International Journal of Advanced studies in Computer Science and Engineering
IJASCSE, Volume 4, Issue 2, 2015

Test case minimization for object oriented technique on the basis of object oriented coupling

SUPRIYA
Deenbandhu Chhotu Ram University of Science & Technology, Sonepat, Haryana

CHINKY ANEJA
Kurukshetra University, Kurukshetra, Haryana

Abstract— Test case minimization techniques involve selecting those test cases in an order that improves the performance of testing. It is inefficient way in which every test cases for every program function should be test at least once if any change occurs. Test case minimization; minimize test cases in a test suite in an order that increases the effectiveness in achieving some performance goals. One of the most important performance goals is the rate of fault detection. Test cases should run in an order that increases the possibility of fault detection and also that detects the most severe faults at the earliest in its testing life cycle. The testing approach used for object oriented programming differs from the one used for conventional programming because an object-oriented programming language provides explicit support for polymorphism, inheritance, overloaded functions, generic functions, templates etc. These features improve the quality of software development but on the other hand these features make traditional software testing difficult to adapt to OO based software testing. Also with strong coupling the calculation of faults cannot be done efficiently. Therefore there should be low coupling and high cohesion in between the modules .The interaction among the modules propagates the features can be reused and if any error occurs the propagation makes the system fails. Which is also called RIPPLE EFFECT ANALYSIS, which is introduced in change impact analysis in coupling impact

Keywords — ***Activity diagram, Coupling,, Change impact analysis, Cyclomatic complexity, Fault detection technique ,Object oriented techniques, Test case generation.***

I. INTRODUCTION

Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not. Testing is a process of demonstrating that there are reduced critical errors in the software. Testing can always show the presence of errors and not the absence of errors.

For the purpose of doing static and dynamic testing, various techniques are followed. System testing is also one of major type. In it load testing, stress testing and volume testing are used for checking performance, recovery testing to check recovery point, configuration testing to check various configuration on the system and regression testing for the purpose of revalidation of the new versions of the existing software and many more.

Coupling, as the name suggests, is the interaction between the two modules. In object oriented testing the testing, involves the testing of objects and classes that are the part of module which is impractical and inefficient with test cases to execute every test for every program function if once any error occurs due to the constraints of time and cost of the project specially when the modules are binded with coupling. Most of the time the testing team is asked to checks last minute changes in the code just before making a release to the client, in this situation the testing team needs to check only the affected areas. So in short for the test case minimization technique help the testing team should get the input from the development team about the nature/amount of change in the fix so that testing team can first check the fix and then the side effects of the fix. Test under coupling should follow on critical module function. To make object oriented testing easier, software engineers typically reuse test suites of the original program, but also new test cases may be required to test new functionalities of the new version. The focus here is on the reuse of test cases as most ideas about costs and benefits come from test suite granularity. There are four methodologies, considered here, that reuse the test suites of the original version of the software: *retest-all, coupled test selection (CTS) , test suite reduction (TSR)* and *test case minimiization (TCM).* Retest all, reruns every test case of the test suite. It is not a feasible approach as time period to complete the work is fixed. CTS, on the other hand selects some of the test cases from the test suite on temporary basis where as TSR permanently removes test cases from the test suite depending upon the modification in the existing project. Selection and removal of test cases from test suite can be problematic in some situation. So, a new methodology is there namely test case minimization.

Test case minimization techniques reduce the number of test cases that are better at achieving the testing objectives are run earlier testing cycle.

There are many possible goals of minimization, including the following:

- Testers may wish to increase the rate of fault detection of a test suite, that is, the likelihood of revealing faults earlier in a run of minimized tests using that test suite.

- Testers may wish to increase the coverage of coverable code in the system under test at a faster rate, allowing a code coverage criterion to be met earlier in the test process.
- Testers may wish to increase their confidence in the reliability of the system under test at a faster rate.
- Testers may wish to increase the rate at which high-risk faults are detected by a test suite, thus locating such faults earlier in the testing process.

Software testing is a strenuous and expensive process. Research has shown that at least 50% of the total software cost is comprised of testing activities. Here we are working with test case minimization technique of object oriented technique with the impact of coupling involves the change impact analysis in which the coupled modules shows the presence of any change. Test case minimization (TCM) involves the explicit planning of the execution order of test cases to increase the effectiveness of software testing activities by improving the rate of fault detection earlier in the software process. To date, TCM has been primarily applied to improve regression testing efforts of white box, code-level test cases. Here we are performing testing by using activity diagram for ATM machine.

II. LITERATURE REVIEW

Impact of coupling on object oriented techniques by using the approach of activity diagram for the generation of test cases. Ordering of test cases is to be done so that maintenance cost can be reduced and resources can be utilized in a proper manner. For considering inheritance hierarchy, the relation between classes at different levels has been considered and testing effort has been calculated for each class at a particular level. For minimization of test cases, impact of coupling over coverage per unit time is considered.

Coupling based testing is a data flow based technique which represents state space interactions between classes and objects. Static and dynamic analysis of the programs and highlights the areas to be tested, for methods under test. Which helps the developers in analyzing and understanding the critical interactions among the modules. Every coupling sequence has an associated set of coupling variables and coupling paths. The power that inheritance and polymorphism concept brings to the expressiveness of programming languages also brings a number of new anomalies and fault types. Offutt et al. have presented a fault model for object oriented programs and discussed specific categories of inheritance and polymorphic faults.[2] [2] This paper have introduced new data flow analysis techniques for object-oriented (OO) software, new testing criteria to address problems that can arise from using inheritance, dynamic binding, and polymorphism, and results from an experimental validation of the techniques. The techniques are based on the previous work for procedure-oriented software called coupling-based testing (CBT).The traditional notion of software coupling has been updated to apply to OO software, handling then relationships of aggregation, inheritance, dynamic binding, and polymorphism. This allows the introduction of a new integration analysis and testing technique for dataflow interactions within OO software, called OO coupling-based testing (OOCBT). This paper also presented a set of test-adequacy criteria that take inheritance, dynamic binding, and polymorphism into account.

[1] Software testing is an essential and integral part of the software development process. The testing effort is divided into three parts: test case generation, test execution, and test evaluation. Test case generation is the core of any testing process and automating it saves much time and effort as well as reduces the number of errors and faults. This paper proposes an automated approach for generating test cases from one of the most famed UML diagrams which is the activity diagram. The proposed model introduces an algorithm that automatically creates a table called Activity Dependency Table (ADT), and then uses it to create a directed graph called Activity Dependency Graph (ADG). The ADT is constructed in a detailed form that makes the generated ADG covers all the functionalities in the activity diagram. Finally the ADG with the ADT are used to generate the final test cases. The proposed model includes validation of the generated test cases during the generation process to ensure their coverage and efficiency. The generated test cases meet a hybrid coverage criterion in addition to their form which enables using them in system, regression as well as integration testing. The proposed model saves time and effort besides, increases the quality of generated test cases. The model is implemented on three different systems and evaluated to show its effectiveness.[1]

Motivation

1.1.1. Inter-dependence of lower levels of inheritance hierarchy on the upper levels after change in any class was not considered earlier.

Current approach presented focus on minimizing of coupling impact from faults on object oriented design by controlling their propagation via coupling. Best test case can be assessed so that unnecessary coupling can be avoided to come up with a better design. A detailed case study of Automated Teller Machine (ATM) has been carried out to assess the effectiveness of proposed approach over the other existing one.

1.1.2 To know how the impact of change t one level can affect the other classes

1.1.3. Every test case detects some fault that faults are new or detected earlier, if new then how critical is that.

Consider all the test cases of a class and calculate number of faults detected per unit time then select the first test case and then calculate new faults per unit time of each test case and select the best one. New fault means which are not discovered by selected test cases. Keep repeating this process until hundred percent faults are detected.

Objectives

In the light of above discussion, the objectives of the thesis are:

1. To minimize the coupling affect by selecting an affective test suite reducing the number of test cases and also utilize the limited resources in such a way so that the cost, time, man power can be used in an efficient manner.
2. To perform the two test case generation and test case minimization by selecting the test paths which will be having least affects of coupling .

1st level is generating a class diagram and an activity diagram for the ATM case study which will involves all the possible paths that can affects badly to the Class which should be selected on the basis of number of descendents, number of inherited attributes and level of class in the inheritance hierarchy.
2nd level is TEST CASE generation from activity diagram, that means test cases corresponding to the badly affected Class would be coupled on the basis of Fault coverage per unit time. Keeping in view of above objectives, the work has been done for designing an algorithm that could be used for generating the test cases of the affected module where the effect of any module change has been propagated then there minimization and extracting the suitable test paths.
3rd level is working on the test path that covers all the possible linking from start to end.

Testing process

Identify and modify/remove the obsolete test cases from T if specifications have changed.
Test case generation problem and selecting the effective path from activity diagram
Select T’⊂T, a set of test cases to execute on P’
test selection problem
Test P’ with T’, establishing P’’s correctness w.r.t. T’
Test suite execution problem

If necessary, create T’’, a set of new functional or structural test cases for P’
Coverage identification problem
Test P’ with T’’, establishing P’’s correctness w.r.t. T’’
Test suite execution problem
Create T’’’, a new test suite and test execution profile for P’, from T, T’, and T’’.
*Test suite maintenance problem and fault cover*age problem.

Methodologies

To support this process of coupling, developers often create an initial test suite, and then minimize it for testing. The simplest regression testing strategy, *retest all*, reruns every test case in the initial test suite. This approach, however, can be prohibitively expensive rerunning all test cases in the test suite may require an unacceptable amount of time.

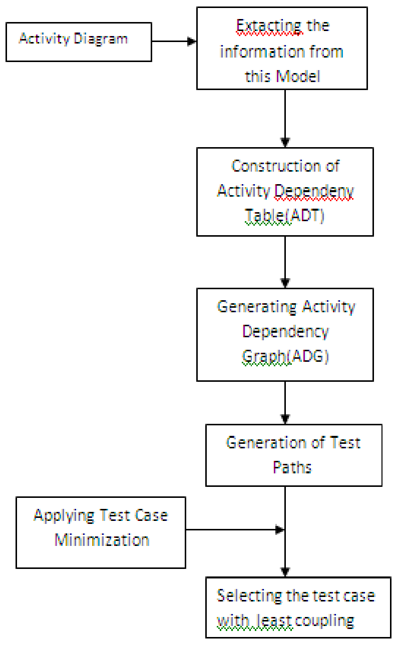

Activity Diagram
Extacting the information from this Model
Construction of Activity Dependeny Table(ADT)
Generating Activity Dependency Graph(ADG)
Generation of Test Paths
Applying Test Case Minimization
Selecting the test case with least coupling

Fig:1 Methodology used for the proposed work

1) Test case generation technique –**“following the proper algorithm for”**
2) test case selection techniques- “**Screen**”
3) Test case minimization techniques- “**Order and selecting”**
4) Test suite reduction techniques- “**Remove**” (minimizing testing paths for the sake of costs (by reducing the test suite) by permanently eliminating redundant test cases from test suites in terms of code or functionalities exercised and selecting the path that cover the main path efficiently)

A. 2.1.2.1 Coupling Test Selection (RTS)

Select T’ to be a subset of T and use this test suite for testing purpose

(A) Minimisation Techniques. Minimization-based test selection techniques attempt to select minimal sets of test cases from T that yield coverage of modified or affected portions of P.

(B) *Dataflow Techniques*. Dataflow-coverage-based test selection techniques select test cases that exercise data interactions that have been affected by modifications.

Pros:

1. Test fixed bugs immediately.
2. Can check side effects of fix (depends on the how broad is the test).
3. Reduce time of rerunning the tests.

Cons: 1.Require development time.

B. 2.1.2.2 Test Suite Reduction (TSR)

This technique uses information about program and test suite to remove the test cases, which have become redundant with time, as new functionality is added. It is different from test selection as former does not permanently remove test cases but selects those that are required

C. 2.1.2.3 Test Case minimization (TCM)

Test case ***minimization*** techniques selects arrange test cases so that those test cases that are better at achieving the testing objectives are run earlier in the testing cycle. For instance, software engineers might want to schedule test cases so that code coverage is achieved as quickly as possible or increase the possibility of fault detection early on in the testing. Studies have shown that some simple test case methods can remarkably improve testing performance, especially the rates at which test suites detect faults. The improved rates of fault detection can provide early feedback on the software being tested.

T1: Total statement coverage . It is possible to measure the coverage of statements in a program by its test cases. Then the test cases can be prioritized in terms of the total number of statements they cover by sorting them in the order of coverage achieved.

T2: Additional statement coverage. This is like T1 but it relies on feedback on the coverage achieved so far in testing so that it then focuses on statements not yet covered. For illustration in Program, both T1 and T2 first choose test case 3 as it covers most of the statements of procedure P. Then T1 selects test case 1 as it covers more statements, but here T2 does not select test case 1 as all of its statements have already been covered, instead it chooses test case 2. After this T1 selects test case 2, but T2 finishes as it does not select test case 1 for its statements have been covered as previously stated.

Experimental evaluation

Module1-

Here the proposed model will works over activity diagram for the ATM Working. In which the user selects the choice that she/he wish to. Then , the system checks whether the choice selected is valid or not, in case it is valid, the system will retrieve the balance of the user then check whether the balance is sufficient to withdraw from or not; otherwise, the system displays an error message for the user to enter an appropriate value. In case the balance is sufficient the system completes the withdraw process successfully by updating the balance, dispensing the cash and finally printing a receipt for the user. In the same way other choices will work for the account entity of the client using the ATM. The input and output of each activity are shown using activity parameter nodes

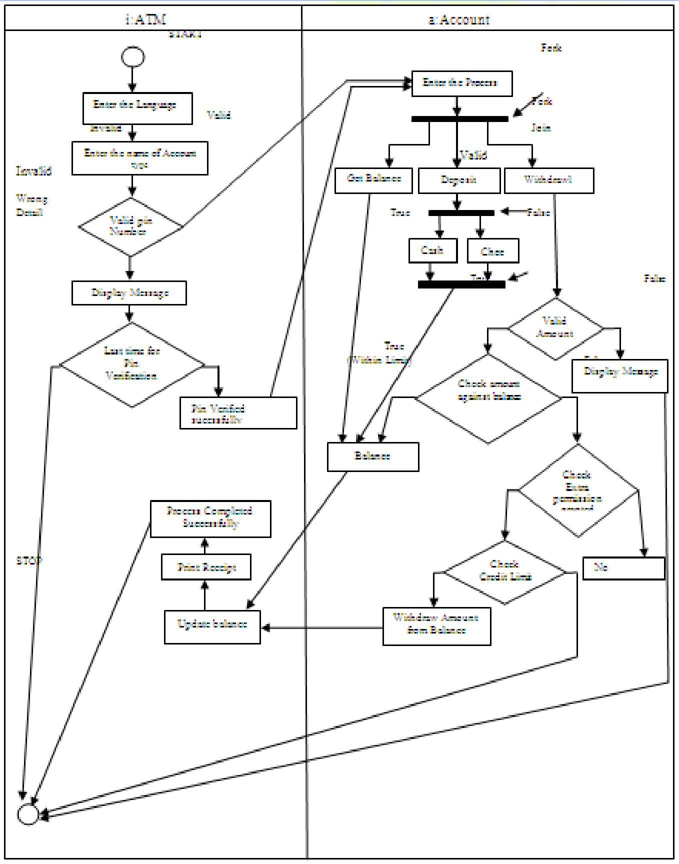


Fig2: Activity diagram for ATM machine

Module 2

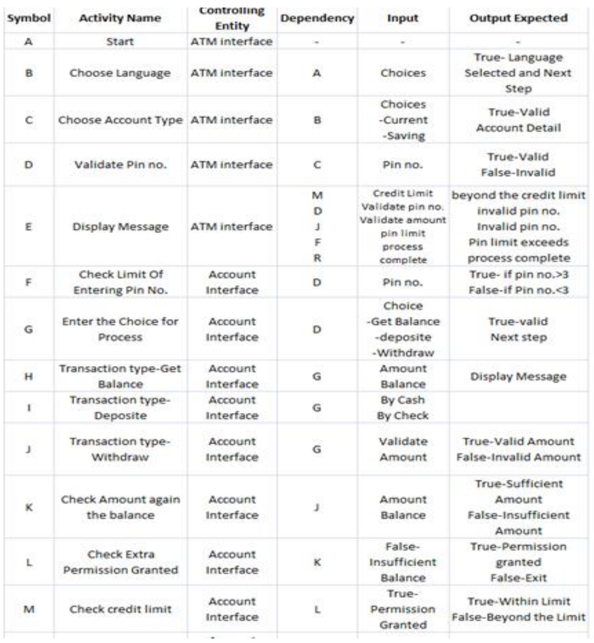
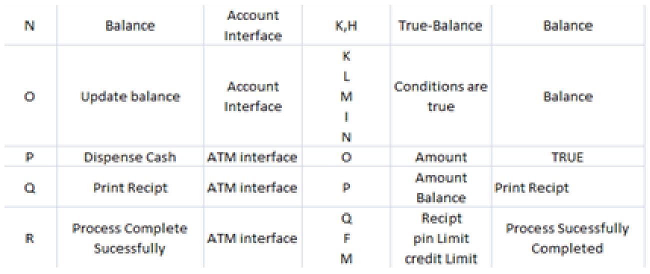
Algorithm for Generating the Test Case

1. Draw the activity diagram.
2. From the activity diagram, generate the Activity Dependency Table (ADT).
3. From the ADT, to generate Activity Dependency Graph(ADG).
4. Pass on ADG and get all the possible test paths using the depth first search technique.

5. Create a table TC with six columns (Test case Number, Test path Node, Node Input, Node Expected Output, Test case Input, and Test case Expected Output) to be filled with the final test cases.
6. Initialize counters to be used for indexing the list of nodes in each path and for indexing the table of final test cases TC
7. For each test path fetch the input and the expected output for each node from the corresponding ADT and add it to the corresponding cells (Node Input, Node Expected Output) in a new test case row.
8. Add to this new test case row from the ADT, the input of the first node in the current test path to the "Test case Input" cell and add the output of the final node in the current test path to the "Test case Expected Output" cell, then add the test case row to the TC.
9. Return the TC table after all the paths are being updated and added in TC.
10. Draw the cyclomatic diagram for the generated test paths.
11. Minimized the test paths that are generated.
12. End the system [10].

Each step in this algorithm is used to generate the Activity Dependency Table (ADT) with all activities which involved decisions, loops and synchronization along with the entity activity. The goal of this step to showing the activities that helps in dealing with every control with other entities which can be useful for system for testing. In which it includes the input and the expected output values for each activity with their Dependencies of each activity on others. Symbols for each activity gives an ease for referencing it in determining dependences and using it in the other involved modules.

Fig 3: Activity dependency table

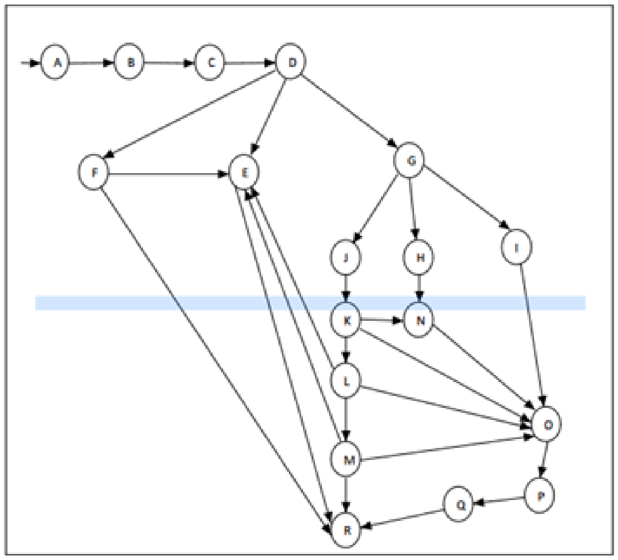
Symbol	Activity Name	Controlling Entity	Dependency	Input	Output Expected
A	Start	ATM interface	-	-	-
B	Choose Language	ATM interface	A	Choices	True- Language Selected and Next Step
C	Choose Account Type	ATM interface	B	Choices -Current -Saving	True-Valid Account Detail
D	Validate Pin no.	ATM interface	C	Pin no.	True-Valid False-Invalid
E	Display Message	ATM interface	M D J F R	Credit Limit Validate pin no. Validate amount pin limit process complete	beyond the credit limit invalid pin no. Invalid pin no. Pin limit exceeds process complete
F	Check Limit Of Entering Pin No.	Account Interface	D	Pin no.	True- if pin no.>3 False-if Pin no.<3
G	Enter the Choice for Process	Account Interface	D	Choice -Get Balance -deposite -Withdraw	True-valid Next step
H	Transaction type-Get Balance	Account Interface	G	Amount Balance	Display Message
I	Transaction type-Deposite	Account Interface	G	By Cash By Check	
J	Transaction type-Withdraw	Account Interface	G	Validate Amount	True-Valid Amount False-Invalid Amount
K	Check Amount again the balance	Account Interface	J	Amount Balance	True-Sufficient Amount False-Insufficient Amount
L	Check Extra Permission Granted	Account Interface	K	False-Insufficient Balance	True-Permission granted False-Exit
M	Check credit limit	Account Interface	L	True-Permission Granted	True-Within Limit False-Beyond the Limit
N	Balance	Account Interface	K,H	True-Balance	Balance
O	Update balance	Account Interface	K L M I N	Conditions are true	Balance
P	Dispense Cash	ATM interface	O	Amount	TRUE
Q	Print Recipt	ATM interface	P	Amount Balance	Print Recipt
R	Process Complete Sucessfully	ATM interface	Q F M	Recipt pin Limit credit Limit	Process Sucessfully Completed

Module 3-

With generation of ADT it will automatically generate the Activity Dependency Graph (ADG). By introducing only one no matter how many times they are used in the activity diagram. This will decrease the search space in the ADG. edged represents the transitions from one activity to another in the ADG.

The presence of an edge determines by checking the dependency column for the current node's symbol. Synchronization, decisions and loops are demonstrated using edges as well

Fig4 Activity dependency graph



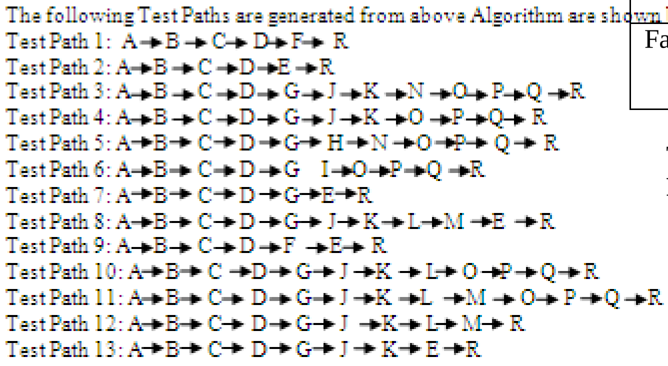
Module 4

The graph helps in determining the test paths. The number of test path is verified by Cyclomatic Complexity, V (CAG) for the flow graph CAG, is defined as:

V (CAG) = E – N + 2

Where E, is the number of edges; N is the number of nodes

Here E- 29 and N is 18 and after calculating them we got 29-18+2 which results with 13paths


The following Test Paths are generated from above Algorithm are shown below:

Test Path 1: A→B→C→D→F→ R
Test Path 2: A→B→C→D→E→R
Test Path 3: A→B→C→D→G→J→K→N→O→P→Q→R
Test Path 4: A→B→C→D→G→J→K→O→P→Q→ R
Test Path 5: A→B→ C→D→G→ H→N→O→P→ Q→ R
Test Path 6: A→B→ C→D→G I→O→P→Q→R
Test Path 7: A→B→ C→D→G→E→R
Test Path 8: A→B→ C→D→G→ J→K→L→M→E →R
Test Path 9: A→B→ C→D→F →E→ R
Test Path 10: A→B→ C →D→ G→ J →K → L→ O→P→Q→ R
Test Path 11: A→B→ C→ D→G→J →K →L →M → O→ P→Q→R
Test Path 12: A→B→ C→ D→G→J →K→L→M→ R
Test Path 13: A→B→ C→ D→G→J → K→E→R

Test Cases of Class STUDY

On the bases of independent path test cases are designed. Test cases are shown in table

Test Cases

Serial number	Test case
1	ABCDG HNOPQR
2	ABCDGJ KNOPQ R
3	AB CD G J K LM ER
4	AB CDG J KLM O P Q R
5	A B CD G J KL M R

Fault can be detected in class STUDY:

Fault1:- at node D, in definition of function
Fault2:- At E node, checking condition
Fault3:-at node J, switch statement
Fault4:-at node K
Fault5:-at node L
Fault6:-at node M
Fault7:-at node R
Each fault is assigned a weight as shown in table 4.3

general fault weight table

Type of fault	**Fault weight**
Type mismatch of arguments in function	**2**
Check condition in if block	**2**
Fault in Statements inside if block	**1**
Fault in switch statement	**2**

The faults of class study are assigned weight as shown in **Fault Weight**

Fault Number	Fault Weight
1	2
2	2
3	2
4	1
5	1
6	1
7	1

Test cases and fault of class STUDY

Random Test Suite and Fault Table

	T1	T2	T3	T4	T5	T6
F1(2)	*	*	*	*	*	*
F2(2)	*	*	*	*	*	*
F3(2)	*	*	*	*	*	*
F4(1)	*	*				
F5(1)			*	*		
F6(1)					*	*
F7(1)	*	*	*	*	*	*
total fault	8	8	8	8	8	8
time taken	5	7	11	4	10	12

AFD Result of Test Suite before minimization:-
TFi= ith fault is detected by which test case.
N=total number of test cases

M=total number of fault

$$APFD = 1 - \frac{TF1+TF2+\ldots\ldots\ldots\ldots+TFm}{n*m} + \frac{1}{2n}$$

References

[1] 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
[2] Mr. Kailash Patidar, Prof. Ravindra Kumar Gupta, Prof. Gajendra Singh Chandel, "Coupling and Cohesion Measures in Object Oriented Programming", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 3, March 2013.
3. A.V.K. Shanthi, G. Mohan Kumar. "A Heuristic Technique for Automated Test Cases Generation from UML Activity Diagram", Journal of Computer Science and Applications. ISSN 2231-1270 Volume 4, Number 2 (2012), pp. 75-86..........3
4. V. S. Bidve and Akhil Khare ," A Survey Of Coupling Measurement In Object Oriented Systems", International Journal of Advances in Engineering & Technology, Jan 2012,18
5. A. Aloysius, L. Arockiam, "Coupling Complexity Metric: A Cognitive Approach", I.J. Information Technology and Computer Science, 2012, 9, 29-35.
6. A. Agrawal and R. A.Khan, "Role of Coupling in Vulnerability Propagation", Software engineering : an international Journal (SeiJ), Vol. 2, no. 1, March 2012.
7. V. S. Bidve 1 , Akhil Khare. "A SURVEY OF COUPLING MEASUREMENT IN OBJECT ORIENTED SYSTEMS", information Technology Department, M.Tech. (II), BVCOE, Pune, International Journal of Advances in Engineering & Technology, Jan 2012.
8. Vipin Saxena, Santosh Kumar , "Impact of Coupling and Cohesion in Object-Oriented Technology", Babasaheb Bhimrao Ambedkar University, Lucknow, India. accepted August 15th, 2012
9. Bixin Li, Xiaobing Sun Hareton Leung and Sai Zhang, " Code-Based Change Impact Analysis Techniques Software Testing, Verification And Reliability", Softw. Test. Verif. Reliab. (2012)
10. 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:
11.Mr. Kailash Patidar, Prof. Ravindra Kumar Gupta, Prof. Gajendra Singh Chandel,"Coupling and Cohesion Measures in Object Oriented Programming", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 3, March 2013.
12. Roger T. Alexander, Jeff Offutt and Andreas Stefik, "Testing coupling relationships in object-oriented programs" Softw. Test. Verif. Reliab. 2010; 20:291–327 Published
online 21 January 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/stvr.417
13. Baikuntha Narayan Biswal. "Test Case Generation and Optimization of Object-Oriented Software using UML Behavioral Models", Department of Computer Science and Engineering National Institute of Technology, Rourkela Rourkela-769 008, Orissa, India, July, 2010
14. Debasish Kundu, Debasis Samanta: "A Novel Approach to Generate Test Cases from UML Activity Diagrams", in Journal of Object Technology, vol. 8, no. 3, May–June 2009, pp. 65–83, http://www.jot.fm/issues/issue 2009 05/article1/
15. Aynur Abdurazik and Jeff Offutt. "Using coupling-based weights for the class integration
and test order problem".,The Computer Journal, pages 1,14, August 2007
16. Aynur Abdurazik and Jeff Offutt. "Coupling-based class integration and test order" . In Workshop on Automation of Software Test (AST 2006), pages 50{56, Shanghai,China, May 2006.