A Hierarchical Test Case Prioritization Technique for Object Oriented Software

Vedpal
Dept. of Comp. Engg.
YMCAUST
Faridabad India
ved_ymca@yahoo.co.in

Naresh Chauhan
Dept. of Comp. Engg.
YMCAUST
Faridabad, India
nareshchauhan19@gmail.com

Harish Kumar
Dept. of Comp. Engg.
YMCAUST
Faridabad, India
htanwar@gmail.com

***Abstract*: Software reuse is the use of existing artifacts to create new software. Inheritance is the foremost technique of reuse. But the inherent complexity due to inheritance hierarchy found in object – oriented paradigm also affect testing. Every time any change occurs in the software, new test cases are added in addition to the existing test suite. So there is need to conduct effective regression testing having less number of test cases to reduce cost and time. In this paper a hierarchical test case prioritization technique is proposed wherein various factors have been considered that affect error propagation in the inheritance. In this paper prioritization of test cases take place at two levels. In the first level the classes are prioritized and in the second level the test cases of prioritized classes are ordered. To show the effectiveness of proposed technique it was applied and analyze on a C++ program.**

Keywords: object oriented testing, test case prioritization, regression testing

I. INTRODUCTION

Test case prioritization (TCP) techniques arrange test cases so that the 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. Research has shown that at least 50% of the total 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. Prioritization of test cases in the order of execution in a test suite can be beneficial. Test case prioritization (TCP) involves the explicit prior 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.

There are many concepts in OO that gives rise to the dependency problem. The dependency problem is mainly due to inheritance, aggregation and association etc. Inheritance 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

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

978-1-4799-6629-5/14/$31.00 ©2014 IEEE

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 the following factors have not been considered that may affect the test case execution:

(1) **Inter-dependence of lower levels of inheritance hierarchy on the upper levels after change in any class. :** Inheritance makes the subclasses dependent on the super class and a change in the super class will directly affect the subclasses that inherit from it i.e. we have to retest all its subclasses. Hence it increases dependency among classes which results in low testability. So, in this case, it is better to check the control flow in the form of classes first and then prioritize the highly affected class and then its test cases.

(2) **Every test case detects some faults that are new or detected earlier:** 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.

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.

The proposed technique order**s** 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 numbers 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 numbers 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 errors get propagated from this class, 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 ordering the test cases of each selected class and the 2nd level of prioritization is performed on the basis of fault criticality. To measure the fault criticality, each fault has been assigned weight.

A. First Level Prioritization

The first level prioritization technique prioritizes the classes of object oriented software using inheritance hierarchy. In inheritance hierarchy the classes at lower level inherits the properties of classes at upper level. Therefore, 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 an order that the classes with higher probability of error propagation get tested first.

The technique for prioritizing the classes of object oriented software has been proposed 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. The base class should be assigned the highest priority because if any errors get propagated from this class, 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 in algorithm 1 given below.

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. Class Level Prioritizer

Output of this component is completion of 1st phase of the block diagram. It prioritizes the classes of program. Highly affected classes are prioritized over less affected classes. Probability of finding error in highly affected classes is more than in less affected classes. Thus test cases of highly affected classes execute before test cases of less affected classes.

C. Second Level Prioritization.

Based on first level prioritization for prioritizing the classes of object oriented software, now second level prioritization has been proposed so that test cases of each class can be prioritized. Second level prioritization is a technique to prioritize test cases on the basis of fault coverage per unit time.

The classes prioritized using first level prioritizations are input to the second level prioritization where the test cases of each individual class are prioritized. The test cases are prioritized based on fault weight and fault coverage.

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 in algorithm 2 given below.

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

In this work faults have been categorized on the basis of severity. The faults are assigned a weight on the basis of structure of program as 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	**4**
Fault in do while loop	**2**
Condition statement under condition statement	**4**
Loop under condition statement	**3**
Fault in nested loop	**4**

IV. Experimental Evaluation and Analysis of Proposed Work

For experimental evaluation and analysis of proposed technique, it has been verified and analyzed by taking a case study of software. The case study consists of four classes, study, lec_time, sports_time and usetime. The class study inherited by two classes, lec_time and sports_time and the lec_time further inherited by 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. The inheritance hierarchy shown in Figure 1 has been used to analyze the proposed technique.

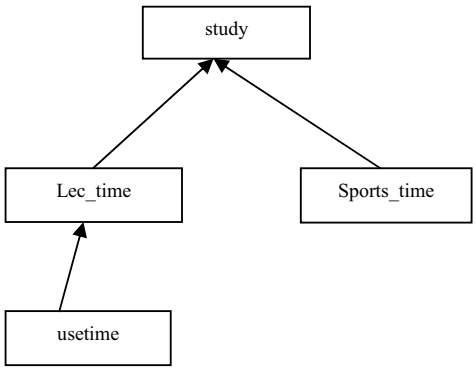

Fig. 1 Inheritance Hierarchy of Case Study

A 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 following table shows the testing effort for each class and the priority assigned to each class.

Table 2. Priority assigned to each class of inheritance heirarchy

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

Lower number indicates higher priority.

B. Second level prioritization

In second level prioritization, test case prioritization technique based on fault coverage per unit time is proposed and an example is explained where APFD metric is used to analyze the proposed technique. The test suite is prioritized on the basis of fault detection per unit time. The prioritized order of test cases of all classes is as shown in the table 3.

Table 3. Result of Proposed Technique

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

C. Comparison of prioritized Test Cases and Random Test Cases of classes

Based on the analysis the prioritized test suite is better as compared to random test suite as shown in Table 4. The graphs comparison of the all the classes is as shown below Figures.

Table 4. *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

D. Comparison of prioritized test cases and random test cases of class study

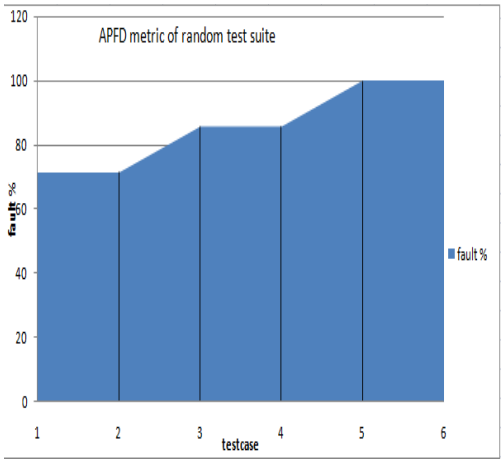

Fig. 2 Fault percentage detected by random test suite (APFD =78)

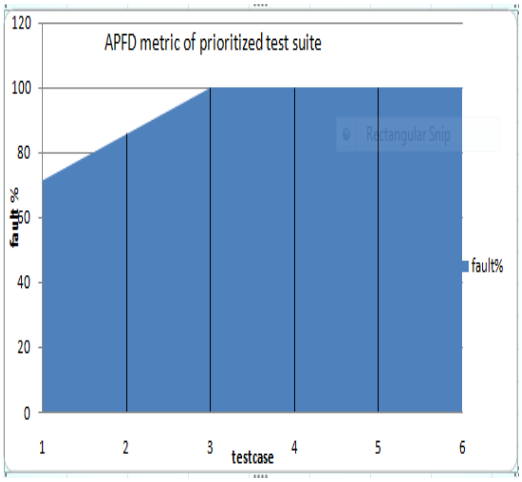

Fig. 3 Fault percentage detected by Prioritized test suite (APFD = 85)

E. Comparison of prioritized test cases and random test cases of class lec_time

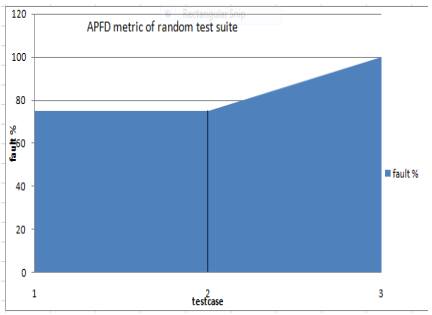

Fig. 4 Fault percent detected by random test suite (APFD = 66.67)

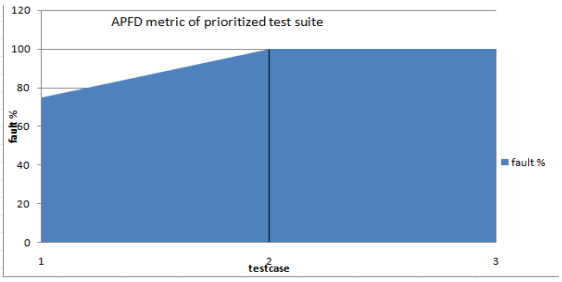

Fig. 5 Fault percent detected by prioritized test suite (APFD = 75)

F. Comparison of prioritized test cases and random test cases of class sports_time

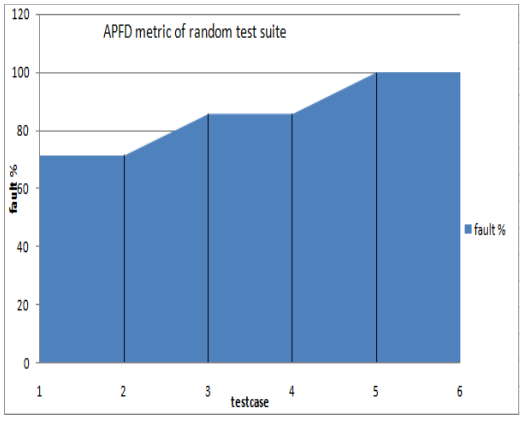

Fig.6 Fault percentage detected by random test suite (APFD = 66.67)

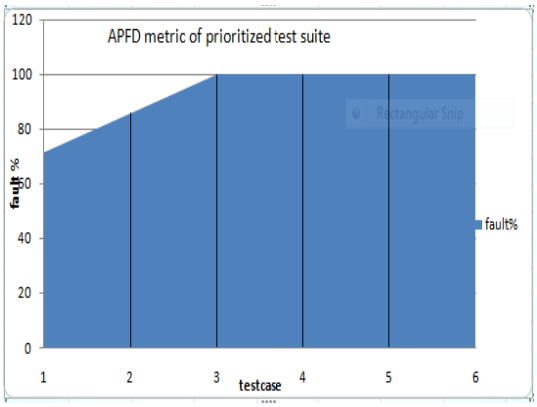

Fig. 7 Fault percentage detected by prioritized test suite (APFD = 85)

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 hierarchy and fault coverage is a beneficial technique for the purpose of saving resources such as time and cost. In the proposed 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 prioritizing the 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 an example (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.

The first level prioritization can be made more effective if multiple inheritance is also been considered. The ambiguity is introduced if base class and derived class has function with same name. Some technique may be designed for dealing with ambiguities of multiple inheritances while prioritizing test cases.

ACKNOWLEDGMENT

I would like to express my gratitude to my colleague Mrs. Rashmi Popli for her valuable advice and helpful discussion. I am also thankful to all my students who help directly or indirectly in completing this paper.

REFERENCES

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

[2] Arti Chhikara, R. S. Chhillar and Sujata 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] Muhammad Rabee Shaheen and Lydie du Bousquet "Relation between Depth of Inheritance Tree and Number of Methods to Test" 2008 International Conference on Software Testing, Verification, and Validation

[4] Pranshu Gupta and David 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] Gagandeep Makkar Jitender Kumar Chhabra2 and Rama Krishna Challa "Object Oriented Inheritance MetricReusability Perspective" 20 1 2 International Conference on Computing, Electronics and Electrical Technologies [ICCEET]

[6] John Daly. Andrew Brooks, James Miller, Marc Roper and Murray 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] Nasib S. Gill and Sunil Sikka "Inheritance Hierarchy Based Reuse & Reusability Metrics in OOSD" International Journal on Computer Science and Engineering (IJCSE)

[9] Chabbi Rani Panigrahi and Rajib Mall Test case prioritization of object – oriented Programs SETlabs Briefings Vol 9 No. 4 2011

[10]Adam Smith , Joshua Geiger, M. Kapfhammer and Mary Lou Soffa "Test Suite Reduction and prioritization with call trees" ASE' 07 November 5 – 9, 2007, Atlanta , Georgia , USA ACM – 978 – 1- 59593 – 882 – 4/07/0011

[11]Sujata Khatri, Dr. R. S. Chhillar and Arti Sangwan,Sujata Khatri Analysis of Factors Affecting Testing in Object oriented systems Mrs. International Journal on Computer Science and Engineering (IJCSE)