TEST CASE PRIORITIZATION ALGORITHM BASED UPON MODIFIED CODE COVERAGE IN REGRESSION TESTING

Usha Badhera[1], G.N Purohit[2], Debarupa Biswas[3]

Computer Science Department, Banasthali University, India
[1]ushas133@yahoo.com,
[2]gn_purohitjaipur@yahoo.co.in
[3]biswas.debarupa@gmail.com

ABSTRACT

Test case prioritization involves scheduling test cases so that performance of regression testing can be improved. To re execute all test cases after every modification in the code is an inefficient process. A technique is to execute the modified lines of code with minimum number of test cases. The proposed test case prioritization technique organizes the test case in a test suite in an ordering such that fewer lines of code need to be re executed thus faster code coverage is attained which would lead to early detection of faults.

KEYWORDS

Code Coverage, Prioritization of Test Cases, Regression Testing.

1. INTRODUCTION

Research studies show that software maintenance activity takes two thirds of the cost of software production. One of the expensive tasks, after maintenance is to establish confidence that changes made in the system are correct and has not developed any side effects in the already checked software. This assurance is achieved by performing regression testing. Regression testing is the re-execution of test cases that have already been executed [1, 8]. In regression testing number of regression tests increases and it is impractical and inefficient to re-execute all test cases after every change made to software. Most of the times running an entire suite is not possible as it takes significant amount of time to run all tests in a test suite. Software testing and retesting occurs continuously during the software development life cycle to detect errors as early as possible, So various techniques have been proposed for minimizing test suite. Test Suite minimization techniques lower costs by reducing a test suite to a minimal subset that maintains equivalent coverage of original set with respect to particular test adequacy criterion [6].

2. TEST CASE PRIORITIZATION

Rothermel at el. [3,4,5] defines the test case prioritization as:

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers. Problem: Find $T' \in PT$ such that $(\forall T'') (T'' \in PT) (T'' \neq T') [f(T') \geq f(T'')]$ Here, PT represents the set of all possible prioritizations (orderings) of T, and f is a function which when applied to any such ordering, yields an award value for that ordering. The definition assumes that higher award values are preferred over the lower ones. There can be number of possible goals of test case prioritization. The objective of this study is to develop a test case prioritization technique that prioritizes test cases for regression testing on the basis of modified code coverage at the fastest rate possible based on early detection of faults. The minimized test cases are based on modified software which provides the confidence that changes are tested.

3. METHODOLOGIES

Some regression testing techniques are based on program specification and others select test cases based on information about code of the program and modified version. The proposed algorithm is based on code coverage techniques of modified version. Program components that have been modified or affected by modifications are provided along with the test suite based on code coverage, and the algorithm select tests in T that exercise those components and the minimization techniques work like coverage techniques, which select minimal sets of tests through modified or affected program components. There are four methodologies that are available for regression testing. These
methods are [7,9]

3.1 Retest all
3.2 Regression Test Selection
3.3 Test Suite Reduction
3.4 Test Case Prioritization

3.1. Retest -all.

This technique discards the test cases which are not applicable to the modified version of program and test all the remaining set of test cases used for testing of modified program.

3.2. Regression test selection.

This uses information about program, modified program and test cases are selected on some basis. Retest all technique is expensive as it takes time and effort as all test cases are considered for regression testing.

3.3. Test suite Reduction.

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 Regression test selection as former does not permanently remove test cases but selects those that are required. Advantage of this technique is that it reduces cost of validating, executing, managing test suites over future releases of software, but the downside of this is that it might reduce the fault detection capability with the reduction of test suite size.

3.4. Test Case Prioritization.

In this technique each test cases is prioritized according to some criterion and higher priority test cases are executed first. Its advantage as compared to Test suite Reduction is that it doesn't discard or permanently remove the test cases from test suite. Present approach considers test selection and test cases are prioritizing based on early coverage of the code.

4. PROBLEM STATEMENT

Regression Test Selection algorithm based on code coverage [6] considers executing test cases which covers modified lines of code. Let P be the previous source code and M be the modified version of T. TH be a set of code coverage based test cases to test P. When P is modified to M the objective is to find T', which is a subset of TH that covers all modified lines of code at the earliest. If two or more test cases, have same number of modified lines and their values also matches then consider test case that has few lines of codes other than modified lines.

Inputs

Previous source code
Modified source code
Test case history showing the list of test cases

Our algorithm performs the following operations:

a) For each test case, it checks the modified source code against the previous source code to Identify only those test cases out of test case history which covers modified source code.

b) If a match is found, then it increments the number of matches found for that test case. It records for each test case the total number of matches and the matched values that is the line numbers. If for two or more test cases, the number of matched values is same, then it checks whether the matched values are also same. If the matched values are also same, then out of the test cases it keeps the test case that covers minimum number of lines of code and if those test cases have equal number of lines of code then it considers any one of them. If any test case has number of matches equals to zero, then discard that test case.

c) Now reduced set of test cases is arranged in decreasing order of number of matches and numbered as 1,2,...,up to the number of test cases remaining.
d) The values of the 1st test case are kept in array A and T'={1st test case}
e) For the remaining test cases repeat steps (i) and (ii),
(i) compare the elements of next test case with the array A. If the test case has at least one different value, add those different values in array A. Otherwise discard that test case.
(ii) If not discarded then compare that test case with the elements of test cases in array T' and store the matched values in different arrays . Add this new test case in array T'.
f) Finally, the resulting test cases in T' are checked for dependencies that is if a particular test case's entries are found in other test cases then that test case is removed.

Output:

Reduced number of test cases.

4.1.Algorithm:-

```
Let the set of test cases be denoted by $T_H$
Let the set of modified lines in the program be denoted by $M$
Let $T$ denote the set of test cases selected for execution.
**Regression_based_function**
$T$=phi
For each test case $t$,
  Compare the modified lines of code against the lines of
  Code covered by the test case then calculate the number
  of matches found and store the matched values
End for
For each test case $t$,
 Check if the number of matches of $t$ is equal to any other
 test cases
        if equal, then compare their matched values
                if matches, then check which out of them have
                    the minimum number of lines of code. keep
                    that test case and discard the others
                    and if the number of elements are equal then keep any
                    one of them.
                End if
            End if
        End if

        If the number of matches of $t$ is equal to zero then
            discard that test case .
        End if
 End for
Arrange the reduced set of test cases of $T$ in decreasing order and numbered as 1,2,...up to
the size of this list.
Put the 1st test case values in array $A$ and $T$={1st test case}
For each test case $t$, Compare the test case $t$ with array $A$
            If $t$ has some different value then
                Put in the array $A$ the different value of $t$.
                Record the matched values $t$ has, with the
                elements of array $T$ separately.
                Put $t$ in array $T$
            Else
                Discard $t$. $T$=$T$-$t$
            End if
 End for
 For each test cases $t$ in $T$
       compare $t$ with the remaining test cases.
```

```
      If all elements of t are found in other test cases then
        Discard t. T'=T'-t
       End if
End for
```

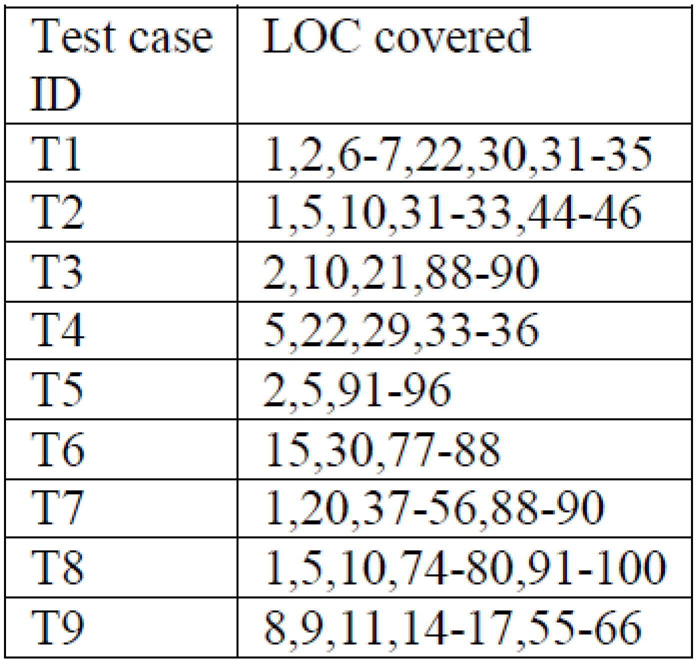
Output

Our final reduced set of test cases is T'.

4.2. Description of the algorithm:-

This algorithm returns minimum number of test cases in order to execute the modified lines of code. Let us consider an example of a program that has 100 lines of code and there are 9 code coverage based test cases. The execution history of each test case is shown in table-1.

Table-1: execution history



Test case ID	LOC covered
T1	1,2,6-7,22,30,31-35
T2	1,5,10,31-33,44-46
T3	2,10,21,88-90
T4	5,22,29,33-36
T5	2,5,91-96
T6	15,30,77-88
T7	1,20,37-56,88-90
T8	1,5,10,74-80,91-100
T9	8,9,11,14-17,55-66

Let the modified lines of code are 1, 2, 5, 10, 15, 20, 21, 22, 29, 30.The steps of our algorithm is explained below with the help of example:

Step 1:-T'=phi

Step 2:

Test case id	No of matches	Matched values
T1	4	1, 2, 22, 30
T2	3	1,5,10
T3	3	2, 10, 21
T4	3	5, 22, 29
T5	2	2, 5
T6	2	15, 30
T7	2	1, 20
T8	3	1,5,10
T9	0	-

Step 3: Since T2 and T8 have same no. of matches and the matched values are also same, T2 is kept as it has fewer remaining number of lines of code then T8. T9 is also discarded as number of matches equals to zero. So the reduced set of test cases are T1, T2, T3, T4, T5, T6, T7.

Step 4: Arrange the test cases in decreasing order of number of matches

T1	1
T2	2
T3	3
T4	4
T5	5
T6	6
T7	7

Step 5: Put T1's values in array A. A={1,2,22,30} and T'={1}

a) Compare A with T2.T2 has got different values which are 5 and 10.So put the values of T1 and T2 in a common array

A= {1, 2, 5, 10, 22, 30}.Matched values of T1 and T2is T1,2={1}.
T'= {1, 2}

b) Compare T3 with A.T3 has a different value ie 21.Put this value into the array

A={1,2,5,10,21,22,30}.Matched values
T1,3={2} and T2,3={10}. T'= {1, 2, 3}.

c) Compare A with T4.It has a different value ie 29.so

A={1,2,5,10,21,22,29,30}
T1,4={22}
T2,4={5}
T3,4=null
T'={1,2,3,4}

d) Compare T5 with A. All values of T5 are covered so T5 is iscarded.
e) Compare T6 with A. Different value is 15.So

A={1,2,5,10,15,21,22,29,30}

T1,6={30}
T2,6=null
T3,6=null
T4,6=null
T'={1,2,3,4,6}

f) Compare A with T7.Different value is 20.So

A={1,2,5,10,15,20,21,22,29,30}
T1,7={1}
T2,7={1}
T3,7=null
T4,7=null
T6,7=null
T'={1,2,3,4,6,7}

Step 6: a) Compare 1 with 2, 3,4,6,7.

T1,2={1}.
Now T1 - T1,2=T1'={2,22,30}.
Now T1,3={2}.so T1'-T1,3={22,30}
T1,4={22} so T1'-T1,4={30}
T1,6={30} so T1'-T1,6=phi
So discard T1 from T'
So T'={2,3,4,6,7}

b) Compare 2 with 3 ,4,6,7

T2,3={10}.so T2-T2,3=T2'={1,5}
T2,4={5} so T2'-T2,4={1}
T2,6=phi
T2,7={1} so T2'-T2,7=phi So
discard T2.
So T'={3,4,6,7}

d) Compare 3 with 4,6,7.

Now T3,4=phi
so T3-phi=T3'={2,10,21}
T3,6=phi and T3,7=phi.
So T3' not equal to phi so don't discard T3keep it
T'={3,4,6,7}

e) Compare 4 with 3,6,7

T4,3=phi so T4'=T4={5,22,29}
T4,6=phi and T4,7=phi

so T4' not equal to phi. so keep T4
T'={3,4,6,7}

f) Compare 6 with 3,4,7.

T6,3=phi
T6,4=phi
T6,7= phi
So T6'=T6={15,30} not equal to phi so keep T6
T'={3,4,6,7}

g) Compare 7 with 3,4,6.

T7,3=phi
T7,4= phi
T7,6=phi
so T7'=T7={1,20} not equal to phi so keep T7
T'={3,4,6,7}

So final minimized set of test cases are {3, 4, 6, 7}, so only 4 out of 9 test cases executed, ie we execute only 44.44% and a saving of 55.56%

5. CONCLUSION

Processes which combine regression test selection method and minimization along with prioritization may be cost-effective.

his paper proposed an algorithm which improves regression testing for modified version of the program considering modified lines of code and for prioritization it selects test cases which execute fewer line of code thus fault detection is earlier.

Earlier fault detection during regression testing provides early feedback on a system under test and thus debugging activities could start early.

In version-specific test case prioritization [2,6,3], given program P and test suite TH, we prioritize the test cases in TH when P is modified to M, the intent is to find an ordering that will be beneficial for specific version M of P. Version specific prioritization is performed after a set of changes have been made to P and prior to regression testing on M, as this prioritization is accomplished after M is available thus version specific test case prioritization may be less effective on subsequent releases. Moreover, our investigation has been confined to prioritization for regression testing, but it would be advantageous to order tests during requirement specification, so that they can be used in the initial testing of software.

6. REFERENCES:

[1] Aditya P.Mathur.,"Foundation of Software testing",Pearson Education 1st edition

[2] Amrita Jyoti, Yogesh Kumar Sharma& Ashish Bagla,D.Pandey.(2010),"Recent Priority algorithm In regression testing ",International Journal Of Information Technology and Knowledge Management vol-2,No-2,pp.391-394

[3] G. Rothermel ,& Mary Jean Harrold(1996), "Analyzing Regression Test Selection Techniques ", IEEE Transactions on Software Engineering

[4] G.Rothermal, R.H.Untch,C.Chu & M.J.Harrold.(2001),"Prioritizing test cases for Regression Testing",IEEE Transactions on software Engineering

[5] G. Rothermel, R.H.Untch, C.Chu & M.J.Harold (2001)," Test Case Prioritization",IEEE Transactions of software Engineering, 27(10),pp 928-948

[6] K.K.Aggrawal, Yogesh Singh & A.Kaur (2004)," Code Coverage Based technique for prioritizing test cases for regression testing", ACM SIGSOFT Software Engineering Notes of vol. 29, No- 5, pp.1-4

[7] Praveen Ranjan Srivastava.(2008),"Test Case Prioritization", Journal of Theoretical and Applied Information Technology IEEE

[8] Roger S. Pressman (2005)," Software engineering a practioner's approach 6/e"

[9] Sebastian Elbaum, Alexey G.Malishevsky & G. Rothermal(2002),"Test case Prioritization: A family of empirical studies", IEEE Transactions on Software Engineering,Vol.28,No-2,pp 159-182.

[10] G. Rothermel and M.Harrold, (1996) “Analyzing regression test selection techniques” IEEE techniques on Software Engineering, 529-551,Aug,.

[11] Elbaum , A Malishevsky and G. Rothermel (2001b.), “ Incorporating varying test costs and fault severities into test case prioritization”, in International conference of software engineering, pages 329-338, may

Authors

Usha Badhera is an active researcher in the filed of software testing, currently working as Assistant Professor in Department of Computer Science at Banasthali University (Rajast han), India. She has done MCA from Rajasthan University and her PhD is in progress from Banasthali University (Rajasthan), India.

Prof. G. N. Purohit is a Professor in Department of Mathematics & Statistics at Banasthali University (Rajasthan). Before joining Banasthali University, he was Professor and Head of the Department of Mathematics, University of Rajasthan, Jaipur. He had been Chief-editor of a r esearch journal and regular reviewer of many journals. His present interest is in O.R.,Discrete Mathematics and Communication networks. He has published around 40 research papers in various journals.

Debarupa Biswas is a student of M.Tech at Banasthali University(Rajasthan).She received her MCS from Assam University, silchar(a central university) in 2010.Her area of interest is Geneti c Algorithms.