A Fault – Severity based Regression Test Case Prioritization Technique for Object Oriented Software

Vedpal
Department of Computer Engineering
YMCA University of Science and Technology, Faridabad, India
ved_ymca@yahoo.co.in

Naresh Chauhan
Department of Computer Engineering
YMCA University of Science and Technology, Faridabad, India
nareshchauhan19@gmail.com

Harish Kumar
Department of Computer Engineering
YMCA University of Science and Technology, Faridabad, India
htanwer@gmail.com

Abstract - Presently due to requirement and expectation of the customer the software are usually updated by modifying the exiting software component. The software industries always updated existing software component to develop new software or enhance the exiting software. There are many constraints on software industries such as to deliver the software, fulfill the customer expectation with in time and allocated budget. Every type of modification in the software new test case are added in addition to the existing test cases, resultant to execute to all the test cases increase the testing cost, waste of time and resources. So therefore there is requirement of a cost effective regression test case prioritization technique to prioritize the test cases. In this paper a hierarchical regression test case prioritization technique for object oriented software is presented. The presented approach works a two levels. At the first level the classes are prioritized by computing the effort to test the class. The computing of testing efforts are performed on the basis of structural factors which affect the testing of the software. At the second level the test case are ordered of the prioritized classes obtained from the earlier phase of the technique. For experimental verification and analysis the proposed approach has been applied on a software module implemented in C++. To show the effectiveness of the proposed approach the results are compared with the random approach. The comparison result shows the efficacy of the proposed technique.

Keywords: Regression testing, test case prioritization, object oriented testing,tcpooc

I. INTRODUCTION

Software testing is a strenuous and expensive process. Research has shown [12] that approximate 50% of the total budgeted software cost is comprised of testing activities. Companies are often faced with lack of time and resources, which limits their ability to effectively complete testing efforts.

Regression testing is the re-execution of test cases that has already been executed. In regression testing as testing proceeds, the regression test suits increases and it is unreasonable and time consuming to re execute every test for every software function when there is a change. The detection of regression testing faults and fixing is very costly testing practice. It is observed that software get modified at last minute and testing team is asked to check changes in the code just before making a release of the software, in this state of affairs the testing team needs to perform the testing of affected areas only. To make regression 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.

Test case prioritization techniques arrange test cases so that those test cases that are better at achieving the testing objectives are run earlier in the regression 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 in the testing. The improved rates of fault detection can provide early feedback on the software being tested. This reduces regression testing costs by allowing the software engineers to tackle the discovered faults and begin debugging earlier in testing.

Object-orientation has rapidly become accepted as the preferred paradigm for large-scale software design. The product created during Software Development effort has to be tested since bugs may get introduced during its development. Software based on Object Oriented technology poses challenges to conventional testing techniques since it involves concept like Inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This provides an opportunity to reuse the code functionality and fast implementation time, but inheritance hierarchy also poses some challenges while prioritizing the test cases. In inheritance hierarchy, the derived classes are dependent on base classes due to inherent relationships, so the probability of error propagation is higher in the inheritance hierarchy if base classes propagate errors through it.

In this paper a hierarchical test case prioritization is proposed wherein the prioritization process is performed at two levels given below:

(1) The classes are first prioritized on the bases of number of inherited attributes/methods, number of descendents and level of class in the inheritance hierarchy.

(2) The test cases of the highest prioritized class are then put in order on the bases of fault coverage.

II. RELATED WORK

This section is about review of some related topics of work to be carried out.

R. Harrison, S. Counsell, and R. Nithi [7] described an empirical investigation into the modifiability and understandability of object-oriented (OO) software. They conducted a controlled experiment to establish the affects of various levels of inheritance on modifiability and understandability . Results indicated that the systems without inheritance were easier to modify and understand than the systems containing three or five levels of inheritance.

Chabbi Rani Panigrahi and Rajib Mall [9] presented model based approach for prioritizing the regression test cases. The proposed model represents all features of object oriented language like inheritance, polymorphism, association, aggregation etc. They also considered the dependencies among test cases.

John Daly, Andrew Brooks, James Miller, Marc Roper and Murray Wood [6] performed experiment and collected data for test the effect of inheritance depth on maintainability of object oriented software. The collected data showed that maintaining task for the object oriented software with the three levels of inheritance depth is quicker than maintaining the equivalent object oriented software with no inheritance.

David C. Kung, jerry Gao and Pei Hsia [1] proposed an algorithm for generating order of tests of affected classes. They used an object relation graph which described all the relation existed in the object oriented program such as inheritance, aggregation, association etc.

Adam Smith, Joshua Geiger and Mary Lou Soffa [10] proposed a tool for reducing and prioritizing test cases. The proposed tool creates a tree based model of program behaviour and by using these trees the test suite are reduced and reordered.

Gagandeep Makkar , [5] Jitender kumar Chhabra and Rama Krishana Challa presented inheritance metric based on reusability of UML software design. They consider the number of inherited attribute and depth level of classes. If the depth of inherited class exceeds the fourth level then the proposed metric imposes the penalty factor.

Arti Chhikara, R.S. Chillar and Sujata Khatri[2] presented the assessment of effect of the inheritance on the object oriented Systems. Their assessment showed that inheritance is a key factor of object oriented Systems.

Nasib S. Gill and Sunil Sikka [8] characterizes metrics of reuse and reusability in object oriented software development . They proposed five new metrics. These matrices are Breadth of inheritance tree, Method reuse per inheritance relation , Generality of inheritance class, Reuse probability and attribute reuse per inheritance relation.

Muhammad Rabee Shaheen and Lydie du Bousquet[3] finds that cost of testing is influenced by Depth of inheritance tree. No of methods to test in each class is related to the depth of inheritance tree.

Pranshu Gupta et.all [4] analyzed hierarchy of test order of classes to find where the faults are concentrated in that hierarchy using CDM. They found that approach for ordering the test cases depend on different categories of fault.

A critical review of the work done by the researchers in the direction of test case prioritization indicates that Dependency between the classes and efficiency of test case to detect the errors have not been considered that may affect the test case execution.

III. PROPOSED WORK

The proposed work includes two level prioritization, in which the first level prioritization involves prioritizing the classes of inheritance hierarchy whereas the second level prioritization involves prioritizing the test cases of each class.

A technique is proposed to order the affected classes intended to find faults quickly**.** The probability of error propagation in inheritance hierarchy depends on the number of inherited attributes/methods, level of class in inheritance hierarchy and the number of descendent classes. So, the first level prioritization involves prioritizing the classes depending on the number of descendents of that class, number of inherited attributes/methods and level of the class in inheritance hierarchy.

If number of levels are less than or equal to 3, the testing effort can be calculated as:

Testing effort = (number of descendents + number of inherited attributes/methods) * (4 -level)

If number of levels are greater than 3, the testing effort can be calculated as:

Testing effort = (number of descendents + number of inherited attributes/methods) * (level - 3)

The base class at level 1 of inheritance hierarchy is always assigned highest priority. If any error gets propagated from this class, it will affect the entire hierarchy, because all the classes below this level will inherit the properties of base class. The second level of prioritization is the ordering of test cases of each selected class and is done by technique of fault coverage per unit time taken. The test case is stored with time taken by it and number of faults detected by it and each fault is assigned a weight on the basis of its criticality.

A. First Level Prioritization

The first level prioritization technique prioritizes the classes of an object oriented software using inheritance hierarchy. In inheritance hierarchy, the classes at lower level inherit the properties of classes at upper level. Hence, the derived classes are dependent on the base classes, this dependency increases the probability of error propagation through the inheritance hierarchy. Hence the classes should be tested in such order that the classes with higher probability of error propagation get tested first.

The base class should be assigned the highest priority because if any error gets propagated from this class, it will affect the entire hierarchy. So the classes should be ordered in such a way that error propagation can be minimized.

The classes at lower level are assigned priority based on the level of class in inheritance hierarchy, number of inherited attributes and number of descendent classes. The technique has been described with the help of a block diagram shown in figure 1.

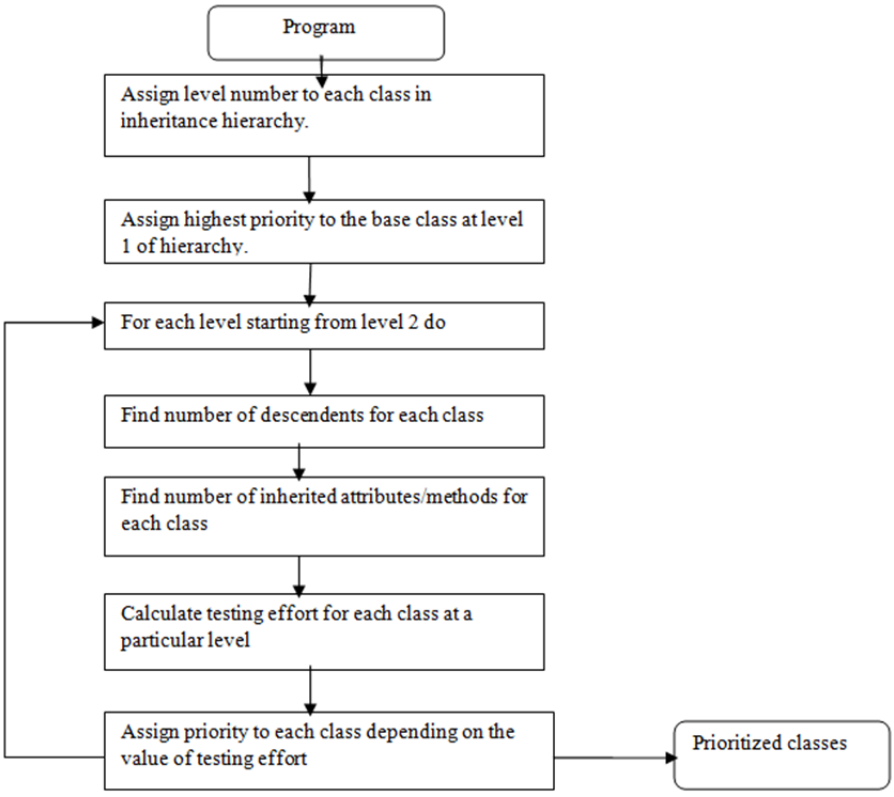

Fig. 1 Block Diagram of 1st Level Prioritization

An algorithm has been proposed for prioritizing the classes of an object oriented software using inheritance hierarchy. The classes of inheritance hierarchy has inherent complex relationships due to the dependency of derived classes over subclasses. This algorithm prioritizes the classes in such a way so that faults could be found earlier and the probability of error propagation through the inheritance hierarchy could be minimized. The algorithm 1 describes the technique used for first level prioritization:

Algorithm 1. First Level Prioritization

First_ level_ prioritization (P, n)

Where P is complete program and n is the number of levels in inheritance hierarchy.
Begin
1. Assign level number to each class in the inheritance hierarchy.
2. Assign highest priority to the base class at level one of the hierarchy.
3. For (level=2; level<=n; level++)
a) Find number of descendents for each class.
b) Find number of inherited attributes/methods for each class.
c) If no of levels is less than or equal to 3,then
Testing effort = (no. of descendents + no of inherited attributes/methods) * (4 - level)
Else
Testing effort = (no. of descendents + no of inherited attributes/methods) * (level - 3)
d) Assign priority to each class depending on the value of testing effort.
(highest testing effort value gets the highest priority)
end

B. Second Level Prioritization

The classes prioritized using first level prioritization are input to the second level prioritization where the test cases of each individual classes are prioritized. The test cases are prioritized based on fault weight and fault coverage in [11] per unit time.

The test cases that detect faults which have not been discovered earlier and are more critical are prioritized first.

The technique proposed for second level prioritization has been explained using block diagram shown in figure 2.

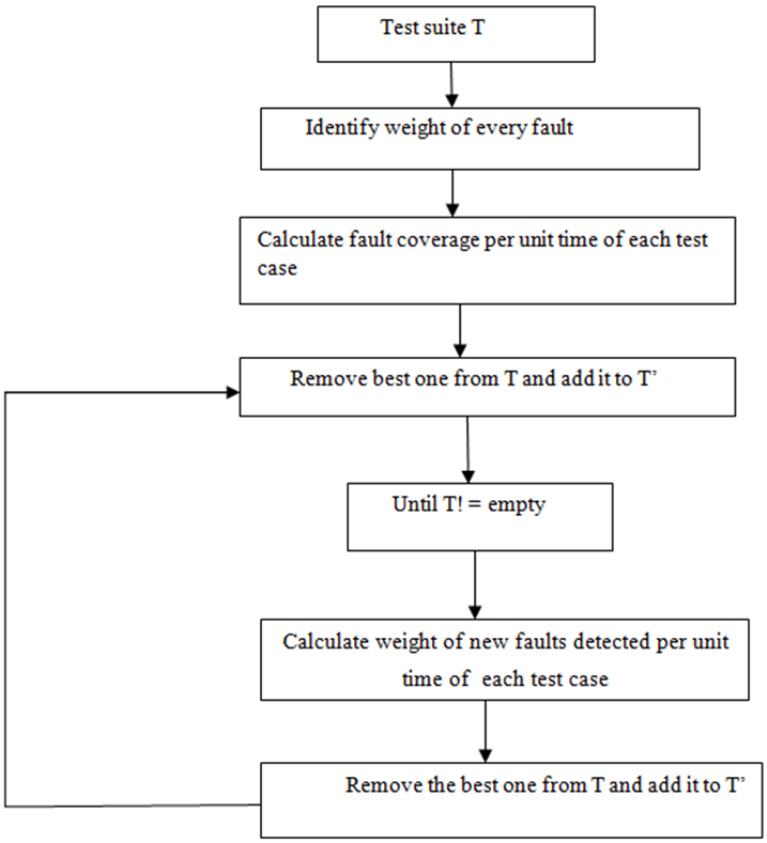

Fig 2. Block Diagram of Second Level Prioritization

C. Algorithm of Second Level Prioritization

The proposed technique order the test cases on the bases of fault coverage. Every test case detects some fault. And faults are assigned weight on the basis of [13] criticality, and fault can be new or detected earlier, so the test cases which are detecting critical and new faults (which are not detected earlier) are prioritized over other test cases. The algorithm 2 explains the second level prioritization used for ordering the test case of each particular class of inheritance hierarchy.

Algorithm 2 Second Level Prioritization

PRIORITIZATION OF TEST CASES OF PRIORITIZED CLASS

//there are M test cases and N faults and each fault is assigned some weight.

Begin

1. T is original test suite, T' is prioritized test suite
2. Calculate fault_weight per unit time by each test case.
3. Arrange them in decreasing order.
4. Remove the best one from T and add it to T'.
5. Repeat step 6 and 7 until T is not empty.
6. Calculate weight of new faults detected per unit time of each test case.

// New Fault means those fault which are not detected by any test case in T'.

7. Remove the best one from T and add it to T'
8. Go to step 5.
9. Return T'.

End

D. Proposed Fault Table

Fault [13] can be categorized on the basis of severity, and faults are assigned a weight on the basis of criticality of the fault. A fault weight shows the capability of faults to affect the working of software. The assigned weights of faults are shown in Table 1

Table 1.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**
Fault in for loop	**3**
Fault in recursion	**1**
Fault in do while loop	**2**
Condition statement under condition statement	**4**
Loop under condition statement	**3**
Fault in nested loop	**4**
Lack of memory	**3**
Improper use of access specifier	**3**
External function not called properly	**2**
Improper Type casting	**3**
Exception handling problem	**2**
Method signature problem	**2**

IV. RESULT AND ANALYSIS

The proposed technique has been verified and analyzed by taking a case study. The considered case study consists of four classes, study, lec_time, sports_time and usetime.the class study inherits two classes, lec_time and sports_time and the lec_time further inherits usetime. The testing effort has been calculated by using number of descendents, number of inherited attributes/ methods and the level of a class in inheritance hierarchy.

A. Case study

The inheritance hierarchy shown in figure 3 has been used to analyze the proposed technique.

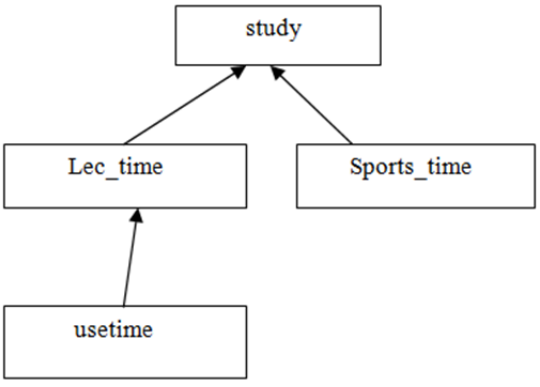

Fig. 3 Inheritance Hierarchy of Case Study

B. First level prioritization

There are four classes in the inheritance hierarchy of the program and they are interconnected. So at the time of regression testing badly affected class will be tested before less affected class. The Table 2 shows the calculation of testing effort for each class and the priority assigned to each class.

Table 2. Calculation of testing effort

Level1	**Level2**	**Level 2**	**level 3**
Study class has highest priority. Level=1 No of descendent classes=2 No of inherited attributes/methods=0 Testing effort=(2+0)*(4-1)=2*3=6	For class lec_time Level=2 No of descendent classes=1 No of inherited attributes/methods=1 Testing effort=(1+1)*(4-2)= 2*2= 4	For class sports_time Level=2 No of descendent classes=0 No of inherited attributes/methods=1 Testing effort=(1+0)*(4-2)=1*2=2 Now,assign priorities on the basis of testing effort values.	For class usetime Level=3 No of descendent classes=0 No of inherited attributes/methods=3 Testing effort=(3+0)*(4-3)=3*1=3

Table 3 shows the priority assigned to each class of inheritance hierarchy

Table 3 priority assigned to each class of inheritance hierarchy

Class name	**Priority number**
Study	1
Lec_time	2
Sports_time	3
Usetime	4

Lower number indicates higher priority**.**

C. Second level prioritization

The first level prioritization technique prioritizes the classes of inheritance hierarchy, and on the basis of output of first level prioritization, second level prioritization is done so that test cases of each class can be ordered. The test cases of each class in the inheritance heirarchy are to be prioritized on the basis of fault coverage by the test cases.

1). Flow graph of class study with labelled edges

The labelled flow graph of class study is shown in figure 3. The independent paths have been recognized using flow graph and the test cases has been designed using independent paths.

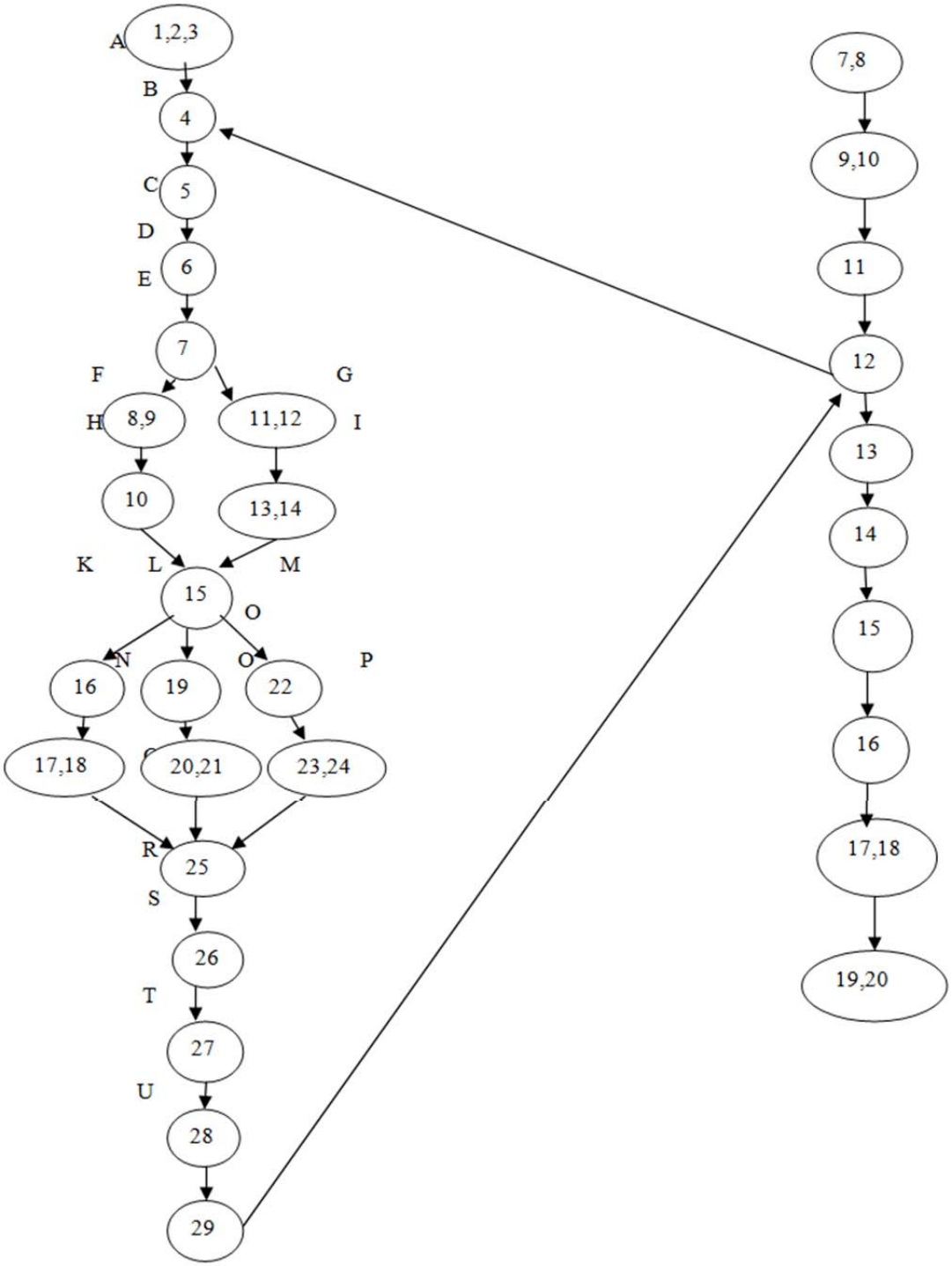

Fig. 4 Flow Graph of Class Study

2. Test Cases of Class STUDY

The Table 4. shows all the independent path which are tested by executing the test cases.

Table 4. Independent paths of class study

Serial number	Independent paths
1	ABCDEFHJKNQRSTU
2	ABCDEGIJKNQRSTU
3	ABCDEFHJLOQRSTU
4	ABCDEGIJLOQRSTU
5	ABCDEFHJMPQRSTU
6	ABCDEGIJMPQRSTU

The Table 5 shows the Faults Detected in class case study and assigned fault weight

Table 5. Fault Weight

S.No	**Fault Number**	**Detected Faults**	**Fault Weight**
1	Fault1	At node D, in definition of function	2
2	Fault2	At E node, checking condition	2
3	Fault3	At node J, switch statement	2
4	Fault4	At node K	1
5	Fault5	At node L	1
6	Fault6	At node M	1
7	Fault7	At node R	1

Test cases and fault of class STUDY are shown in table 6

Table 6. 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

APFD Result of Test Suite before Prioritization is 78%

3) Prioritization of test suite based on proposed algorithm

STEP1: VTI=FAULT/TIME [11] (RATE OF FAULT DETECTION)

VT1=8/5=1.60 VT2=8/7=1.14

VT3=8/11=0.72 VT4=8/4=2.0

VT5=8/10=0.80 VT6=8/12=0.66

STEP2: SORTING OF VTI

T4 T1 T2 T5 T3 T6

STEP3: REMOVE T4 FROM T AND ADD T4 TO T'
Now T' contain= T4.
T contain=T1,T2,T3,T5,T6
STEP 4: UNTIL T IS NOT EMPTY
STEP5: NEW FAULT COVERAGE OF TEST CASES PER UNIT TIME
VT1=1/5=0.20 VT2=1/7=0.14
VT3=0 VT5=1/10=0.10
VT6=1/12=0.08
STEP6: REMOVE T1 FROM T AND ADD TO T'
T contain=T2, T3, T5, T6
T' contain=T4, T1
STEP7: GO TO STEP 4
STEP4: UNTIL T IS NOT EMPTY
STEP5: NEW FAULT COVERAGE OF TEST CASES PER UNIT TIME
VT2=0.00 VT3=0
VT5=1/10=0.10 VT6=1/12=0.08
STEP6: REMOVE T1 FROM T AND ADD TO T'
T contain= T2, T3, T6
T' contain=T4, T1, T5
STEP7: GO TO STEP 4
STEP4: UNTIL T IS NOT EMPTY
STEP5: NEW FAULT COVERAGE OF TEST CASES PER UNIT TIME
VT2=0.00 VT3=0.00
VT6=1/12=0.00
STEP6: REMOVE T7 FROM T AND ADD TO T'
T' contain=T4, T1, T5, T2,T6,T3.
STEP8: RETURN T' WHICH IS PRIORITIZED TEST SUITE.
Prioritized test suite is shown in Table 7

Table 7 Prioritized Test Suite

	T1(T4)	T2(T1)	T3(T5)	T4(T2)	T5(T6)	T6(T3)
Fault 1	*	*	*	*	*	*
Fault 2	*	*	*	*	*	*
Fault 3	*	*	*	*	*	*
Fault 4		*		*		
Fault 5	*					*
Fault 6			*		*	
Fault 7	*	*	*	*	*	*

Apfd of Prioritized Test Suite is 85%
Fault percent detection corresponding to each test case of random and prioritized test suites are shown in table 8

Table 8. Test Suite and Fault % detected

Random test suite	Fault % detected	Prioritized test suite	Fault % detected
1	71.4	1	71.4
2	71.4	2	85.7
3	85.7	3	100
4	85.7	4	100
5	100	5	100
6	100	6	100

Based on the analysis it is found that the prioritized test suite is better as compare to random test suite. Using APFD metric comparison is shown in Table 9

Table 9 APFD Metric

Test Suite	**APFD (%)**
Random	78
Prioritized	85

Fault percent detected by test case of random and prioritized test suite is shown in Fig. 5 and Fig .6 respectively.

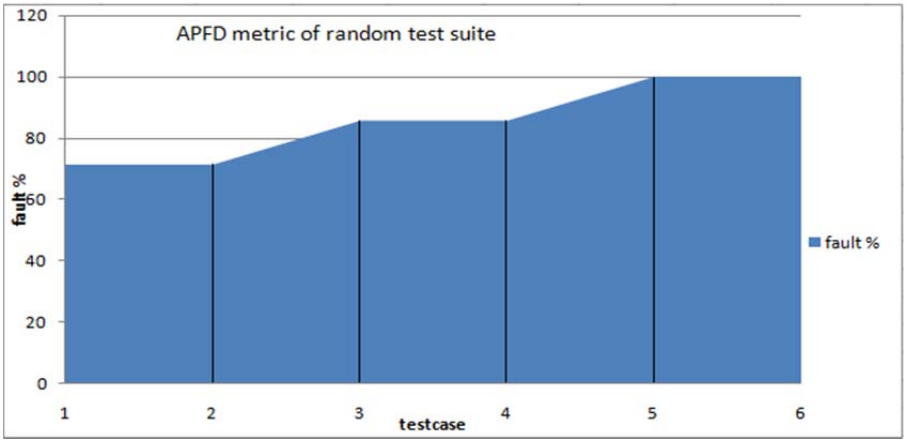

Fig 5. Fault percentage detected by random test suite

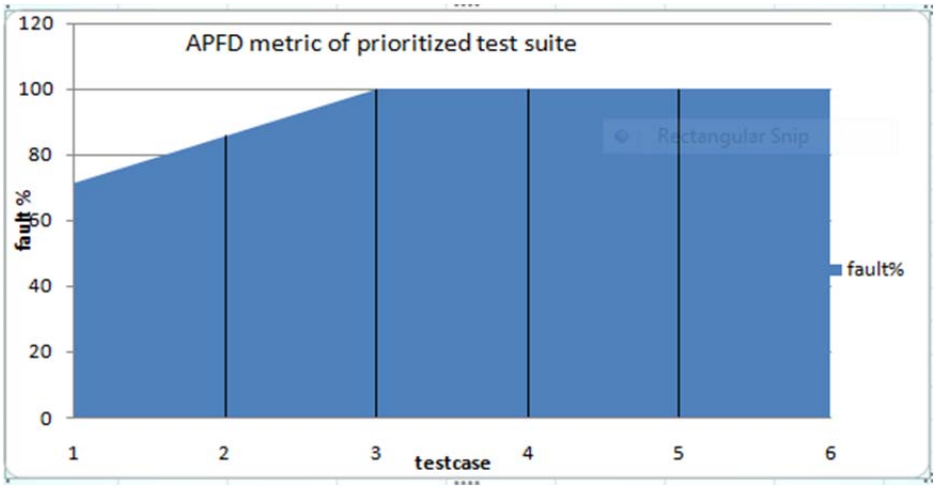

Fig 6. Fault percentage detected by prioritized test suite

As the simplicity of the approach it has been applied on other classes and result are discussed below:

4.) *Test case prioritization of class lec_time*

The test case and fault table of class lec_time are shown in table 10

Table 10 Test Case and Fault Table of Class lec_time

Fault name & weight	Test case 1	Test case 2	Test case 3
Fault 1 (2)	*	*	
Fault 2 (2)	*	*	
Fault 3 (1)	*		
Fault 4 (2)			*
Total fault weight	5	4	2
Time taken	6	2	3

The prioritization order of the test cases are TC1, TC3, TC2

5). Comparison of Prioritized and Non Prioritized Test Suite

The comparison of APFD of prioritized and non prioritized test suite is shown in Table 11

Table 11 APFD Metric

TEST SUITE	APFD
Random	66.67
Prioritized	75

Fault percent detection corresponding to each test case of random and prioritized test suites are shown in table 12

Table 12 Test Suite and Fault % detected

Random test suite	Fault % detected	Prioritized test suite	Fault % detected
1	75	1	75
2	75	2	100
3	100	3	100

Fault percent detected by test case of random and prioritized test suite is shown in Fig. 7 and Fig. 8 respectively.

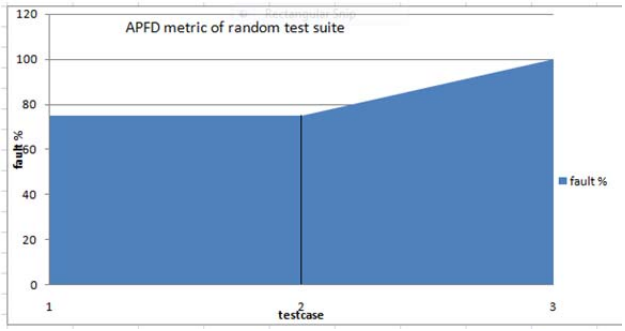

Fig. 7 Fault percent detected by random test suite

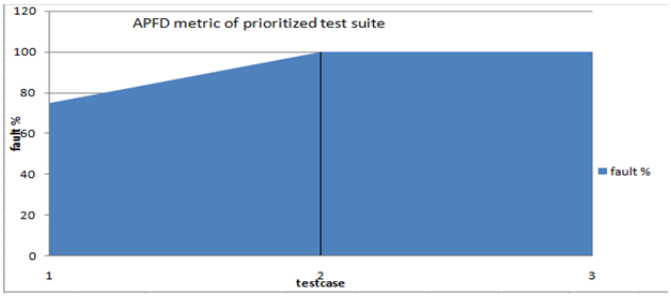

Fig. 8 Fault percent detected by prioritized test suite

6.) Test case prioritization of class sports_time

The test cases and faults of class sports_time are shown in table 13

Table 13 Test Case and Fault Table of Class sports_time

Fault name & weight	Test case 1	Test case 2	Test case 3
Fault 1 (2)	*	*	
Fault 2 (2)	*	*	
Fault 3 (1)	*		
Fault 4 (2)			*
Total fault weight	5	4	2
Time taken	6	2	3

The order of prioritized test cases is TC1, TC3, TC2 :

7) Comparison of Prioritized and Non Prioritized Test Suite

The comparison of APFD of prioritized and non prioritized test suite is shown in Table 14

Table 14 APFD Metric

TEST SUITE	APFD
Random	66.67
Prioritized	75

Fault percent detection corresponding to each test case of random and prioritized test suites are shown in table 15.

Table 15 Test Suite and Fault % detected

Random test suite	Fault % detected	Prioritized test suite	Fault % detected
1	75	1	75
2	75	2	100
3	100	3	100

Fault percent detected by test case of random and prioritized test suite is shown in Fig 9 and Fig 10 respectively.

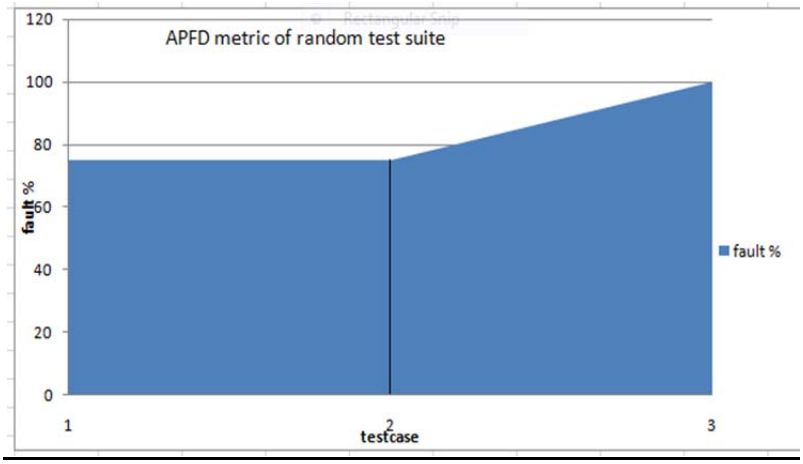

Fig 9 Fault percent detected by random test suite

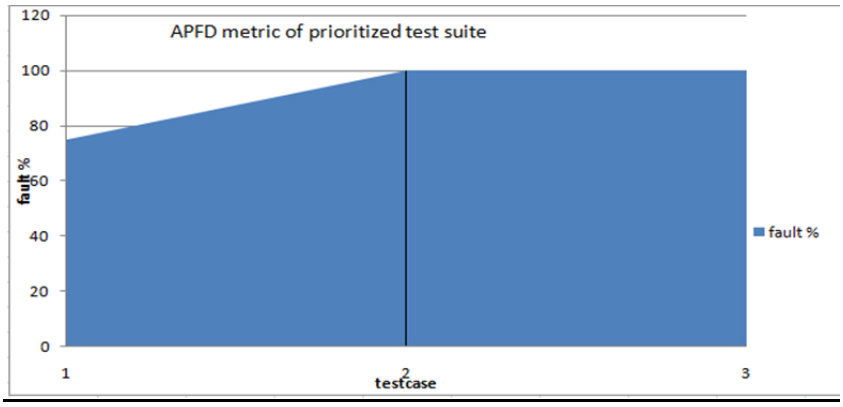

Fig 10 Fault percent detected by prioritized test suite

8.) Test Case Prioritization of Class usetime

On the basis of code coverage test cases are designed here. There is only one function in class usetime. Only one test case is enough to cover all statements of function sum in class usetime. So there is no need to prioritize test cases of class 3.

The prioritized order of test cases of the class study, lec_time, sport_time and use_time is shown in table 16.

Table 16 Result of Proposed Technique

Class study	TC4	TC1	TC5	TC2	TC6	TC3
Class lec_time	TC2	TC1	TC3			
Class sports_time	TC2	TC1	TC3			
Class usetime	TC1					

The experimented results of the proposed technique are analyzed by average percentage of fault detection metric. The result of proposed technique is shown in table 17.

Table 17 Analysis of APFD metric

Class name	**Random test suite**	**Prioritized test suite**
Study	78	85
Lec_time	66.67	75
Sports_time	66.67	85

The analysis shows that proposed technique is better as compared to random test case prioritization approach.

V. CONCLUSION

The proposed technique for "Regression Test Case Prioritization in object oriented program" using inheritance heirarchy and fault coverage is a beneficial technique for the purpose of saving resources such as time and cost. In this technique, the classes are prioritized based on the number of descendents, number of inherited attributes and level of class in the inheritance hierarchy, so that the classes which have high probability of error propagation in the inheritance hierarchy are prioritized first. It is proved to be effective because of two level prioritization, including identification of classes with higher degree of error propagation using first level prioritization technique and doing prioritization of test cases of affected class using fault coverage per unit time approach. The prioritization is better because the classes are prioritized in such a way that the classes with high degree of error propagation are prioritized first and the test cases with high rate of fault detection are prioritized first.. The experimental evaluation has also been performed using a case study (program written in C++). To illustrate the effectiveness of the proposed prioritization technique, Average percentage of fault detection (APFD) metric has been used. The analysis shows that proposed technique is better as compared to random test case prioritization approach.

References

[1] D. C. Kung , J. Gao and P. Hsia, 'Class, Firewall , Test Order, and regression testing of object oriented programs'. JOOP 8(2): 51 – 65 (1995)

[2] A. Chhikara, R. S. Chhillar and S. Khatri " Evaluating the impact of different types of inheritance on the object oriented software metrics" International Journal of Enterprise Computing and Business Systems ISSN (Online) : 223-8849 Volume 1 Issue 2 July 2011

[3] M. R. Shaheen and L. D. Bousquet "Relation between Depth of Inheritance Tree and Number of Methods to Test" 2008 International Conference on Software Testing, Verification, and Validation

[4] P. Gupta and D. A. Gustafson "ANALYSIS OF THE CLASS DEPENDENCY MODEL FOR OBJECT-ORIENTED FAULTS" International Journal of Advances in Engineering & Technology, May 2012. IJAET ISSN: 2231- 1963

[5] G. Makkar, J. K. Chhabra2 and R. K. Challa "Object Oriented Inheritance MetricReusability Perspective" 20 1 2 International Conference on Computing, Electronics and Electrical Technologies [ICCEET]

[6] J. Daly. A. Brooks, J. Miller, M. Roper and M. Wood "An Empirical Study Evaluating Depth of Inheritance on the Maintainability of Object Oriented Software"

[7] R. Harrison, S. Counsell, and R. Nithi"Experimental assessment of the effect of inheritance on the maintainability of object oriented system" http://citeseer.uark.edu

[8] N. S. Gill and S. Sikka "Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD" International Journal on Computer Science and Engineering (IJCSE)

[9] C. R. Panigrahi and R. Mall "Test case prioritization of object – oriented Programs" SETlabs Briefings Vol 9 No. 4 2011

[10] A. Smith, J. Geiger and M. L. Soffa "Test Suite Reduction and prioritization with call trees"

[11] P. R. Srivastava " Test case prioritization " Journal of theoretical and applied information technology 2008

[12] Sunita and M. Gulia " Study of regression test Selection Technique " International Journal of advanced research in computer science and software engineering" Volume 4, issue 2, 2014

[13] N. chauhan (2010) Software Testing principle and practices . Oxford University Press.

[14] A. Bakar Md sultan, A. A..A. Ghani, S. Baharom and S. Musa. (2014) An Evolutionary Regression test case prioritization based on dependence graph and genetic algorithm for object oriented programs . 2nd International conference on emerging trends in engineering and technology , May 30-31 , London(UK).

[15] A. A. Acharya, D. P. Mohapatra and N. Panda. (2010) Model based test case prioritization for testing component dependency in CBSD using UML sequence diagram. International journal of advanced computer science and applications, Vol. 1, No. 6

[16] E. ashraf , A. Rauf and K Mahmood.(2012) Value based regression test case prioritization. Proceedings of world congress on Engineering and Computer Science Vol. 1 San Francisco,USA

[17] M. Shahid & S. Ibrahim.(2014) A new code based Test case Prioritization technique. International Journal of software Engineering and its application Vol. 8, No 6 , PP. 31 – 38

[18] R. Kavitha & N. Suresh Kumar.(2010) Test case prioritization for regression testing based on severity of fault International journal of computer science and engineering vol. 02, No. 05 ,1462-1466

[19] R. Beena & S. Sarala.(2013) Code coverage based test case selection and prioritization. International journal of software engineering and applications (IJSEA), Vol.4, No. 6

[20] S. Khatri, R. S. Chhillar and A. Sangwan, " Analysis of Factors Affecting Testing in Object oriented systems" International Journal on Computer Science and Engineering (IJCSE)

[21] Vedpal, N. Chauhan and H. Kumar. " A hierarchical test case prioritization for object oriented software" International conference on contemporary computing and informatics(IC3I) ,2014