in the result to approximate the solution and error in the result is called residual. Stationary iterative methods use an operator to approximate the solution of linear system.

Let an equation ax+by+cz+d=0 (1)

and its approximate solution is (x_0,y_0,z_0), then substituting the values of (x_0, y_0, z_0) in the equation we will get a positive value residual.

$$ax_0+by_0+cz_0+d=r_0 \quad (2)$$

If increments in x, y and z are such that they satisfy the linear constraint

$$a\Delta x+b\Delta y+c\Delta z=-r_0 \quad (3)$$

Then from 2 and 3

$$a(x_0+\Delta x)+b(y_0+\Delta y)+c(z_0+\Delta z)+d=0 \quad (4)$$

Next approximate solution of the equation will be (x_1, y_1, z_1)=(x_0+Δx,y_0+Δy, z_0+Δz).

In the present technique of test cases generation from UML activity diagram, authors used this iterative method. For formulating the problem of test cases generation from activity diagram as an iterative method, we derived the decision functions corresponding to each decision nodes of the activity diagram. These decision functions are used to derive the system of equations. Here initial solutions of inputs are guessed and increments in the initial values are computed by solving the equations.

3. PROPOSED METHODOLOGY

The steps in the proposed methodology are described below:

1. Draw UML activity diagram;
2. Convert the activity diagram into Activity Flow Graph(AFG);

   AFG for UML activity diagram is drawn, for each component of activity diagram there is a node in AFG. Decision nodes are represented by circle and for fork/join rectangle are used.

3. From AFG generate activity paths;

   Activity paths are generated by traversing the activity flow graph in DFS manner and if there is fork node then BFS are used in traversing the nodes of activity flow graph.

4. For each activity path, test data is generated. The decision nodes of each activity paths are used to form the equations;

   4.1 **Decision slice** Decision slice of a decision node (DN) on path P is the set of nodes of AFG, upon which the outcome of decision node depends directly or indirectly.
   4.2 Then compute the input dependency set for each decision node on activity path. A subset of input (I) variables along an activity path (P) on which decision node (DN) outcome depends is called **input dependency set** ID (DN, I, P).
   4.3 Formulate the decision functions from decision nodes. Decision functions are formulated based on input variables on which that decision node depends. Suppose if a decision node i depends on input variables x, y and z.

Then decision function will be F_i: $a_ix+b_iy+c_iz+e_i$

For computing the coefficients a_i, b_i and c_i with respect to their variables evaluate the decision function at the current input $I_k=(i_1,\ldots.i_j,\ldots.i_m)$ and at $I_k+(0,\ldots..,\Delta i_j,\ldots 0)$ and compute the divided difference by using the formula given below

$$F(I_k+(0,\ldots., \Delta i_j,\ldots,0)- F(I_k)/ \Delta i_j$$

This gives the coefficient of i_j in the linear function for the decision function F corresponding to node i in P. Similarly other coefficients are computed.

Decision functions are also formulated from values of each decision nodes on activity path; If a decision nodes have no input variables then decision function for that decision node will contain one Boolean variable with 0 for false and 1 for true value of the function.

   4.4 Formulate the linear equations from decision nodes.

   Convert linear functions into inequalities. For conversion of linear functions into inequalities we choose >,= and < operators. If a decision node on an activity path needs to be true for traversal of that path then relational operator will be same as used in decision node else it will be reverse of the decisional nodes.

Compute decision residuals. Decision residual of a decision node for an input is the value of decision function computed by executing its decision slice at that input;

Then apply the relaxation technique and formulate the inequalities of increment of the input variables and convert these into equations and solve these equations. On solving these equations we get the increment values which will give the next input values. This process of iteration continues till we get the values which traverse the activity path.

4. A CASE STUDY OF TICKET PURCHASING FROM AUTOMATIC TICKET VENDING MACHINE

In the present section UML activity diagram is designed for ticket purchasing from automatic ticket vending machine and test cases are generated using presented approach.

Fig. 1 presents UML activity diagram for purchasing tickets from ticket vending machine. There are three partitions which are Passenger, Ticket Vending Machine and Bank. Passenger can perform different activities. Passenger selects the ticket types and destination of travel which inputs the price of ticket. Then passenger selects number of ticket types (n). Then ticket vending machine computes the total amount and displays the total fare. Then machine process for payment. Ticket vending machine accepts money either by card or cash. If passenger pays with card then bank is used for authorization of payment and ticket is issued. If passenger inserts cash then machine calculates the difference between inserted money and total fare. If money inserted is less than the total fare then a message of insufficient amount is displayed. If difference between total fare and money inserted is 0 then ticket is issued and money inserted is more than the total fare then machine issues ticket along with the returned money.

Fig. 2 represents the activity flow graph of activity diagram given in Fig. 1. In Fig. 1 there are 21 different constructs like start, end node, activities, fork, join and decision node these are converted into nodes of activity flow graph. Nodes 8, 13 and 15 represent decision nodes of activity diagram. Nodes 17 and 20 represent the fork and join nodes of activity diagram respectively.

Activity paths from activity flow graph are generated by traversal the graph in depth first search manner and there is a fork and join node then node between them are traversed in breadth first search manner which is activity path4. Generated activity paths are given below:

Activity Path1 1-2-3-4-5-6-7-**8**-9-16-21

Activity Path2 1-2-3-4-5-6-7-**8**-9-10-12-**13**-**15**-16-21

Activity Path3 1-2-3-4-5-6-7-**8**-9-10-12-**13**-14-21

Activity Path4 1-2-3-4-5-6-7-**8**-9-10-12-**13**-**15**-17-18-19-20-21

Below the steps of test case generation for activity path2 are given:

4.1 Derivation of Decision Functions

Let x for the decision nodes which do not contains constraints and values of x are 0 or 1.

Activity path 2 contains three decision nodes 8, 13 and 15. For each decision nodes presented in the corresponding activity path, we identify the input variables which will affect the outcome of the decision node. It is given by

$$ID(DN_1,P)=x \quad (5)$$

$$ID(DN_2, P)=(p, n, a) \quad (6)$$

$$ID(DN_3, P)=(p, n, a) \quad (7)$$

For each decision nodes available in the corresponding activity path authors formulated the linear equations. Activity path2 has three decision nodes so there will be three equations:

$$F_1=a_1x+e_1 \quad (8)$$

$$F_2=a_2p+b_2n+c_2a+e_2 \quad (9)$$

$$F_3=a_3p+b_3n+c_3a+e_3 \quad (10)$$

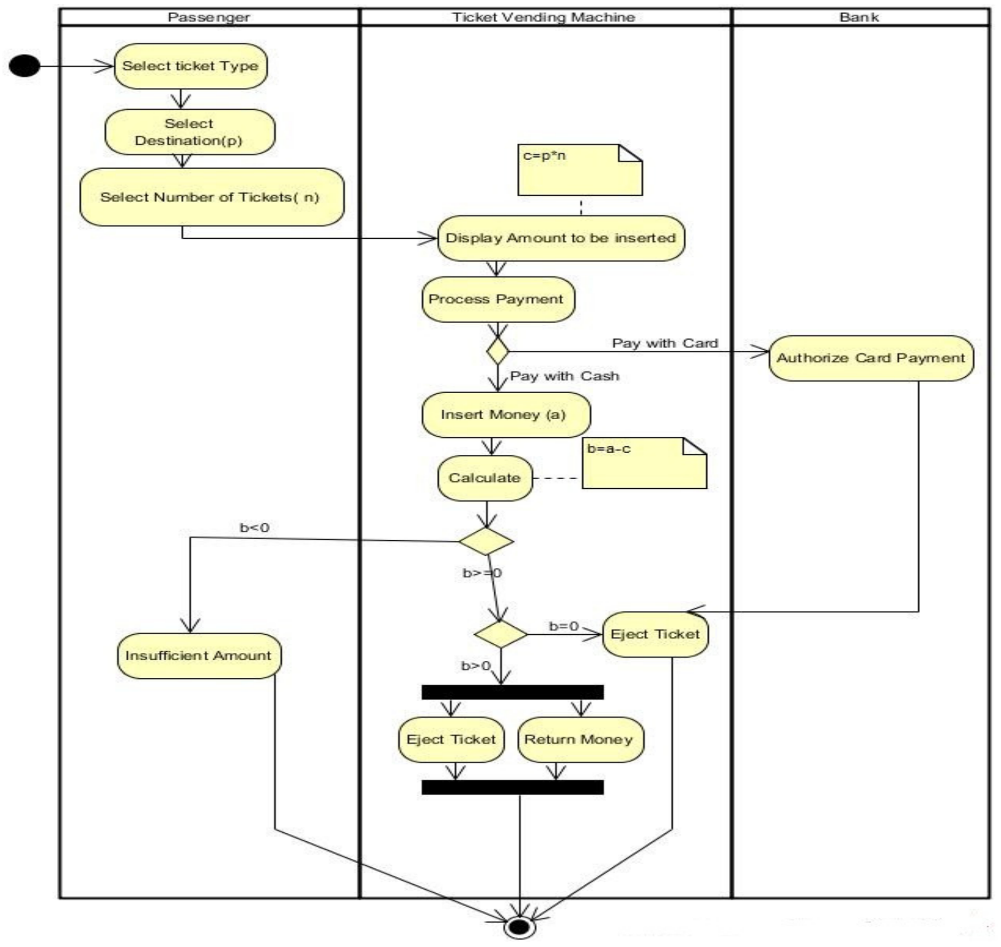


Fig. 1. UML activity diagram for ticket purchasing from automatic ticket vending machine

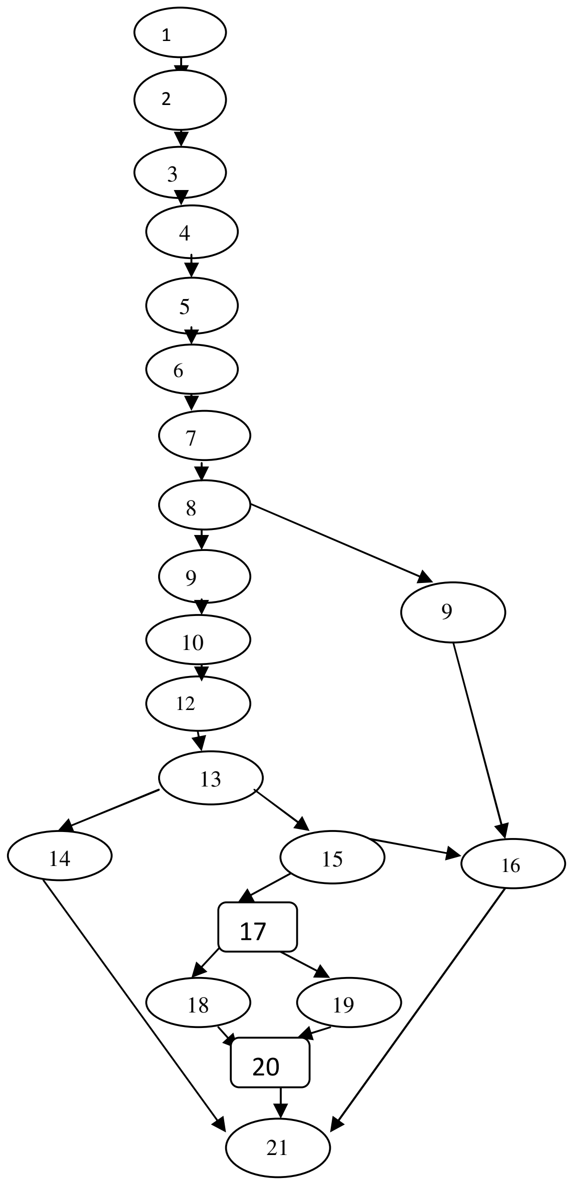


Fig. 2. Activity flow graph (AFG) for Fig. 1

Here equation 8 represents decision node with no input variables so it takes the form of equation 11;

$$F_1=x \tag{11}$$

We select x=1 for traversal of activity path2.

Decision functions corresponding equations 9 and 10 are formulated from values of corresponding decision nodes on activity path and are given by equations;

$$F_2=b-0 \tag{12}$$

$$F_3=b-0 \tag{13}$$

Let I_0=(p, n, a)=(100, 2, 220) then input will be (x, p, n, a)=(1, 100, 2, 220), this will not traverse the activity path2.

Then next input is obtained using the above iterative method:

We take increment $\Delta p=2$ and computed the values of F_1 at I_0.

and at $(p_0, n_0, a_0)+(\Delta p,0,0)=(100, 2, 220)+ (2, 0, 0)=(102, 2, 220)$.

Then divided difference $F_1(p_0+\Delta p,n_0,a_0)-F_1(p_0,n_0,a_0)/\Delta p$ is calculated which gives the value of a_1

$a_{1=}(16-20)/2=-2$,

Similarly other values are calculated and recorded in Table 1.

Table 1. Values of coefficient of equations

a_1	a_2	b_1	b_2	c_1	c_2
-2	-2	-100	-100	1	1

Values of F_2 and F_3 given in equations 12 and 13 are computed at I_0 and recorded in Table 2

Table 2. Calculation of functions with the input values

a	p	n	F=b=a-c=a-p*n
220	100	2	F_2=20
220	100	2	F_3=20

Constant terms e_i in the equations 9 and 10are computed by executing the decision slices at I_0. Then the values of input I_0 and coefficient of equations recorded in Table 1 are put in the linear equations 9, 10 and equated with the corresponding values computed in Table 2.

$$a_1100+2b_1+220c_1+e_1=20 \tag{14}$$
$$-200-200+220+e_1=20$$
$$e_1=200,$$

Similarly value of e_2 is calculated and the values of a_i, b_i, c_i and e_i and corresponding functions 9 and 10 are recorded in the Table 3.

4.2 Computation of Decision Residuals

Decision slices corresponding each decision nodes on the activity path are executed with current program inputs I_0 and values of decision functions are evaluated.

Table 3. Derivation of decision functions

a_1	-2	a_2	-2
b_1	-100	b_2	-100
c_1	1	c_2	1
e_1	200	e_2	200
F_2	-2p-100n+a+200	F_3	2p-100n+a+200

Decision residual at I_0 for decision nodes on Activity Path2 are recorded in Table 4:

Table 4. Computation of decision residual

I_0	Residual
(100, 2, 220)	$R(DN_2, I_0, AP_2)=20$
(100, 2, 220)	$R(DN_3, I_0, AP_2)=20$

Construction of a system of linear constraints and to solve them to obtain increments for the current input:

Linear arithmetic representation of decision nodes are converted into inequalities:

Decision nodes corresponding decisional functions F_2 and F_3 which are given in Table 2 needs to be true for traversal of Activity path2 So we have taken the same relational operator as used in these decisional nodes and inequalities corresponding each nodes is written in equations 15, 16.

$$-2p-100n+a+200>=0 \tag{15}$$

$$-2p-100n+a+200=0 \tag{16}$$

Then relaxation technique is applied to equations 15 and 16 and decision residual computed in Table 3. Set of constraints on increments of p, n and a are given in equations 17 and 18.

$$-2\Delta p-100\Delta n+\Delta a>=-20 \tag{17}$$

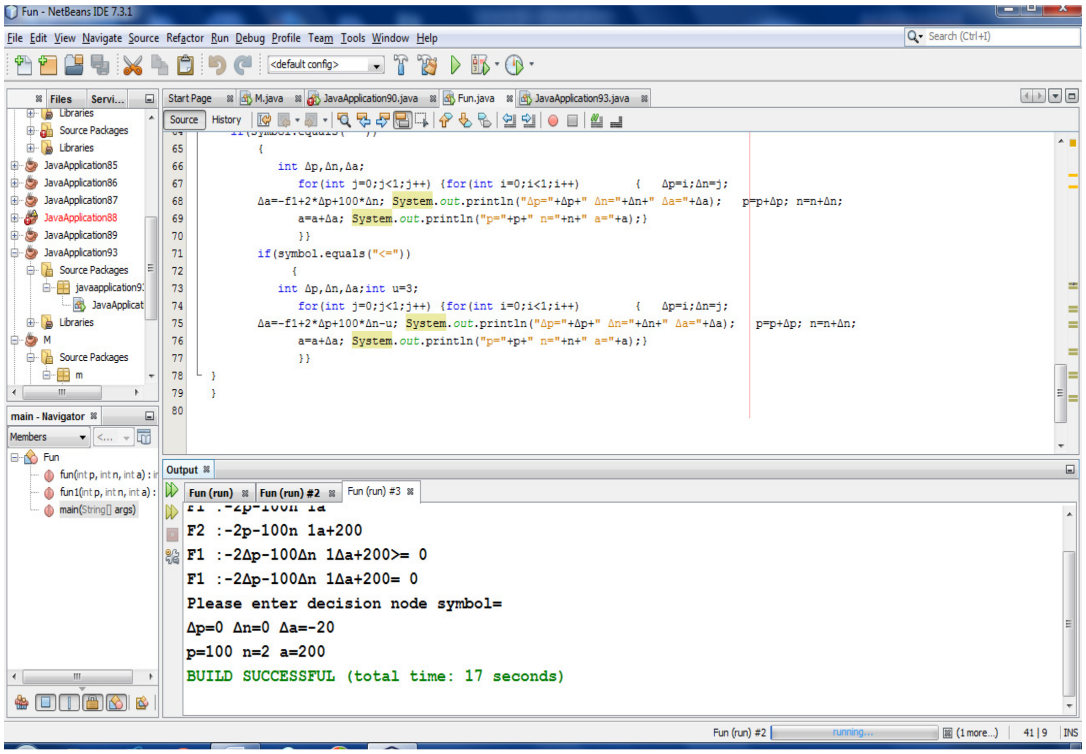
$$-2\Delta p-100\Delta n+\Delta a=-20 \tag{18}$$

4.3 Converting the Inequalities into Equations

Inequalities are converted into equations by introducing the variable u:

$$-2\Delta p-100\Delta n+\Delta a-u=-20 \tag{19}$$

$$-2\Delta p-100\Delta n+\Delta a=-20 \tag{20}$$



```
        {
          int Δp,Δn,Δa;
            for(int j=0;j<1;j++) {for(int i=0;i<1;i++)        {   Δp=i;Δn=j;
        Δa=-f1+2*Δp+100*Δn; System.out.println("Δp="+Δp+" Δn="+Δn+" Δa="+Δa);   p=p+Δp; n=n+Δn;
            a=a+Δa; System.out.println("p="+p+" n="+n+" a="+a);}
            }}
        if(symbol.equals("<="))
             {
          int Δp,Δn,Δa;int u=3;
            for(int j=0;j<1;j++) {for(int i=0;i<1;i++)        {   Δp=i;Δn=j;
        Δa=-f1+2*Δp+100*Δn-u; System.out.println("Δp="+Δp+" Δn="+Δn+" Δa="+Δa);   p=p+Δp; n=n+Δn;
            a=a+Δa; System.out.println("p="+p+" n="+n+" a="+a);}
            }}
  }
  }
```

```
F2 :-2p-100n 1a+200
F1 :-2Δp-100Δn 1Δa+200>= 0
F1 :-2Δp-100Δn 1Δa+200= 0
Please enter decision node symbol=
Δp=0 Δn=0 Δa=-20
p=100 n=2 a=200
BUILD SUCCESSFUL (total time: 17 seconds)
```

Fig. 3. Screenshot of test data generated for test path2

Table 5. Test cases number, test paths and input data corresponding each test path

Test case no.	Test path	Input data (x, p, n,a)
TC_1	Select ticket Type->Select Destination(p) -> Select Number of Tickets(n) -> c=p*n -> Display Amount to be inserted -> Process Payment-> Authorize Card Payment -> Eject Ticket ->end	(0,0,0,0)
TC_3	Select ticket Type->Select Destination(p) -> Select Number of Tickets(n) -> c=p*n -> Display Amount to be inserted -> Process Payment-> Insert Money(a) -> Calculate -> b=a-c -> Insufficient Amount->end	(1, 100, 3, 200)
TC_2	Select ticket Type->Select Destination(p) -> Select Number of Tickets(n) -> c=p*n -> Display Amount to be inserted -> Process Payment-> Insert Money(a) -> Calculate -> b=a-c -> Eject Ticket ->end	(1, 100, 2, 200)
TC_4	Select ticket Type->Select Destination(p) -> Select Number of Tickets(n) -> c=p*n -> Display Amount to be inserted -> Process Payment-> Insert Money(a) -> Calculate -> b=a-c -> Eject Ticket -> Return Money -> end	(1, 100, 2, 2010)

On solving the equations 19 and 20, we found the values of u, Δa, Δp, Δn given below:

$u=0,$
$-2\Delta p-100\Delta n = -20- \Delta a$
$\Delta a=-20,\ \Delta p = 0,\ \Delta n=0$

New values of x, p, n, a are obtained by adding the values of Δa=-20, Δp =0, Δn=0 to (x, p, n, a).

(x, p, n, a)=(1, 100, 2, 200).

Here computed new values lead to traversal of the desired activity path and computed new approximation of test inputs which traverse an activity diagram is obtained from previous approximation of the solution and it's residual. This technique will be used iteratively to obtain new input values until desired activity path is traversed.

Similarly test data for other test paths are calculated and recorded in Table 5.

Java programming language is used for implementation of the proposed approach. For the given problem there are two decision nodes 13 and 15 which take input values. Java program contains functions of the decision nodes 13and 15 of the programs. And Initial input is taken in the program as (p, n, a) = (100, 2, 220).

Then decision node symbol is provided based on the path traversal then our program generates test data corresponding that path. Fig. 3 shows the test data for test path 2.

5. COMPARISON WITH THE RELATED WORKS

Swain et al. [24] presented a technique of test case generation using UML 2.0 sequence diagram. Authors constructed a Message Dependency Graph from sequence diagram. Conditional predicates are selected from message dependency graph by traversing the graph. Slices are computed for each conditional predicates and test cases are generated for each slice. Test data is generated by satisfying all constraints corresponding each slice. Swain et al. [25] used condition slicing and generated test cases from UML interaction diagram. Authors used message guards of interaction diagram and created conditioned slice for each message guards. For each conditioned slices test cases are generated. This approach is advantageous when number of messages in sequence diagram is in large number. Li et al. [26] used extenics theory for generation of test cases from UML activity diagram. Authors converted the UML activity diagram into Euler circuit and applied the Euler circuit traversal algorithm to generate test cases. Generated test cases consist of test cases sequences there is no input data.

Jena et al. [27] used UML activity diagram for test paths generation and test cases are generated from Activity flow graph of the activity diagram by traversing the graph in Depth First Search manner. Technique presented by authors only generates test paths not exactly test cases. Kundu and Samanta [28] presented an approach of test cases generation from UML 2.0 with use case scope. Authors considered test coverage criteria called activity path coverage criteria. Test cases generated from their approach can detect faults like synchronization faults, faults in loops. Their technique will not be efficient when the activity diagram will be larger.

Samuel and Mall [29] used dynamic slicing technique on the flow graph of activity diagram then sliced the diagram for each conditional predicates and test cases generated for each slice. In their technique if there are large number of conditional predicates then there will be a lot of test cases corresponding each conditional predicated while in our technique conditional predicates on each path are considered simultaneously to generate test data which will reduce the number of test cases. In their approach authors do not check each valid test path. Their technique generates two test cases for each condition while our technique generates only one test case for each condition or decision nodes. In Fig. 4 number of generated test cases from our technique and technique presented by Samuel are compared. In the presented case study there are four test cases generated while by using Samuel technique there will be 6 test cases corresponding each conditional node in the flow graph as there are 3 conditional nodes in flow graph.

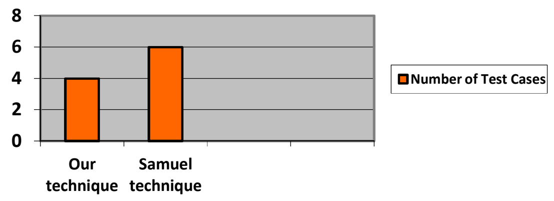


Fig. 4. Number of test cases by our technique and technique presented by Samuel

*% reduction in number of test cases will be=2/6*100=33.33%*

6. CONCLUSIONS

Slicing was initially developed for software programs. It helps to manage the complexity of the large programs. In the present work, test cases are generated from UML activity diagram using an iteration method. For each test path generated from AFG of the UML activity diagram there is test data which executes that test path. Our approach used a decision slicing criteria which slices the activity diagram based on decision nodes of the activity diagram. Then iterative method is used to generate the test data corresponding each path of the activity diagram. This method is an innovative method which uses the iterative method of numerical analysis for generation of test cases from UML activity diagram. Most of the techniques used by researchers generate only test paths of activity diagram while our technique generates test paths along with the input data. Our approach generates the reduced number of test cases and test data generated by our approach satisfies the activity paths. In future research other UML diagrams may be used with the iterative methods for test data generation. Presented relaxation based technique which is used to solve the equations of test paths may be useful in identifying the infeasible test paths from UML diagrams and identification of infeasible paths from UML may cut down the testing cost drastically.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. Mall R. Fundamentals of software engineering. Second Edition, Prentice Hall, India; 2003.
2. Offut J, Abdurazik A. Generating tests from UML specifications. Proceeding of the second international conference on UML, lecture notes in computer science. Berlin. 1999;416-429.
3. Utting M, Legeard B. Practical model based testing a tool approach. Elsevier. San Francisco; 2007.
4. Larman C. Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process. 2nd edition, Prentice Hall Professional; 2002.
5. OMG UML Specification. OMG unified modeling language TM, super structure version 2.2; 2011. Available:http://www.omg.org/spec/UML/2.2/Superstructure
6. Booch G, Raumbaugh J, Jacobson I. The unified modeling language user guide (second edition). Addison Wesley; 2005.
7. France R, Ghosh S, Trong TD, Solberg A. Model-driven development using UML 2.0: promises and pitfalls. IEEE Computer. 2006;39(2):59-66.
8. Andrews A, France R, Ghosh S, Craig G. Test adequacy criteria for UML design

models. Software testing verification and reliability. ISSRE. 2003;13(2):95-127.
9. Andrews A, France R, Ghosh S, Craig G. Test adequacy assessment for UML design model testing. 14th International Symposium on Software Reliability Engineering. 2003;332-343.
10. Pilone ND, Pitman N. UML 2.0 in nutshell. O' Reilly; 2005.
11. Linzhang W, Jiesong Y, Xiaofeng Y, Jun H, Xuandong L, Guoliang Z. Generating test cases from UML activity diagram based on gray-box method. In 11th Asia-pacific Software Engineering Conference (APSEC04). 2004;284-291.
12. Rhmann W, Zaidi T, Saxena V. Use of genetic approach for test case prioritization from UML activity diagram. International Journal of Computer Applications. 2015; 115(4):8-12.
13. Boghdady PN, Badra NL, Hashem M, Tolba MH. A proposed test case generation technique based on activity diagrams. International Journal of Engineering and Technology. 2011;11(3): 37-57.
14. Canfora G, Cimitile A, Lucia AD. Conditioned program slicing. Information and Software Technology. 1998;40(11): 595–607.
15. Weiser M. Programmers use slicing when debugging. Communication of the ACM. 1982;25(7):446-452.
16. Gallagher KB, Lyle JR. Using program slicing in software maintenance. IEEE Transactions on Software Engineering. 1991;17(8):751-761.
17. Harman M, Okunlawon M, Sivagurunathan B, Danicic S. Slice-based measurement of coupling. 19th ICSE, Workshop on Process Modeling and Empirical Studies of Software Evolution, Boston, Massachusetts, USA; 1997.
18. Harman M, Danicic S. Using program slicing to simplify testing. Journal of Software Testing, Verification and Reliability. 1995;5(3):143-162.
19. Gupta N, Mathur AP, Soffa ML. Automated test data generation using an iterative relaxation method. ACM SIGSOFT. 1998; 11:231-244.
20. OMG. Unified modeling language (UML) Superstructure Specification, version 2.1. Technical report.
21. Swain RK, Panthi V, Mohpatra DP, Behera PK. Prioritizing the test scenarios from UML communication and activity diagrams. Innovations Syst. and Soft. Eng. 2014; 10(3):165-180.
22. Sapna PG, Balkrishnan AK. An approach of generating minimal test cases for regression testing. Procedia Technology. 2015;87:188-196.
23. Scheid F. Numerical analysis. Schaum's Outline Series. McGraw-Hill Book Company; 1968.
24. Swain RK, Panthi V, Behera PK and Mohapatra DP. Slicing based test case generation using UML 2.0 sequence diagram. Int. J. Computational Intelligence Studies. 2014;3:2-3.
25. Swain RK, Panthi V, Behera PK. Test case design using slicing of UML interaction diagram. 2nd International Conference on Communication, Computing & Security, Procedia Technology. 2012;6: 136-144.
26. Li L, Li X, He T, Xiong J. Extenics-based test case generation for UML activity diagram. First International Conference on Information Technology and Quantitative Management, Procedia Technology. 2013; 17:1186-1193.
27. Jena AK, Swain SK, Mohapatra DP. A Novel Approach of test case generation from UML Activity diagram. International Conference on Issues and Challenges in Intelligent Computing Techniques. 2014; 621-629.
28. Kundu D, Samanta D. A novel approach to generate test cases from UML activity diagram. Journal of Object Technology. 2009;8(3):65-83.
29. Samuel P, Mall R. Slicing-based test case generation from UML activity diagrams. ACM SIGSOFT Software Engineering Notes. 2009;34(6):1-14.

Peer-review history:
The peer review history for this paper can be accessed here:
http://sciencedomain.org/review-history/15675