Analyzing and Enhancing Code Coverage Based Test Case Selection and Prioritization

Anshuman Malhotra¹ Harish Kumar²
¹Research Scholar ²Assistant Professor
¹,²Department of Computer Engineering
¹,²YMCA University of Science & Technology, Faridabad

Abstract—Software Testing is an expensive, time consuming, important activity that controls the quality of the software development & the maintenance. Regression Testing is exclusively executed to guarantee the desirable functionality of existing software after pursuing quite a few amendments or variations in it. Perhaps, it testifies the quality of the modified software by concealing the regressions or software bugs in both functional and non-functional applications of the system. In fact, the maintenance of test suite is enormous as it necessitates a big investment of time and money on test cases on a large scale. So, minimizing the test suite becomes the indispensable requisite to lessen the budget on regression testing. Precisely, this research paper aspires to present an innovative approach for the effective selection and prioritization of test cases which in return may procure a maximum code average.

Key words: Test Case Selection, Test Case Prioritization, Code Coverage

I. INTRODUCTION

Regression testing is an authentication method pursued in all levels of system and software testing. Despite ensuring the functioning capacity of the software or system after making amendments, Regression Testing, exhibits a predominant function with the previously deployed test codes of the enhanced software. The prime aspiration of running a Regression Test is to assure that modified or amended component of software does not give way for bugs in the unaltered portion of the software. The re-execution of test cases are performed to verify that the previous functionality clubbed with the present changes is desirably functioning. The various regression testing techniques are test case minimization, test case selection and test case prioritization. The aim of test case minimization technique is to eliminate the redundant test cases, while test case selection techniques are performed to reduce the size of a test suite. Test case prioritization techniques are concerned with ordering of test cases for detection of faults at the earliest. This paper presents a customized technique for Test case selection and Test case prioritization. Test case selection implies identifying a smaller subset of test suite from the existing large test suite [1]. According to [2], test case selection problem is stated subsequently.

Given: The original program, P, the revised version of P, P' and a test suite, T.

Aim: To identify T′ ∈ T, for the modified version P'.

Test Case Prioritization is the process of arranging test cases in an order according to some criteria. Test case prioritization problem defined by Rothermel et al. [3] is follows:

Given: A test suite, T, the set of permutations of T, PT, a function from PT to the real numbers, f.

Aim: To identify T′ ∈ PT such that (∀T") (T"∈PT) (T"≠T') f(T) ≥ f(T")

Here, ‘PT’ represents the set of all possible prioritizations of ‘T’ and ‘f’. The function that is applied to any such ordering actually yields an award value for the respective ordering.

II. BACKGROUND & RELATED WORK

Code coverage is a quality metric that calculates how thoroughly the test cases exercise a given program. Thus, code coverage provides valuable information as the following: code coverage provides software developers which piece of code is tested as well as which is not. In other words, which portion of code is poorly tested? In addition, code coverage provides a quantitative measure, which is used as an indicator of reliability of software product. Furthermore, code coverage helps to quantify the progress of testing phase. This leads to enhance the test suite without affecting the defect detection process. Moreover, code coverage plays a significant role to discover the dead piece of code as well as code coverage might be used to assess the progress of quality assurance process, and at the same time, plays a guidance of developers. To end this, code coverage is effective to assist in test cases prioritization and generation, which reduces the effort and cost, increases the number of effective test cases as well. For example, in test cases prioritization, code coverage may help to determine which tests we need to remove from the test suite because the redundant tests consume the resources and time. However, there are quite drawbacks such as code coverage may not be able to determine or predict how many defects likely to be found when the code coverage increases. Unfortunately, the code fully covered does not ensure the absence of defects but it is used to assure the quality of test cases. To make code coverage process valuable in the software development process, the developers of code coverage tools provide several coverage metrics: Line coverage, statement coverage, branch coverage, method coverage, class coverage, path coverage, loop coverage, and requirement coverage. In summary, Most of code coverage tools assist in evaluating the effectiveness of testing process by providing a set of code coverage metrics. To answer our research questions, we used line coverage, statement coverage, method coverage, and branch coverage. In addition, code coverage tools, are we used in our experiment, provide these coverage metrics. Therefore, in next subsections, we present an overview of code coverage analysis process as well as we illustrate the features of each code coverage tool that has been used in our experiment.

Before reaching to the proposed work, here introduced basic algorithms on behalf of which we build our new work and a new algorithm for proposed work. These
two algorithms are namely as Test Case Selection & Test Case Prioritization.

A. Test Case Selection
The test cases those are available for the existing version of the program is grouped into three clusters. Those clusters are named as out-dated, required and surplus. The out-dated cluster contains the test cases that are not required by both the original program and the modified program. The required test case group consists of the test cases that are required to be executed for the modified version of the software. The surplus group comprises of test cases that may be essential for the later versions of P but are not required for the modified version of P i.e. P'. The algorithm for Test Case Selection (TCS) which is contributed in the previous work is shown in algorithm 1.

1) Algorithm 1: Test Case Selection

<table>
<thead>
<tr>
<th>Algorithm TCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
</tr>
<tr>
<td>- Matrix TCCij representing the test cases and their statements covered</td>
</tr>
<tr>
<td>- Vector SDELi representing the statements deleted in p'</td>
</tr>
<tr>
<td>- Vector SMODi representing the statements modified in p'</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>- Modified Matrix TCCij, cluster of test cases out dated, surplus, required</td>
</tr>
<tr>
<td>Begin</td>
</tr>
<tr>
<td>1. for each statement that belongs to SDELi</td>
</tr>
<tr>
<td>Remove the corresponding statements from TCCij</td>
</tr>
<tr>
<td>2. Find the sum of each row</td>
</tr>
<tr>
<td>3. if sum of row is 0 then</td>
</tr>
<tr>
<td>Add the corresponding test case in the vector out datedi and</td>
</tr>
<tr>
<td>Remove it from TCCij</td>
</tr>
<tr>
<td>4. Find the test case that do not cover the statements in the vector SMODi</td>
</tr>
<tr>
<td>Add the corresponding test case in the vector surplusi and</td>
</tr>
<tr>
<td>Remove it from TCCij</td>
</tr>
<tr>
<td>5. Add the left over test case in the vector required</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

B. Test case prioritization
The output obtained from algorithm TCS is supplied as input to the algorithm Test Case Prioritization (TCP). Through Test Case Prioritization algorithm we get the required test case vector which represent the optimum set of test cases from available test suite.
1) Algorithm 2: Test Case Prioritization

Algorithm TCP

Input:
- Modified Matrix TCCij representing selected test case and their statement covered

Output:
- Vector TCPi which contains the test cases to achieve 100% code coverage.

Begin

1. Find the sum of each row of the matrix TCCij.
2. Select the test case with highest sum and add the test case into the vector TCPi.
3. Remove all the statements covered by that test case.
4. Repeat steps 1 until all the statements are deleted.

End

III. PROPOSED WORK

Testing involves major role in Software Development Life Cycle (SDLC), a testing is a process and part of SDLC that need to be carried out to obtain a quality software. Testing can be performed on software through designing n number of Test Cases. A Suite of Test Case would able to measure quality of software. A Test Suite may consist n number of test cases which sometime take lot of time to execute and also a lot of efforts of the tester.

Thus we need to perform operation to minimize & prioritize the test cases available in the test suite. The main objective of performing minimization and prioritization on the test cases will helpful in reduction of the number of test cases, overall tester efforts and also time consuming in execution of the test cases. The quality and reliability of software depends on its testing. If testing is done in a proper way then there is high probability for achieving better quality software. Here are some important defects that better testing would have found.

The main objective of testing is to prevent the bugs; by designing the suitable test cases will improve the software quality. The Main Objective of prioritization will be minimization the test suite. The prioritization of the test case in such a way that it maximizes the objective function. Test Case Prioritization would be possible through many methods but in our proposed work Test Case Prioritization based on Code Coverage Technique. Code Coverage Technique states that the:
- Identification of total number of Test Cases available for any program
- Calculate number of statements covered by each Test case in a test suite
  - Test Case covered most statements of any Code will get most priority,
  - Test Case do not cover any of the statement will be eliminated.

There are two algorithms namely Test Case Selection & Test Case Prioritization that will produce the final Test Case Vector known as Required Test Case Vector.

A. Proposed Technique to Minimize & Prioritize Test Cases

The Proposed technique is represented through a Block diagram that is shown in fig. 3.1
Fig 3.1: Steps followed for the proposed work

- Step 1: First of all, there is identification for the Test Cases Covering statements in the program, there are n number of test cases for any program so we have to make cluster of them. However all available test cases are grouped into three clusters.
- Step 2: After this there is identification of the test cases that are not covering any statement at all. There may some test case exist that don’t cover any statement that will be discarded at the initial step.
- Step 3: Now prioritization is based on the code coverage (statement coverage) is performed and the set of test cases are available on the basis of number of statement covered.
- Step 4: In this step, proposed factors are introduced with their corresponding weight value. Here identify type of the statement covered by the test case corresponding to that assign weight value, and new score will calculated on behalf of which we can now prioritize the required test cases.
- Step 5: Finally a prioritized set of test cases is obtained with their final weight prioritized value. This set also known as required prioritized test case suite.

B. Algorithm For The Proposed Work

1) Algorithm 3: Algorithm for the proposed work

```
BEGIN
Step1: Identification of the Test Cases in the program.
      a) Eliminate all test cases that are not covering any statement.
      b) Else, count number of statements that covered by a test case.

Step2: Generate a matrix of test cases that is without out dated test cases.
Step 3: Generate a matrix of test cases that is without Surplus test cases.
Step 4: Assign weight value to the corresponding statement.
Step 5: Demonstrate Test Case set with new score calculated on the factors.

END
```

The algorithm 3.1 is representing the proposed work, it consist of number of steps that need to be carried to accomplish the proposed work.

C. Proposed Factors with their corresponding weights

The proposed approach considers some factors which becomes the basis of test case prioritization. These factors are assigned weights. The weights assigned to these factors is multiplied with the number of statement covered by test case to get the final weight prioritize value on the basis of this weight prioritize value the test cases are prioritize. These proposed factors with their corresponding weights are shown in Table 3.2

```
<table>
<thead>
<tr>
<th>Factor</th>
<th>Factor Weight value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop statement</td>
<td>3</td>
</tr>
<tr>
<td>Conditional statement</td>
<td>3</td>
</tr>
<tr>
<td>Initialization statement</td>
<td>2</td>
</tr>
<tr>
<td>Type casting</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Table 3.2: Proposed factors with their weight values

Based on these factors, a weight prioritization value (WPV) for each test case is calculated by using Formula 1

\[ WPV = \sum_{i=1}^{4}(PF valuei \times PF weighti) \]  

Where, WPV is weight prioritization value for each test case.
PF value, is a value assigned to each test case
PF weight is a weight assigned for each factor.

D. Result Table covering Proposed Work

Table 3.3 consists of the factors of proposed work, in which we can obtain the actual score of the test case on behalf of which we can prioritize the required set of test cases.

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>Statement Covered</th>
<th>Weight Value</th>
<th>Total Score(WPV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T4</td>
<td>2</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>T5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T6</td>
<td>4</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>T7</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.3: Result Table covering proposed work

IV. CONCLUSIONS

The study of Test Case Minimization and Prioritization concluded that overall time in execution of the test cases, cost and efforts could be reduced by applying an optimized technique as presented in our work. Test Case Minimization and prioritization technique specifies a path through which overall test cases available in the test suite would be minimized and prioritized in order to reduce overall execution time, tester efforts and also cost incurred. In our work, an existing technique i.e. code coverage technique is enhanced with more proposed factors that helps in improving the technique that would be better to some extent as compared to existing one.

REFERENCES