International Journal of Information Technology and Knowledge Management
July-December 2010, Volume 2, No. 2, pp. 391-394

RECENT PRIORITY ALGORITHM IN REGRESSION TESTING

Amrita Jyoti, Yogesh Kumar Sharma, Ashish Bagla, D. Pandey

Regression testing is an expensive but necessary maintenance activity, performed on a modified program to instil confidence that changes are correct and have not adversely affected unchanged portions of the program. An important difference between regression testing and development testing is that during regression testing an established suite of tests may be available for reuse. One regression testing strategy, the retest-all approach, reruns all such tests, but this strategy may consume excessive time and resources. Regression test selection techniques, in contrast, attempt to reduce the time required to retest a modified program by selecting some subset of the existing test suite. To reduce the cost of regression testing, software testers may prioritize their test cases so that those which are more important, by some measure, are run earlier in the regression testing process. We propose an algorithm for prioritizing test cases. In this work, we propose a model that achieves 100% code coverage optimally during version specific regression testing. Here prioritization of test cases is done on the basis of priority value of the modified lines covered by the test case.

1. Introduction

Software products continually change and evolve. As the software is modified, renewed testing must ensure both that the new product features work properly and that the changes have not introduced new defects into the product. This process is regression testing and it requires the following steps:

1. Determine the current operational profile.
2. Retain the currently existing test cases that remain valid in the new context.
3. Delete test cases that are no longer valid.
4. Create new test cases for new, enhanced or modified software capabilities.

Regression test selection techniques reduce the cost of regression testing by selecting an appropriate subset of the existing test suite based on information about the program, modified version and test suite.

2. Test Case Prioritization

Test case prioritization techniques let testers order their test cases so that those test cases with the highest priority, according to some criterion, are executed earlier in the regression testing process that lower priority test cases.

There are two varieties of test case prioritization viz. general test case prioritization and version specific test case prioritization. In general test case prioritization, for a given program P and test suite T, we prioritize the test cases in T that will be useful over a succession of subsequent modified version of P without any knowledge of modified version. In version specific test case prioritization, we prioritize the test cases in T, when P is modified to P', with the knowledge of the changes that have been made in P.

In this work, we concentrate on version specific test case prioritization. We propose a prioritization technique that achieves modified code coverage at the fastest rate possible.

3. Problem Statement

Let P be a procedure or program.

P' be a modified version of P

T is a test suite created to test P.

When P is modified to P', we have to find T' which is a subset of T that achieves maximum code coverage at the earliest and should be given highest priority during regression testing, For this purpose, we want to identify tests that

- Execute modified code at least once at the earliest;
- Execute code that has been inserted or deleted so that the changes in the program due to insertion or deletion of statements can be taken care of.

4. Algorithm for Regression Testing

A. Inputs to the Algorithm

1. Old source code;
2. New source code;

[1]ABES Engg. College, Ghaziabad (U.P.)
[2,3]SDCET Muzaffarnagar (U.P.)
[4]ABES Engg. College, Ghaziabad (U.P.)
E-mail: [1]amrita_p2@rediffmail.com, [2]yks_mzn@rediffmail.com, [3]baglaashish@rediffmail.com

3. Test case history showing the list of test cases;
4. Changes in the old source code. It indicates the line nos. that is affected by the changes.
5. Priority associated with the modified lines. It indicates the priority value of the modified lines given by the tester according to some rules*. These rules can be change according to tester and as well as requirement of the program which is to be under regression testing.
6. Execution history of each test case that tells the code lines that have been covered by each test case *in our algorithm priority value given to the modified lines 1(lowest priority value) to 7 (highest priority value) by the following rules (we are taking some examples for C++ language)

Rules Priority Value

- modification in o/p function 1

  example1

  original- printf("The output is");

  modified- printf("Now the result is");
- modification in constant value 2

  example2

  original- cost=num*2;

  modified- cost=num*3;

  example3

  original – salary=salary(salary*0.4);

  modified – salary=salary(salary*0.2);
- modification in variables Value 3

  example4

  original – rate=50;

  modified – rate=100;
- modification in i/p function 4

  example5

  original – scanf("%f", &hour);

  modified – scanf("%2f", &hour);
- modification in constant value

  in branch (if, while etc) 5

  example6

  original- elseif (validinput= =–2);

  modified- elseif (validinput= =–1);
- modification in variable value in branch (if, while etc) 6

  example7

  original – else rate=50;

  modified – else rate=100;
- modification in range in branch (if, while etc) 7

  example8

  original- if(hours<0&& minute>200);

  modified- if(hour<0&& minute>150);

B. Output of Algorithm

The output is a set of test cases to be run on new program with the priorities associated with them. The algorithm saves the cost and effort of running extra test cases.

C. Algorithm

Let T be the set of test cases for the program.

```
M be the set of modified lines in the program.
T' be the set of test cases selected for execution.
Regression_test function:
1.  T' = ∅
2.  While (no. of modified lines not equal to 0)
3.  For each test case t
        Count the number of modified lines covered by test
        case t  End for.
4.  For each test case t
          Count the sum of the priority value of the
          modified lines covered by the test case t.
        End for.
5.  For each test case t
        Choose the test case with maximum number of
        modified lines If two or more test cases cover same
        number of modified lines then choose the test case
        which covers maximum priority value.
        End if.
        End for.
6.  Now remove the modified lines covered by the
        selected test case t from modified lines set M.
        M=M-(No. of lines covered by the test case t)
7.  Insert the particular test case t into set T'.
        End While.
```

8. Goto step 2.
9. Exit

5. Description of the Algorithm

Our aim is to execute the modified lines of code with minimum number of test cases. Let us take that for a program of 40 lines of code, there are 7 code coverage based test cases. The execution history of each test case is shown in Table 1. It indicates the line numbers that are traversed by the test case during its execution.

Table 1
Execution History

Test case Id	LOC covered
T1	1-8,13,36,37-40
T2	1-5,13-16,19-22,25-31, 35-40
T3	1-9,13-16,19-22,25-32,35-40
T4	1-9,13-16,19-22,25-32,35-40
T5	1-13,36-40
T6	1-9,13-21,27-30,35-40
T7	1-4,13-16,19-32,35-40

Suppose that the lines of code 5,7,10,17,21,23 and 26 are modified and priority value of the modified lines according to given rules are shown in table 2.

Table 2
Given Rules

Modified line	Priority value
Line no.5	1
Line no. 7	5
Line no. 10	4
Line no. 17	6
Line no. 21	3
Line no. 23	2
Line no. 26	5

Table 3 shows the no. of modified lines of code and the sum of the priority value of the modified lines. We can find out that test case T1 covers 2 modified lines of code. Similarly T2 covers 3 modified lines of code. This is computed for all test cases.

Table 3
No. of Modified Lines

Test case Id	No.of modified Lines covered by test case (line nos.)	Sum of the priority value of the modified lines= total sum
T1	2 (5,7)	1+5 = 6
T2	3 (5,21,26)	1+3+5 = 9
T3	4 (5,7,21,26)	1+5+3+5 = 14
T4	4(5,7,21,26)	1+5+3+5 = 14
T5	3(5,7,10)	1+5+4 = 10
T6	4(5,7,17,21)	1+5+6+3 =15
T7	3(21,23,26)	3+2+5 =10

Test cases T3, T4 and T6 cover maximum number of modified lines of code. But T6 is executed first since it has maximum priority value.

Modified lines of code still to be executed= {10, 23, 26}

Again we check for the no. of modified lines of code that are not yet executed but covered by the test case. By doing this, we see that now T1 covers none of the modified lines of code that are not yet executed, T2 covers one of the modified lines of code and so on. This is represented in Table 4.

Table 4
Lines Covered by Code

Test case Id	No. of modified lines covered by test case (line nos.)	Sum of the priority value of the modified lines= total sum
T1	0	0
T2	1(26)	5
T3	1(26)	5
T4	1(26)	5
T5	1(10)	4
T7	2(23,26)	5+2 =7

First T7 is executed since it covers maximum modified lines of code of all the test cases listed in Table 4.

Modified lines of code still to be executed= {10}

The same prioritization process is repeated till all the modified lines of code are covered. This is represented in Table 5.

Table 5
Prioritization Process

Test case Id	No. of modified lines covered by test case (line no.)	Sum of the priority value of the modified lines= total sum
T1	0	0
T2	0	0
T3	0	0
T4	0	0
T5	1(10)	4

Now T5 is executed.

The test cases in order of priority are:

Test case no.: 6

Test case no.: 7

Test case no.: 5

Hence 3 out of 7 test cases is executed. We need to run only 43% of test cases to achieve 100% code coverage of modified lines of code. Hence 57% saving in test cases is achieved using this method of prioritization.

In case of any deletion of statements in the original program, we remove the lines of code that have been deleted from the execution history of the test cases. Then we check whether there are any redundant test cases and if any such test case is found, it is removed from the test suite.

The algorithm has been implemented in C++ language. The cost of implementing the algorithm is almost negligible but it saves the cost and effort of running extra test cases.

REFERENCES

[1] Aggarwal K.K., Yogesh Singh and Arvinder Kaur(2004) “Code Coverage Based Technique for prioritizing Test Cases for Regression Testing”, ACM SIGSOFT, 29, Issue 3, September.

[2] Anneliese von Mayrhauser and Kurt Olender(1993) “Efficient Testing of Software Modifications”, IEEE International Test Conference.

[3] Elbaum S., A.Malishevsky and G.Rothermel(2002) “Test Case Prioritization: A Family of Empirical Studies”, IEEE Transactions on Software Emgineering,28, No. 2, February.

[4] Gregg Rothermel and Mary Jean Harrold(1996) “Analyzing Regression Test Selection Techniques”, IEEE Transactions on Software Engineering, 22, No.8, August.

[5] Gregg Rothermel, Roland H.Untch and Mary Jean Harrold(2001) “Prioritizing Test Cases for Regression Testing”, IEEE Transactions on Software Engineering, 27, No. 10, October.

[6] Todd L.Graves, Mary Jean Harrold,Jung Min Kim, Adam Porter and Gregg Rothermel(1998) “An Empirical Study of Regression Test Selection Techniques”, IEEE Transactions on Software Engineering.

[7] Yogesh Singh. Arvinder Kaur and Bharti Suri(2005) “A New Technique for Test Case Prioritization for Regression Testing”, BIMC, April.

[8] Yuejian Li and Nancy J.Wahl(1999) “An Overview of Regression Testing”, ACM SIGSOFT, 24, No.1, Janua.