International Journal of Engineering &Technology, 7 (3.12) (2018) 444-448

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Framework for Optimizing Test Cases in Regression Testing

R.P Mahapatra[1], Aparna Ranjith[2], A. Kulothungan[3]

[1]Professor, [2]M.Tech Student, [3]Assistant Professor
[1,2,3]Department of Computer Science and Engineering
SRM University, NCR Modinagar

Abstract

Software once developed is subject to continuous changes. Software Regression Testing thus can be used to reduce the efforts of testing the software by selecting only the required number of test cases and ordering them to test the software after changes have been made to it. In order to improve the fault detection rate, the selection of efficient test cases and order of execution of these tests is important. Here is when the test case selection comes into picture where in, the fault detection rate during the working of any software has to be improved. The test case selection process will find the most efficient test cases which can fully functionally test the software that has been modified. This indeed will contribute to an improved fault detection rate which can provide faster feedback on the system under test and let software engineers begin correcting faults as early as possible. In this paper, an approach for test case selection is proposed which takes into consideration the effect of three parameters *History, Coverage and Requirement* all together in order to improve the selection process. This will also ensure that the rejection of some efficient test cases is reduced when compared to the selection process in conventional methods, most of them making use of a single parameter for test case selection. These Test cases are further optimized using Genetic Algorithm. Results indicate that the proposed technique is much more efficient in terms of selecting the test cases when compared to conventional techniques, thereby improving fault detection rate.

Index Terms: *Regression Testing, History, Coverage, Requirement, Mutation, Crossover.*

1. Introduction

Software testing is the process of testing the quality of the product or software under test and providing the relevant information about the quality to its stakeholders. Software Testing eases out the process of understanding risks associated with the implementation of software and gives a detailed idea on which part of the software is to be worked on in order to get the required output. For this purpose, the program or application is run with intent to find errors or bugs. It also helps to verify that the end product meets all the requirements of the users or clients who requested for development of the
particular application. There are certain factors on the basis of which software is tested:

a. It should meet all the requirements as initially stated before development
b. It should have correct response to all kinds of inputs
c. All its functions are performed within an estimated accurate period of time
d. It should be usable
e. It should be capable of getting installed and run in the expected environment

It should achieve all the results that are desired by its stakeholders
Regression testing is a type of software testing which is done to check whether the software gives desired output even after changes have been made to it. Changes may include software enhancements, patches, configuration changes, etc. During regression testing, new software bugs or faults can be found. The purpose of regression testing is to ensure that changes such as those mentioned above have not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software. Software Regression Testing is carried out to reduce the efforts of testing the software by selecting only the required numbers of test cases in order to test the software after changes have been made to it. This ensures that the fault detection rate is increased.

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

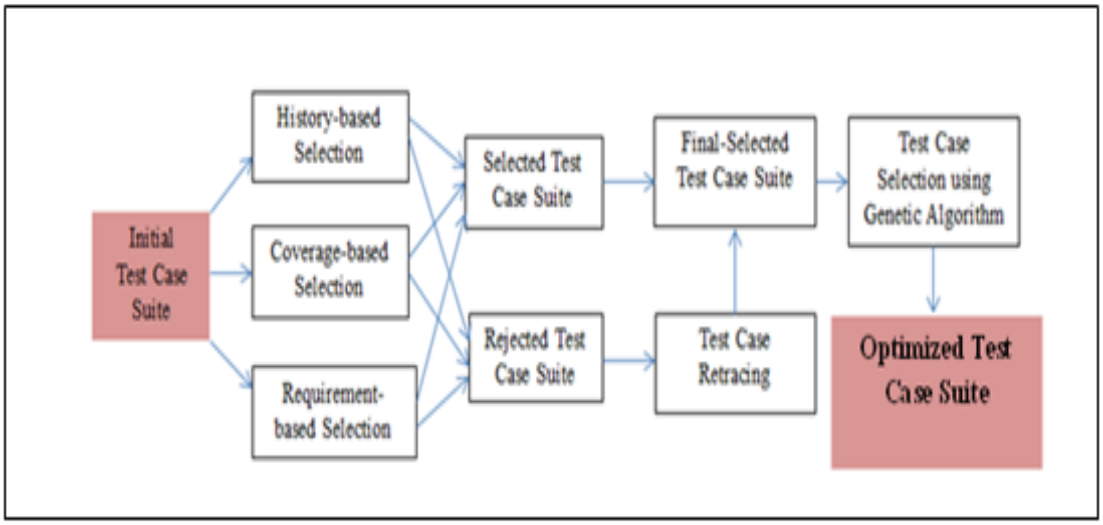

Fig. 1: The Overall Proposed System

2. Problem Description

Existing Problem

1. Most of the conventional test case selection techniques are based only on a single factor like the coverage and not on combination of two or more factors like history, coverage and requirement. Also, some efficient test cases get rejected from the test case suite for the very same reason.
2. In conventional Test case selection techniques, some effective test cases are rejected due to incorporation of too few parameters for selection , for ex: considering only the coverage information for test case selection. Moreover, there are no methods proposed for retracing a few of these test cases.

Proposed Solution

1. To select effective test cases into the final test case suite for regression testing the proposed approach takes into consideration, the use of three different variables/parameters: History, Coverage and Requirement for each test case. This will improve the selection of test cases into the final test case suite.
2. The proposed approach will also include retracing of test cases when no more test cases can be selected using the parameters: History, Coverage and Requirement, This will ensure that effective test cases are retained which is not possible in case of conventional methods which use only single parameter for test case selection .

3. Overall Proposed System

The Overall Proposed System, Fig. 1 shows the order of processes carried out to obtain the Effective order of Test Cases. The Initial Test case Suite undergoes various steps of Selection and then Prioritization to yield a set of Effective Test Cases.
In the proposed approach, the test case selection is done considering three parameters: History Coverage and Requirement and each selected test case is added to the Final Selected Test Case Suite. Few Test cases are initially selected on the basis of History in descending order of History values such that every new test case selected covers at least one new statement that has not been previously covered. Otherwise, the test case is added to Rejected Test Case Suite.
Then selection is based on Coverage, where in a few more test cases are added to the Final- Selected Test Case Suite. Then selection is based on requirement.
Throughout this selection process, if a point is encountered where no more test cases can be selected, Test Case Retracing can be done. This will select a few of the efficient test cases from the Rejected Test Case Suite, which using conventional approach is not an option. These test Cases are also added to meet the number of test cases required after selection process.
Now, the Final Selected Test Case Suite is optimized using Genetic algorithm to obtain minimal number of test cases that can form a Regression test case Suite.

4. Methodology

A. Test Case Selection based on History, Coverage and Requirement

Input: Initial Test Case Suite TC (i, j); i € m, j € n, with m test cases and n statements
Coverage Vector T1
N, Number of test cases to be covered based on history (or coverage or requirement)
Output: A set of Final Test cases TCF (i, j); i € m, j € n
A set of Rejected Test Cases TCR (i, j); i € p, j € q

1. Set Count =1
2. Sort TC (i, j) based on descending values of history (or coverage or requirement)
3. for i: 1 to 20 do
3.1.if Count = N, then
break
3.2.Otherwise, if TC (i) covers a statement j not yet covered, then
3.2.1. Add TC (i) to TCF
3.2.2. Increment Count
3.3. Otherwise
3.3.1. Add TC (i) to TCR

B. Test Case Retracing Algorithm

Input: Complete set of Rejected Test Cases TCR (i, j) after history and coverage based selection; i € p, j € q
Output: A set of Final Test cases TCF (i, j); i € m, j € n

1. Sort TCR (i, j) based on descending values of sum of coverage and requirement.
2. Sort TCR (i, j) based on descending values of requirement.
3. For i: 1 to n do
3.1.Add TCR (i) to TCF

C. Test Case Selection using Genetic Algorithm

Input: Complete set of Final-Selected Test Cases
TCF (i, j); i € m, j € n
Output: Optimized set of Final Test cases TCO (i, j); i € x, j € y

1. Initially set prob (desired probability) =100
2. for i: 1 to m do
2.1. Initialize sum=0
2.2. for j: 1 to n do
2.2.1. sum=sum +gene (j)*Requirement (j)
2.2.2. Fitness value F (i) =sum
3. For every two test cases TCF1 and TCF2 in TCF do
3.1. Perform TCF1 OR TCF2
3.2. Check if prob is reached
3.2.1. Add TCF1 and TCF2 to TCO
3.3. Otherwise
3.3.1. Perform Cross Over
3.3.2. Check if prob is reached
3.3.2.1. Add TCF1 and TCF2 to TCO
3.3.3. Otherwise perform mutation
3.3.3.1. Check if prob is reached
a. Add TCF1 and TCF2 to TCO
3.3.4. Otherwise
3.3.4.1. Take next two test cases (one with higher fitness value among TCF1 and TCF2) and other test case as next test case from TCF for next iteration of genetic loop
4. If (no test cases obtained prob =100) then
4.1. Set prob = 70
4.2. Call genetic loop again (Step 3) for all combinations of test cases

5. Case Study

Let us consider the following source code for across which test cases are to be evaluated.

Input: Values of variables a, b, c, d
Pseudo Code:

```
Function f1 ()
{
        if ((a<b) && (a<c))
                t=a+b;

        else if (d<a)
                t=a+d;
        else if ((a==c) &&(c==d))
                t=a+c+d;
        else
                t=a+b+c+d;
}

Function f2 ()
{
        if (b>c)
                t=b+c;
        else if ((c>d) &&(c<a))
                t=c+d;
        else if ((d<c) && (a<b))
                t=a+b+c+d;
        else
                t=a;
}
```

The source code has two functions with seven statements each and each statement has an associated requirement cost shown as in Table 1

Table 1: Requirement cost of every statement in the source code

Statement	Requirement Cost
1	5
2	5
3	9
4	9
5	2
6	2
7	6
8	10
9	10
10	6
11	6
12	8
13	8
14	5

A. Implementation of Test Case Selection based on History, Coverage and Requirement

The Initial Test Case Suite consists of all test cases to be executed to test the software. Through Regression Testing, the main aim is to select the minimum number of test cases that can efficiently test all functionalities of the software which has been modified. For this purpose, we first go for Test Case Selection which is aimed at selecting test cases on the basis of certain variables like *History*, *Coverage* and *Requirement*.
For example: Consider that the criterion given by the client company is to select 50 percentage of test cases initially with 20 percent cases based on History, 20 percent based on Coverage and 10 percent given to Requirement. Let us consider the initial Test Case Suite as shown in Table 2.
The Requirement value of every test case is the sum of corresponding requirement values of statements covered by the test case in each function. The Coverage of every test case is the count of number of statements covered collectively in both the functions of source code.

Table 2: Initial Test Case Suite

Test Case	a	b	c	d	History
1	12	2	4	2	8
2	7	6	5	2	2
3	1	3	3	2	3
4	2	3	7	8	5
5	1	2	2	2	8
6	3	3	3	3	8
7	4	2	6	2	6
8	7	7	7	7	6
9	6	5	12	13	3
10	8	13	7	2	3
11	7	1	4	2	9
12	10	11	12	13	10
13	12	13	12	1	7
14	2	2	2	2	7
15	8	7	6	5	2
16	13	24	1	12	8
17	1	2	34	5	1
18	9	0	2	7	4
19	1	2	8	1	3
20	4	4	4	4	4

Table 3: Initial Test Case Suite with Coverage and Requirement values calculated

Test Case	a	b	c	d	History	Coverage	Requirement
1	12	2	4	2	8	6	15
2	7	6	5	2	2	5	19
3	1	3	3	2	3	6	13
4	2	3	7	8	5	6	10
5	1	2	2	2	8	6	10
6	3	3	3	3	8	8	7
7	4	2	6	2	6	7	14
8	7	7	7	7	6	8	7

Test Case	a	b	c	d	History	Coverage	Requirement
9	6	5	12	13	3	8	11
10	8	13	7	2	3	5	19
11	7	1	4	2	9	6	15
12	10	11	12	13	10	6	10
13	12	13	12	1	7	5	19
14	2	2	2	2	7	8	7
15	8	7	6	5	2	5	19
16	13	24	1	12	8	5	19
17	1	2	34	5	1	6	13
18	9	0	2	7	4	7	14
19	1	2	8	1	3	6	13
20	4	4	4	4	4	8	7

According to the Test Case Selection Algorithm Based on History, Coverage, Requirement, we will perform the following actions:
The first 20 percent of test cases or Four Test cases are to be selected based on history first. So, the initial test case suite will be sorted based on descending values of history, which have been shown calculated in Table 3.
Moving on to the selection process, we will select the first test case from the sorted test case suite and add it to the Final Selected Test Case Suite.
Next test Case is selected from the sorted list only if the new test case covers at least one of the statements which has not been covered by any of the previous test cases. Otherwise, it is added to the Rejected Test Case Suite.
Tables 4 and 5 will show the test cases selected into the final test case and the ones rejected respectively, based on history.

Table 4: Final Selected Test Case Suite after History based Selection

Test Case	History	Coverage	Requirement
12	10	6	10
11	9	6	15
6	8	8	7
16		5	19

Table 5: Rejected Test Case Suite after History based Selection

Test Case	History	Coverage	Requirement
1	8	6	15
5	8	6	10

Table 6: Final Selected Test Case Suite after Coverage based Selection

Test Case	History	Coverage	Requirement
12	10	6	10
11	9	6	15
6	8	8	7
16	8	5	19
9	3	8	11
3	3	6	13

Table 7: Rejected Test Case Suite after Coverage based Selection

Test Case	History	Coverage	Requirement
1	8	6	15
5	8	6	10
8	6	8	7
14	7	8	7
20	4	8	7
7	6	7	14
18	4	7	14

B. Implementation of Test Case Retracing Algorithm

According to the Test Case Retracing Algorithm, we will now select some of the previously rejected but effective test cases from the Rejected Test case Suite on certain criteria. The Rejected test case Suite is first sorted based on descending values of sum of requirement and coverage and then sorted again based on descending order of Requirement values.
Now, from the sorted Rejected Test Case Suite we will simply select the first four test cases into the Final Selected Test Case Suite. This will form the complete Final Selected Test Case Suite as shown in Table 8.

Table 8: Complete Final Selected Test Case Suite

Test Case	History	Coverage	Requirement
12	10	6	10
11	9	6	15
6	8	8	7
16	8	5	19
9	3	8	11
3	3	6	13
1	8	6	15
7	6	7	14
18	4	7	14
5	8	6	10

This gives the target 50 percent of Selected Test cases which will be optimized further. These Test Cases will collectively cover all the statements of the Source Code.

C. Implementation of Test Case Selection using Genetic Algorithm

According to the Algorithm for Test Case Selection using Genetic Algorithm, the fitness values calculated for selected Test Cases are shown in Table 9.
We will initially set the desired probability to 100 %. After performing OR operation on all combinations of the selected test cases it was found that 100 % probability was not reached in any of the cases. So, after relaxation of probability value to 70% genetic loop will be called again. Taking T12 = {11000001010101} and T11= {10110001011000} and Performing OR operation on them we get probability of coverage as 64.2% which is less than the desired prob value according to the algorithm proposed. So, next step is to perform crossover (Fig 2.) and check if the desired probability is reached.

Table 9: Selected Test cases and their fitness values

Test Case	Fitness Value
12	39
11	45
6	47
16	43
9	51
3	42
1	45
7	39
18	52
5	47

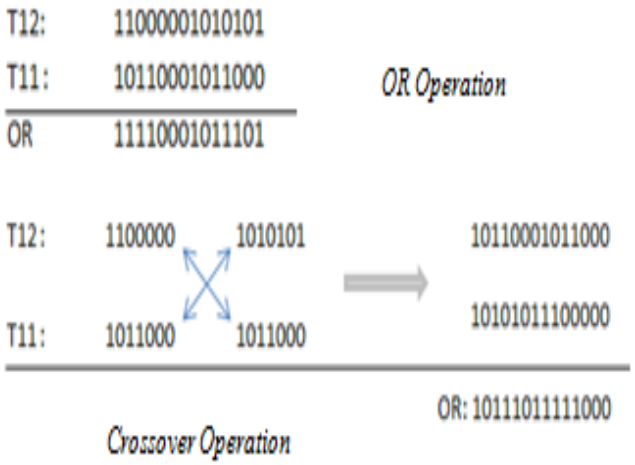

Fig. 2: OR and Crossover Operations

Still the desired probability is not reached. So, next set of parent test cases will be selected. One parent is the one among T12 and T11 having higher fitness value. So, T11 is selected along with T6 as the next set of parents and genetic loop is called. Similarly, we will select five more test cases to form the Final Test Case Suite for Regression Testing.
Table 10 shows the optimized set of test cases obtained finally.

Table 10: Test Cases Selected after Genetic Algorithm

Test Case	Weighted Coverage
11	45
6	47
16	43
3	42
1	45
9	51

6. Performance Analysis

Test Case Selection Based on Coverage V/S Combination of History, Coverage and Requirement

The graph in Fig 3 shows the number of statements covered using conventional method (which uses single parameter like coverage) compared to that of proposed approach (using three parameters: History, Coverage and Requirement) against the number of test cases executed. When all 14 statements have been covered using six test cases in both the approaches, the rejected number of test cases is lesser in the proposed approach compared to that of conventional method. This in turn means that the current approach will have lesser computation cost since it will cover all statements in lesser the number of iterations as compared to the conventional method.

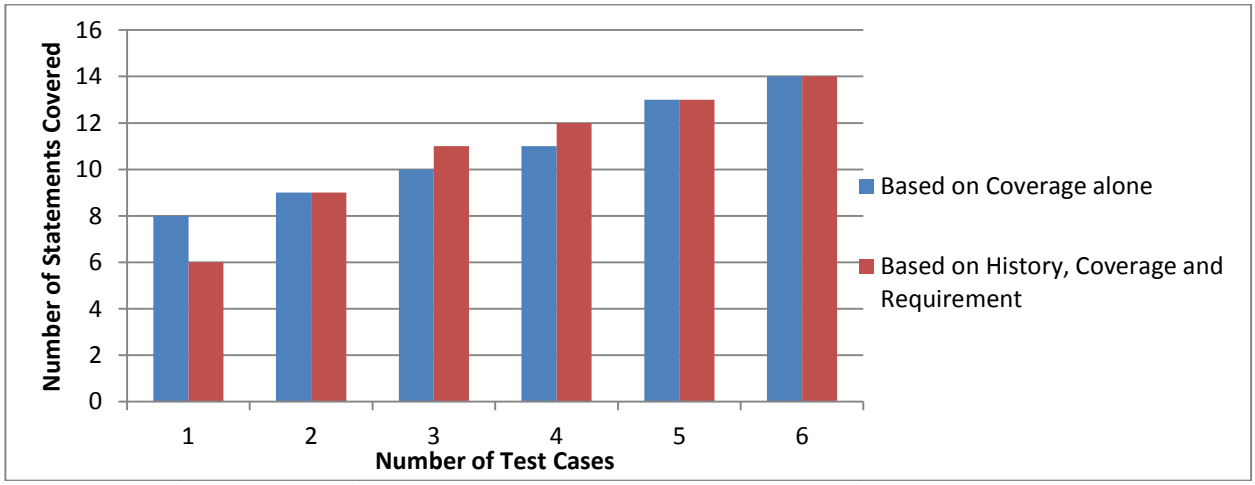

Fig. 3: Performance Analysis of Test case selection based on Coverage v/s Combination of History, Coverage and Requirement

7. Conclusion and Future Work

In this paper, we first propose techniques for selecting test cases for the regression test case Suite from the initial set of test cases by using parameters like History, Coverage and Requirement all together to improve the selection procedure. As compared to the conventional methods, most of which make use of only one parameter like Coverage, the proposed approach has been proved to be more efficient on covering statements and also improving computational cost by reducing the number of iterations needed for test case selection. It also reduces the number of test cases rejected which have a good coverage value.

The test cases are optimized using the Genetic Algorithm which has proved to be efficient in covering all statements of the source code. The Analysis records clearly show that proposed approaches for Test Case Selection and further Optimization using Genetic Algorithm is more efficient than conventional methods. As future work, these selected test cases can be prioritized so as to improve fault detection rate.

8. Acknowledgment

Aparna Ranjith would like to express her deep gratitude to Mr. A. Kulothungan and Dr. R.P. Mahapatra from Department of Computer Science and Engineering, SRM University for their constructive comments and significant support which contributed to the quality of this work.

References

[1] Ekta Khandelwal, Madhulika Bhadauria, "Various Techniques used for Prioritization of Test Cases" IJSRP 2013
[2] Sepher Eghbali, Ladan Tahvildari, "Test Case Prioritization Using Lexicographical Ordering" IEEE 2016
[3] Marwah Alian, Dima Suleiman, Adnan Shaout, "Test Case Reduction Techniques- Survey" IJACSA 2016
[4] Dr. V. Sangeetha, T. Ramasundaram, "Application of Genetic Algorithms in Software Testing Techniques" IAJRCCE 2016
[5] Jaspreet Singh Rajal, Shivani Sharma, "A Review on Various Techniques for Regression Testing and Test Case Prioritization" IJCA 2015
[6] Albert Pravin and Subramaniam Srinivasan, "Effective Test Case Selection and Prioritization in Regression Testing" JCSSP 2013
[7] Shun Akimoto, Rihito Yaegashi, Tomohiko Takagi, "Test Case Selection Technique For Regression Testing Using Differential Control Flow Graphs" IEEE 2015
[8] Shun Akimoto, Rihito Yaegashi, Tomohiko Takagi, "Extended Differential Control Flow Graphs for the Selection of Test Cases in Regression Testing" IEEE 2016
[9] Ritu, Sukhdip Singh, "A Review Paper on Test Case Selection in Regression Testing" IJARCSSE 2016
[10] Hong Mei, Dan Hao, Lingming Zhang, Lu Zhang, Ji Zhou, Gregg Rothermal, "A Static Approach to Prioritizing JUnit Test Cases" IEEE 2012
[11] Emanuela G. Cartaxo, Francisco G. Oliveira Neto, and Patrıca D. L. Machado., "Automated test case selection based on a similarity function. Lecture Notes in Informatics", 7:399–404, 2007.
[12] Tsong Yueh Chen and Man Fai Lau. "Dividing strategies for the optimization of a test suite. Information Processing Letters", 60(3):135–141, 1996.
[13] Tsong Yueh Chen and Man Fai Lau. "Test case selection strategies based on boolean specifications. Software Testing, Verification and Reliability", 11(3):165–180, 2001.
[14] William Dickinson, David Leon, and Andy Podgurski. "Finding failures by cluster analysis of execution profiles". International Conference on Software Engineering, pages 339–348, 2001.
[15] Sahil Gupta, Himanshi Rapria, Eshan Kapur, Harshpreet Singh, And Aseen kumar, " A Novel Approach for Test Case Prioritization " at IJCSEA, Vol. 2, No.3, June 2012.