Enhanced Weighted Method for Test Case Prioritization in Regression Testing Using Unique Priority Value

Asmaa Ammar
Computer Science and Information Technology
Universiti Putra Malaysia
Serdang, Malaysia
eng.asmaa1991@gmail.com

Salmi Baharom, Abdul Azim Abd Ghani, Jamilah Din
Computer Science and Information Technology
Universiti Putra Malaysia
Serdang, Malaysia

***Abstract*—Regression testing is an integral and expensive part in software testing. To reduce its effort, test case prioritization approaches were proposed. The problem with most of the existing approaches is the random ranking of test cases with equal weight. In this paper, an enhanced weighted method to prioritize the full test suite without using random ranking is presented. In addition, a controlled experiment was executed to evaluate the effectiveness of the proposed method. The results show an improved performance in terms of prioritizing test cases and recording higher APFD values over the original weighted method. In future, a larger experiment would be executed to generalize the results.**

Keywords—Regression testing, test case prioritization, weighted method

I. INTRODUCTION

Software testing is the process of validating and assessing the functionality of a software product. It has been proven that testing, analyzing, and debugging would normally cost over 50% of the cost associated with the development of large software [1]. Regression testing, which is part of the testing process, deals with changes that are often dynamic during a software development life cycle. To enhance the efficiency of regression testing, three main approaches have been studied: selection, minimization, and prioritization. Test case minimization and selection select a subset from existing test cases while neglecting others. In contrast, prioritization method prioritizes the whole test suite. Neglecting some test cases that could find faults will lessen the effectiveness of the regression test. This explains an increased concentration on developing test case prioritization techniques. For this reason, researchers developed many techniques, algorithms, and methods to prioritize test cases [16], [6], [8], [11] and [9]. These techniques order test cases based on specific criteria to increase the possibility of fault detection in early stages of testing. They give high priority to the test cases that are expected to detect more faults than the others.

Most of the existing prioritization methods, regardless of their ranking criteria, share a common issue that should not be neglected. As these methods calculate the value or the weight of test cases, some test cases could have equal values. In this case, they would be sorted randomly. However, this random approach is known as the least effective approach in revealing faults. In addition, almost all techniques were found to be better than the random techniques [4]. Some researchers compared their technique's performance with the random approach and agreed that random prioritization is the least effective approach [13], [5], [15], and [10].

Starting from the test cases with equal weight as a research problem, the existing techniques were examined to look for a solution. The investigation shows that the weighted method [2] is one of the effective existing prioritization methods that uses multi criteria. However, this method still sorts test cases with equal weight randomly. This research chose to enhance the weighted method performance by sorting test cases based on coverage information and the history of last execution to avoid random ranking.

The paper is structured as follows: Section 2 gives a background of prioritization techniques for regression testing; Section 3 presents the principles of the weighted methods and the proposed algorithm; Section 4 presents the experiment; and Section 5 presents the results, discussion and future work.

II. RESEARCH BACKGROUND

First, This section provides an overview of regression testing, test case prioritization, and other earlier works done by authors that encouraged the researchers to propose a unique priority value formula.

Regression testing is a type of software testing that seeks to uncover new software bugs, in existing functional and non-functional areas of a system after changes such as developments or configuration are made to the software. The main drawback of regression testing is the additional cost, time, manpower, etc. that are needed for retesting the program for defects. However, these factors could be reduced to some extent by some techniques such as test case reduction, test case optimization, and test case prioritization [3].

[2] presented different types of regression testing techniques. This research has encouraged new researchers who are planning to start their research to focus on regression testing including its branches. [17] presented a survey on regression test selection, minimization, and prioritization. They argued that there is evidence to suggest the topic of test case prioritization is of

978-1-5090-3493-0/16/$31.00 ©2016 IEEE

increasing importance, judging by the shift in emphasis towards it that is evident in the literature.

In [13] a 4c classification for the existing test case prioritization techniques was presented based on their characteristics. According to this study, the categories of prioritization techniques are: Customer Requirement-based techniques, Coverage-based techniques, Cost Effective-based techniques, and Chronographic history-based techniques. [19] argued that code coverage-based TCPs are the most common TCPs in practice. [14] described several techniques for prioritizing test cases for regression testing and studied their relative capabilities to improve faults detection during regression testing. However, they have found that if code coverage techniques are used in the testing, then they can gain additional improvements through prioritization.

[21] presented several test case prioritization techniques including the total and additional strategy. Each of these techniques depends on single criterion (statement coverage, branch coverage, fault coverage …etc.). The total techniques give priority to the test case that cover the largest amount of code while the additional techniques pick a next test case having the highest coverage of statements not yet covered by previously ranked test cases. However, it is stated that when multiple test cases cover the same number of statements not yet covered, an additional rule is necessary to choose one of these test cases. In this case random selection will be performed.

[8] used coverage-based prioritization techniques (i.e., prioritization in terms of the number of statements, path coverage, branch coverage and fault coverage). The test case that covers more than one coverage criteria is considered to be better than those that cover one criterion. When using this approach, the possibility of two test cases that cover one criterion with equal weight still exists. However, this case has not been discussed, and no practical example is provided.

In [6], the clustering Approach Test Case Prioritization Using statement Coverage Metric was presented. The algorithm calculates the test weight by multiplying the product of two values: statement coverage and function calls. One of the stages of this algorithm is sorting the values of the test cases in descending order. They stated that some test cases could have the same value but without providing any procedure to deal with this case.

[16] presented a dynamic test case prioritization technique based on multiple objectives. The five metrics used are: statement coverage C(t), Fault Exposing Potential FEP(t), requirement property relevance RP(t), historical information H(t), and time spending TC(t). This technique not only missed the issue of many test cases with the same weight, but it was also found that the performance was the same as the statement coverage technique alone.

In [5], new prioritization approach which calculates the product of statement coverage value SC and function calls value FC was presented. They stated that two or more test cases could be given the same priority. To solve the problem, they had two possible scenarios: (1) two product values are equal with different statement and function call values where the priority will be given to the higher function calls, and (2) two product values are equal with equal statement and function calls values where the basis of first come first serve FCFS will be applied. FCFS is a simple approach, but it is not efficient. For example, in the case of the number of faults coverage, the two test cases could have the same priority. In contrast, the test case that reveals more faults may come later than the other one.

[7] presented MOTCP (Multi Objective Test Case Prioritization), a software tool that implements a multi-objective test prioritization technique based on the information related to code and requirements coverage, as well as the execution cost of each test case. Some of the stages of this approach use random selection to prioritize the test cases.

On the other hand, to solve the issue of test cases with equal weight, [13] presented a new method called MTSSP. It consists of four stages. In the first stage, test cases are prioritized according to the Defect factors. If the problem is not solved, then the suite is prioritized according to the Time factors. In the next stage, if the problem is still not solved, then the suite is prioritized according to the Cost factors. If this step also fails, it moves on to the Complex factors. Finally, if the problem is not solved by the Complex factors, then the suite is prioritized by a random method. However, this technique has two main problems: (1) all the 13 factors need to be calculated before the prioritization process which involves more time and cost and (2) the technique solves the problem partially because the problem will go through a random order following the failure of the final stage.

In [12], the weighted method to prioritize the test cases based on five coverage criteria was proposed. These criteria are statement coverage, function coverage, path coverage, branch coverage, and fault coverage. By analyzing its performance, it was found that some of the test cases had an equal weight; thus, random ordering was used. In this research, the researchers prefer to enhance this method for the purpose of prioritizing test cases without depending on random sorting. These are the reasons for choosing the weighted method: (1) many prioritization methods have been proposed, so it is better to enhance their performance instead of proposing new techniques; (2) this method covers more than one coverage criteria for a single set of test cases; (3) ease of replication; and (4) it still depends on random order for multiple test cases with equal weight.

After a thorough review for the existing methods, it was found that some of these papers did not elaborate the details of prioritization process. In addition, other papers either did not provide the experiment objects or the results could not be obtained in the re-execution process. The reasons for choosing the weighted method for this study are: (1) It is coverage based method; (2) the prioritization process was explained in details with illustration experiment; (3) when the experiment re-executed, same results were obtained.

In order to prioritize test cases with the same weight, the principles of history based prioritization was used. According to [18], test case that was executed recently has less priority than test case that was not executed for a long time. In spite of the ability of this criterion to assign unique weight for test cases, researchers believe that its performance would be better if it is used alongside with the code coverage methods [20].

III. Proposed Work

In this section, the principles of the weighted method and the proposed algorithm to get better prioritization results are introduced.

A. Weighted Method by Prakash

As mentioned earlier in Section 1, this method uses five coverage criteria: statement coverage, function coverage, branch coverage, path coverage, and fault coverage. The weight calculation process involves the following steps:

- Statement coverage calculation:

The weight WcdTi for test case Ti can be computed by dividing the number of code covered by the test case Ti to the highest number of code covered by any test case Ti.

$$W_{cd}T_i = \frac{N_{cd}}{M_{cd}} \times 10 \qquad (1)$$

Where: N_{cd} - number of codes covered by the test case T_i

M_{cd} - Maximum number of codes covered by any test case T_i

- Function coverage calculation:

The weight WfnTi for test case Ti can be calculated by dividing the number of functions covered by the test case Ti to the maximum number of functions covered by any test case Ti.

$$W_{fn}T_i = \frac{N_{fn}}{M_{fn}} \times 10 \qquad (2)$$

- Path coverage calculation:

The weight for the path coverage WptTi for the test case Ti can be calculated by dividing the number of paths covered by the test case Ti to the maximum number of paths covered by any test case Ti in the test cases.

$$W_{pt}T_i = \frac{N_{pt}}{M_{pt}} \times 10 \qquad (3)$$

- Branch coverage calculation:

The total of Nbr branches covered by the test case Ti and Mbr branch is the maximum number of branches covered by the test case Ti, then the weight for the test case Ti.

$$W_{br}T_i = \frac{N_{br}}{M_{br}} \times 10 \qquad (4)$$

- Fault overage calculation:

The weight for the fault coverage WflTi for the test case Ti is calculated by dividing the number of faults covered by the test case Ti to the maximum number of faults covered by the test case Ti in the test case list.

$$W_{cfl}T_i = \frac{N_{fl}}{M_{fl}} \times 10 \qquad (5)$$

- For n test cases and m coverage criterion, the overall weight for each test case TCW is as follows.

$$TCW_i = \frac{\sum_{j=1}^{m} Wx_jT_i}{m} \qquad (6)$$

Where:

X_j – test case criteria

M – total number of criterion

Sort the test case based on its weight value from maximum to minimum. The resulted order will be considered as the initial order for the proposed algorithm.

B. The Proposed Algorithm

- Create a group from each set of test cases having the same weight (G1, G2 …), where G1 includes all the test cases with weight w1, G2 includes test cases with weight w2 …etc.
- Arrange test cases within the group based on the previous execution order (initial order).
- It was noticed that when two or more test cases still have equal weight after the weight calculation, they are mostly cover the same segment of the code. Based on this fact, researchers chose to execute one test case from each group while delaying the others.
- Choose the first test case within the group. This test case will keep its weight.
- Delay the other test cases within the groups until the last test case in the initial order is executed. The formula for calculating the new weight for the amended test cases is:

T_i W_new=T_l W-(j× 1/n) (7)

Where:

Tl – last test case in the initial order

J – position of test case Ti within a group G

Update the initial order after each group weight calculation.

IV. Methodology

As stated in Section 1, researchers aimed to enhance the performance of the weighted method and avoid the random ranking of test cases. In addition, the researchers wish to consider whether the proposed algorithm will produce better APFD measurements from those obtained from the existing weighted method. Hence, the null hypothesis H_0 will be:

H_0: There is no difference in APFD values between the proposed and the existing weighted method. In contrast, the alternative hypothesis Ha will be:

H_a The APFD values obtained from the proposed weighted method are higher than those obtained from the existing weighted method. Based on the nature of the data and to test the null and alternative hypothesis, the proposed solution had been verified /tested using Experimentation. The following subsections present experiment definition, objects of analysis, dependent variables and measures, experiment operation, and threats to validity.

A. Experiment Definition

The basic definition of the experiment is: To examine the proposed technique for the purpose of evaluation with respect to effectiveness in maximize Statement, function, path, branch, and fault coverage from the point of view of the researcher in the context of the software laboratory.

B. Objects of analysis

In this paper, a small experiment was conducted to assess the explanation of the proposed method and to obtain primary results. The experiment used a small Java code of 12 statements and seven test cases with two seeded faults. To execute the code and collect coverage information, netbeans 8.0.1 was used with Junit and JaCoCoverage plugin. Finally, Microsoft access was used to build tables and visualize the results.

Fig. 1. Java code with test cases

```
public class Quiz {

static public String QStatus(int a, int b,
int c) {
int status = -1;
if (a > b && a > c) { //seeded fault a < c
status = 1;
} else if (b > c) {
status = 2;
} else {
status = 3;
}
switch (status) {
case 1:
return "a is the greatest";
case 2:
return "b is the greatest";
       //seeded fault "c is …"
case 3:
return "c is the greatest";
default:
return "Cannot be determined";
}
}
}
```

```
Test Cases:
T1: a= 3, b=1, c=1
T2: a= 1, b=3, c=2
T3: a= 1, b=2, c=3
T4: a= 2, b=3, c=3
T5: a= 2, b=3, c=1
T6: a= 3, b=1, c=2
T7: a= 2, b=1, c=2
```

C. Dependent Variables and Measures

To evaluate the effectiveness of the original and enhanced methods and compare the results, the five measurements used in the experiment were: APcdC (Average Percentage of Code Coverage), APfnC (Average Percentage of Function Coverage), APptC (Average Percentage of Path Coverage), APbrC (Average Percentage of Branch Coverage), and APflC (Average Percentage of Fault Coverage).

These measurements are derived from the APFD metric. Let T be a test suite contains n test cases, and let cd be a set of m codes revealed by T. Let Tcdi be the first test case in the reordered test suite T' of T that reveals statement i. The formulas are:

$$\text{APcdC} = 1 - \frac{\sum_{i=1}^{m} Tcdi}{nm} + \frac{1}{2n} \quad (8)$$

$$\text{APfnC} = 1 - \frac{\sum_{i=1}^{m} Tfni}{nm} + \frac{1}{2n} \quad (9)$$

$$\text{APptC} = 1 - \frac{\sum_{i=1}^{m} Tpti}{nm} + \frac{1}{2n} \quad (10)$$

$$\text{APbrC} = 1 - \frac{\sum_{i=1}^{m} Tbri}{nm} + \frac{1}{2n} \quad (11)$$

$$\text{APflC} = 1 - \frac{\sum_{i=1}^{m} Tfli}{nm} + \frac{1}{2n} \quad (12)$$

D. Experiment operation

The experiment was executed using the following steps:

1. The coverage information for each test case was calculated. Results are shown in Table 1. It can be seen that even though some test cases could cover different parts, they cover the same amount of statements, functions, paths, branches, and faults.

TABLE 1: COVERAGE VALUES

TC	Code Cov.	Function Cov.	Path Cov.	Branch Cov.	Faults Cov.
T1	6	1	1	4	1
T2	6	1	1	3	1
T3	6	1	1	4	0
T4	6	1	1	4	1
T5	6	1	1	3	1
T6	6	1	1	4	1
T7	6	1	1	4	1

2. The weight of each criteria for all test cases was calculated using (1) to (5), before the overall weight for each test case was calculated using (6). Table 2 shows the resulted weights.

TABLE 2: TEST CASE WEIGHT

Test Case	Cd	fn	pt	br	fl	Overall Weight
T1	10	10	10	10	10	10
T2	10	10	10	7.5	10	9.5
T3	10	10	10	10	0	8
T4	10	10	10	10	10	10
T5	10	10	10	7.5	10	9.5
T6	10	10	10	10	10	10
T7	10	10	10	10	10	10

3. Test cases were ranked based on the weight from maximum to minimum and test cases with same weight were arranged randomly. The resulted order is:

   T4, T7, T1, T6, T5, T2, T3

4. The weighted method was stopped at this point while the proposed method considered the resulted order as the previous execution.

5. A group from each set of test cases having the same weight was created. G1= {T4, T7, T1, T6}, G2= {T5, T2}.
6. The first test case in G1 kept its original weight, T4 = 10. The weights for T7, T1, and T6 were calculated using (7). The new values are:

   T7= 7.72, T1= 7.58, T6= 7.44.
7. The initial order was updated with the new weights so that the last test case in the order would be T6, and steps 5, 6 and 7 for G2 were repeated.
8. The final weights are: T1= 7.58, T2= 7.16, T3= 8, T4= 10, T5= 9.5, T6= 7.44, and T7= 7.72.

The final order is: T4, T5, T3, T7, T1, T6, T2.

E. *Threats to validity*

In this section, the researchers describe the internal, external, and construct threats to the validity of this experiment, and the procedures used to limit the effects of these threats.

Concerning the internal validity, the risk of manually seeded faults is a threat. However, the test cases were randomly generated. Thus, the threat was minimized.

Threats to construct validity were not considered critical. The dependent APFD measures used to evaluate the effectiveness, have some limitations. They suppose that faults are with equal severities and test cases are with equal costs. In this experiment, these assumptions were acceptable. Furthermore, it is a common measure for effectiveness.

The largest threat that may affect external validity is the size of code. However, this experiment was conducted for the purpose of explanation of the proposed method. In future, a larger experiment will be conducted to generalize the results.

V. Data and Analysis

To investigate whether the proposed method is more effective than the weighted methods, five coverage measures were collected using (8) – (12). The recorded values are shown in Table 3.APFD measurements from those obtained from the existing weighted method. Hence, the null hypothesis H_0 will be:

TABLE 3: AVERAGE PERCENTAGE OF DIFFERENT CRITERIA

Method	APcdC	APfnC	APpaC	APbrC	APflC
Proposed Method	89.28	92.85	69.04	87.14	85.71
Weighted Method	78.75	92.85	59.52	87.14	64.28

The bar chart in Figure 2 provides an overview of the collected measurements. The lighter-colored bars indicate the percentage of coverage obtained by the existing weighted method while the darker ones represent the proposed weighted method. Measurements were recorded for the five metrics: APcdC, APfnC, APptC, APbrC, and APflC. It is obvious that the results of the proposed method are higher than the weighted method in three measurements while for APfnC and APbrC, they are the same.

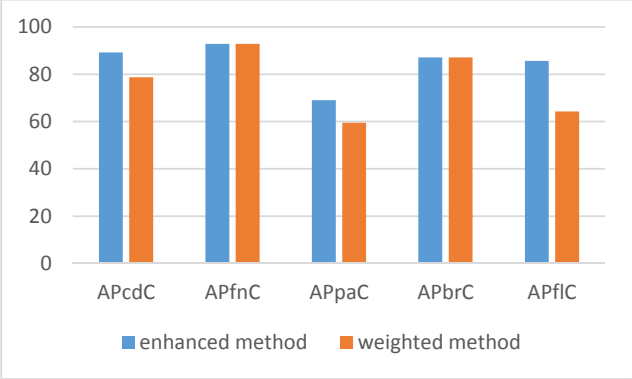

Fig. 2. Difference in Measurements between the Weighted and the Proposed Methods

In spite of the high coverage percentage, these values do not reflect the actual amount of the covered code. The APFD and the derived measurements indicate the results of the covered part by the test only. In our program, 4 out of 12 statements were not covered; thus, achieving only 89.28% of code coverage. This indicates a weak test rather than faulty representation of the data. To give an accurate description, it is better to achieve 89.28% coverage for 66% of the code covered by the test suite.

VI. Conclusion and Future Work

This paper presents an enhanced weighted method for test case prioritization. The main objective of this method is to prioritize test cases without depending on random sorting for the test cases with equal weight. To attain this goal, a controlled experiment was executed, and the results were compared with the existing weighted method to demonstrate its effectiveness. Results show that the enhanced method not only prioritized test cases but also recorded higher percentage of coverage for some criteria. Future work will expand upon this short paper by examining a larger set of experiments to generalize the results.

References

[1] Boris Beizer. 1990. Software Testing Techniques (2nd Ed.). Van Nostrand Reinhold Co., New York, NY, USA.

[2] Duggal, G., & Suri, B. (2008). Understanding regression testing techniques. InProceedings of the 2nd National Conference on Challenges and Opportunities, Mandi Gobindgarh, India, March (29).

[3] Elbaum, S., Rothermel, G., Kanduri, S., & Malishevsky, A. G. (2004). Selecting a cost-effective test case prioritization technique. Software Quality Journal, 12(3), 185-210.

[4] Felderer, M., & Fourneret, E. (2015). A systematic classification of security regression testing approaches. International Journal on Software Tools for Technology Transfer, 1-15.

[5] Gupta, S., Raperia, H., Kapur, E., Singh, H., & Kumar, A. (2012). A NOVEL APPROACH FOR TEST CASE PRIORITIZATION.

International Journal of Computer Science, Engineering and Applications, 2(3).

[6] Hashini, M., & Varun, B. (2014). Clustering Approach to Test Case Prioritization Using Code Coverage Metric. 4th National Conference on Advanced Computing, Applications & Technologies, (May), 3–6.

[7] Islam, M. M., Marchetto, A., Susi, A., Kessler, F. B., & Scanniello, G. (2012, September). MOTCP: A tool for the prioritization of test cases based on a sorting genetic algorithm and Latent Semantic Indexing. In Software Maintenance (ICSM), 2012 28th IEEE International Conference on (pp. 654-657). IEEE.

[8] Kaur, N., & Mahajan, M. (2014). Prioritization of Test Cases using Branch Coverage with Multiple criteria for Regression Testing. / (IJCSIT) International Journal of Computer Science and Information Technologies, 5(2), 972–974.

[9] Korel, B., Koutsogiannakis, G., & Tahat, L. H. (2007, July). Model-based test prioritization heuristic methods and their evaluation. In Proceedings of the 3rd international workshop on Advances in model-based testing (pp. 34-43). ACM.

[10] Muthusamy, T. (2014). A New Effective Test Case Prioritization for Regression Testing based on Prioritization Algorithm. International Journal of Applied Information Systems (IJAIS) 6(7), 21–26.

[11] Muthusamy, T., & Seetharaman, K. (2013). A Test Case Prioritization Method with Weight Factors in Regression Testing Based on Measurement Metrics. International Journal of Advanced Research in Computer Science and Software Engineering, (12), 390–396.

[12] N, P., & T.R, R. (2013). Weighted Method for Coverage Based Test Case Prioritization. Journal of computational Information systems, 235-243.

[13] Roongruangsuwan, S., & Daengdej, J. (2010). Test case prioritization techniques. Journal of Theoretical and Applied Information Technology, 18, 45–60.

[14] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. Software Engineering, IEEE Transactions on,27(10), 929-948.

[15] Tonella, P., Avesani, P., & Susi, A. (2006, September). Using the case-based ranking methodology for test case prioritization. In Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on (pp. 123-133). IEEE.

[16] Wang, X., & Zeng, H. (2014, June). Dynamic test case prioritization based on multi-objective. In Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on (pp. 1-6). IEEE.

[17] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. software testing verification and reliability , 22(2), 67–120.

[18] Fazlalizadeh, Y., Khalilian, A., Azgomi, M.A., et al: 'Incorporating historical test case performance data and resource constraints into test case prioritization', Tests Proofs Lect. Notes Comput. Sci., 2009, 5668, pp. 43–57

[19] Hadi Hemmati, Zhihan Fang, Mika V. Mantyla, "Prioritizing Manual Test Cases in Traditional and Rapid Release Environments", ICST, 2015, 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST) 2015, pp. 1-10,

[20] W. N. Liu, C. Y. Huang, C. T. Lin, and P. S. Wang, "An evaluation of applying testing coverage information to historical-value-based approach for test case prioritization," Proc. Asia-Pacific Symp. Internetware, December 2011, pp. 73-81.

[21] Rothermel, G., Untch, R., Chu, C., Harrold, M.: Prioritizing test cases for regression testing. IEEE Trans. Softw. Eng. 27(10), 929–948 (2001)