International Journal of Engineering & Technology, 7 (4.31) (2018) 20-27

International Journal of Engineering & Technology

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

Research paper



The Effectiveness of an Enhanced Weighted Method with a Unique Priority Value for Test Case Prioritization in Regression Testing

Asmaa Ammar[1]*, Salmi Baharom[2], Abdul Azim Abd Ghani[3], Jamilah Din[4]

[1]Faculty of Computer Science &Information Technology, Universiti Putra Malaysia UPM
**Corresponding author E-mail: eng.asmaa1991@gmail.com*

Abstract

Regression testing is an important and costly strategy in software testing. To decrease its cost, many methods have been proposed including the test case prioritization methods. The aim of the prioritization methods is to define an ideal order of test cases that allows for higher coverage and early fault detection with minimal number of executed test cases. However, the problem with most of the existing test case prioritization methods is the random sorting of test cases when two or more test cases record equal priority values. In this paper, the effectiveness of an enhanced weighted method using a unique priority value, UniVal, is proposed. Univalprioritizes test cases based on code coverage criteria with information from history of previous executions. In addition, a controlled experiment was executed, and the results were statistically analyzed to assess the effectiveness of the proposed method. The results indicates better performance with regard to prioritize test cases and achieve higher APFD values.

Keywords: *Regression testing; Software testing; Statistical test; Test case prioritization; Unique priority value*.

1. Introduction

Software testing is the procedure of authenticating and evaluating the functionality of a software artifact. It was proved that testing, examining, and fixing would normally cost over 50% of the cost related to the development of large software [1]. Therefore, regression testing ought to be an essential phase of software testing. It deals with the modifications that are often dynamic during the life cycle of software development. To improve the efficiency of regression testing, three major methods have been considered by researchers in this field. They are test case selection, test case minimization, and test case prioritization. According to the survey of [2], the field of regression testing approaches continues to grow between 1977 and 2009, and the importance of test case prioritization has gradually risen since the late 90s. As a result, researchers have established many techniques, algorithms, and methods to prioritize test cases [3], [4], [5], [6], [7], [8] & [9]. These techniques rank test cases based on precise conditions to increase the probability of fault uncovering in the early phases of testing. In spite of the differences in the factors, test case prioritization techniques calculate the value or weight of all test cases and use these values to decide which test case should be executed first. Most of the current prioritization methods, irrespective of their sorting criteria, share a common problem that ought not to be ignored. As these methods estimate the value or weight of test cases, and some test cases might hold equal values. To deal with this issue, most references choose the random technique. According to [10] , "When multiple test cases cover the same number of statements, the additional strategy is necessary to select one of these test cases, generally by random selection". [11] Added, "Existing techniques randomly prioritize all test cases with the same weight values, without any systematic algorithm". Nevertheless, this random approach is identified as the least effective method in divulging faults [12], [13], [14], [15] & [11]. Several researchers [15] & [11] proposed additional procedures to sort out test cases that have the same priority values. Unfortunately, these proposed techniques are complex and still have the possibility to generate the same priority values. The objective of this research is to generate unique priority values for test cases. The following sections of the paper are structured as follows: Section 2 presents background information on the issues of multiple test cases with equal weights that provides crucial inputs to this research and consequently leads to the formulation of the study; Section 3 explains the research methodology involved in this study including variable selection and threats to validity; Section 4 describes the design of the enhanced weighted method, and this section also explains the algorithm for generating unique priority values; Section 5 describes the experimental process for enhanced weighted method in terms of its effectiveness by comparing it with weighted method, whereby a statistical test, a paired t-test, was employed to measure the significant differences between the methods under comparison; and Section 6 concludes the study with a general discussion of research outcomes, research contributions, and suggestions for further research.

2. Research Background

The reasons behind software testing are to ensure the quality or acceptability of software and to reveal faults. Software testing is the process of testing bugs in lines of code of a program [16], and system testing is split into two steps: regression and progression testing [17]. Regression testing is also the process of testing changes to a program to check whether the older code still works with the new changes or after a period of active time. [18] Pre-

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.

sented a research travelogue on software testing which began in 2000 and ended in 2014. One of their findings stated that the main challenge for the researchers was to define the techniques for regression testing that could be applied in the real world and on modern software. The main weakness of regression testing is the added charges, time etc. that are required to test a program again for faults. Though, these aspects could be fairly reduced using procedures like test case reduction, test case optimization, and test case prioritization [19] & [20]. [2] Who presented a survey on regression test selection, minimization, and prioritization found proofs to argue that the subject of test case prioritization is of increasing importance, judging by the change in emphasis on it which is obvious in their literature. Drift analysis shows a rise in publications, providing confirmation to support the statement that the field continues to gain attention from the broader research community. There are many objectives for using the prioritization techniques. Software testers may aim to increase the code coverage in a software at an earlier stage, increase the confidence in the reliability of a system, or increase the rate of fault recognition [19]. [11] Presented a 4c classification, which was similar to the classification of [21] for the current test case prioritization methods according to their objectives: Customer Requirement-based techniques, Coverage-based techniques, Cost Effective-based techniques, and Chronographic history-based techniques. [10] Defined some methods in prioritizing test cases for regression testing and studied their comparative skills to improve fault detections during regression testing. Their results suggested that these techniques could increase the frequency of fault revealing of test suites, and this result could be gained even for simple techniques. Moreover, if the code coverage procedures are used in testing, they can score further enhancements by prioritization. Since the concentration of this study is on mixing code coverage criteria with information from history of previous executions, the prioritization techniques were divided into three categories: code coverage based techniques, history based techniques, and other techniques.

2.1. Code Coverage Based Techniques

The main purpose of code-based test prioritization is getting early fault detection during a regression testing of a modified system code [19]. According to [22], the coverage based prioritization is depending on the hypothesis that the increased coverage accomplished by the test suite will increase the probabilities of finding faults faster in the testing procedure. [23] Proposed the potential weighted method to prioritize test cases based on the five coverage criteria proposed: statement coverage (SC), function coverage (FunC), path coverage (PC), branch coverage (BC) and fault coverage (FC). This method scored an improved rate of coverage criteria; however, based on the experiment conducted, it was concluded that several test cases had an equal weight. Consequently, a random sorting was used. An overview of how the weighted method works is shown in Figure 1.

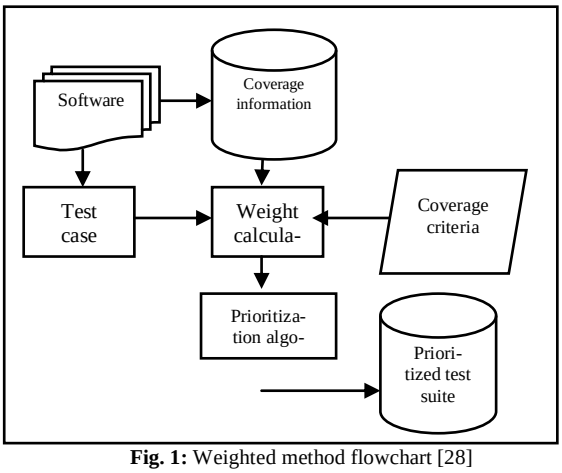

Fig. 1: Weighted method flowchart [28]

2.2. History Based Techniques

The objective of these techniques is to prioritize test cases without accessing the code. However, they require documentation of the software in order to extract the required factors from the historical record. [24] Proposed a history based test case prioritization technique for failure information AFSAC. This approach is consist of two stages. First, it statistically analyses the failure history for each test case to order test cases. Then it reorders test cases by applying the correlation data of test cases collected from earlier test outcomes.

2.3. Other Techniques

Besides code coverage and history based techniques, different factors were used under this category. However, most of these methods share a common factor, which is the requirements. [5] Presented a dynamic test case prioritization technique based on multi objectives. Five factors which have been 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). However, this method in addition to neglecting the issue of many test cases with the same weight, it was also found that its effectiviness was the same as the statement coverage technique alone. After a thorough review of the existing methods, it was found that some of these papers did not elaborate the details of the 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. In this research, the weighted method by [25] was selected to be compared with the proposed method for the following reasons: (1) this method covers more than one coverage criteria for a single set of test cases; (2) ease of replication; and (3) it still depends on random order for multiple test cases with equal weight. [26] Presented an evidence-based review on regression test case prioritization. According to their study, in order to realize the effects and the outcomes of any case study or experiment precisely, there is a need to quantify the results or evaluate them with respect to the measures. Moreover, the famous and most used metrics should be used. APFD is also being used in its altered form as APFDc, APFDp, ASFD, WPFD, TSFD, APBC, APDC, APSC, NAPFD, APMC, TPFD, APRC, and BPFG. 'APFD alike' are mainly the metrics which compute the average percentage of faults detected with slight differences in the calculation process. The notations of APFD is:

$$APFD = 1 - \frac{TF1 + TF2 + \cdots + TFm}{nm} + \frac{1}{2n} \quad (1)$$

where:
m = number of faults detected during the execution of test suite
n = total no of test cases
TFi = position of the initial test in test suite T that detects fault i
To evaluate the effectiveness, [27], [28], [29], [15], [30], & [31] used the APFD metric.

3. Research Methodology

Research methodology is the essence of this study as it determines the methods used for conducting the research, assists in explaining the nature of data, and highlights the methods employed that would lead to the generation of accurate and reliable conclusions. [32] Listed two approaches that can be used when conducting a research i.e. the quantitative and qualitative research methods. This study used the deductive approach by achieving experiments on specific conditions (i.e. a selection of tools under comparison) for the purpose of drawing a general conclusion for the study. For this reason, this study is considered to have applied the quantitative research method.

3.1. Variable Selection

An important stage in planning the activity is to define the variables [the independent variables (inputs) and the dependent variables (outputs) together]. Independent variables refer to the alternative methods to be evaluated while dependent variables were used to measure the effects of the treatments.

3.2. Instrumentation

All The objective of this phase is to choose the appropriate instruments, objects, etc. The objects of the experiment are the three java programs with different LOC. These programs were chosen based on specific criteria i.e.: (1) they were bundled with seeded faults and test cases, (2) they were of different lengths in terms of lines of code LOC, and (3) two of them were open source and one had been used in a previous research on software testing. Table 1 summarizes the details of the experiment objects.

Table 1: Experiment Objects

Object program	source	Lines of Code LOC	Seeded faults	Test cases
Circular Queue	Previous research work[1]	42	9	33
Calculator	Sourceforge.com	148	15	40
ATM	Github.com	295	28	60

The experiment was run using NetBeans IDE 7.1 and Junit 4.0. [33] who conducted a study on various code coverage tools claimed that with the use of code coverage, the testing process could be enhanced, and the cost for fixing the faults could be reduced. Moreover, they described JavaCodeCoverage as a bytecode analyzer tool for test coverage analysis for Java software which doesn't require the language syntax or the source code. One of the important features of JavaCodeCoverage is that it stores the coverage information for test cases individually, thereby simplifying a detailed coverage examination. Additional important feature of JavaCodeCoverage is that it archives all the dynamic code elements and test coverage information in an open source database software, MySQL. Finally, for visualization of results and creating descriptive statistics (bar charts), Microsoft Excel was used in this research.

3.3. Hypothesis Construction

In this experiment, five measurements used to reveal the effectiveness of the proposed method. Therefore, five hypotheses should be tested. The null and alternative hypotheses were formulated as follows:

a) Average percentage of statement coverage $APst C$

H_{0_1}: There is no difference between the two methods in terms of average percentage of statement coverage $APst C$ and $\mu_{0_1} = \mu_{a_1}$.

H_{a_1}: The enhanced weighted method has greater average percentage of statement coverage $APst C$ over the weighted method and $\mu_{0_1} < \mu_{a_1}$.

b) Average percentage of function coverage $APfn C$

H_{0_2}: There is no difference between the two methods in terms of average percentage of function coverage $APfn C$ and $\mu_{0_2} = \mu_{a_2}$.

H_{a_2}: The enhanced weighted method has greater average percentage of function coverage $APfn C$ over the weighted method and $\mu_{0_2} < \mu_{a_2}$.

c) Average percentage of branch coverage $APbr C$

H_{0_3}: There is no difference between the two methods in terms of average percentage of branch coverage $APbr C$ and $\mu_{0_3} = \mu_{a_3}$.

H_{a_3}: The enhanced weighted method has greater average percentage of branch coverage $APbr C$ over the weighted method and $\mu_{0_3} < \mu_{a_3}$.

d) Average percentage of path coverage $APpa C$

H_{0_4}: There is no difference between the two methods in terms of average percentage of path coverage $APpa C$ and $\mu_{0_4} = \mu_{a_4}$.

H_{a_4}: The enhanced weighted method has greater average percentage of path coverage $APpa C$ over the weighted method and $\mu_{0_4} < \mu_{a_4}$.

e) Average percentage of fault coverage *APFD*

H_{0_5}: There is no difference between the two methods in terms of average percentage of fault coverage *APFD* and $\mu_{0_5} = \mu_{a_5}$.

H_{a_5}: The enhanced weighted method has greater average percentage of fault coverage *APFD* over the weighted method and $\mu_{0_5} < \mu_{a_5}$.

3.4. Validity Evaluation

When designing an experiment in software engineering, the key question is "Are the results valid for the population of interest?" In order to be adequately valid, there are four types of validity that have to be considered: internal validity, external validity, construct validity, and conclusion validity [34] & [35].

- Threats to conclusion validity: It refers to the capability of drawing the correct conclusion regarding the relation between the treatment and the outcome of an experiment. This includes matters related to the selection of statistical tests, selection of sample sizes, and accuracy in the implementation and measurement of an experiment. One possible threat to validity in this experimental research is the validity of the statistical tests. To decrease this threat, popular statistical methods were applied which are tough to violations of their assumptions.

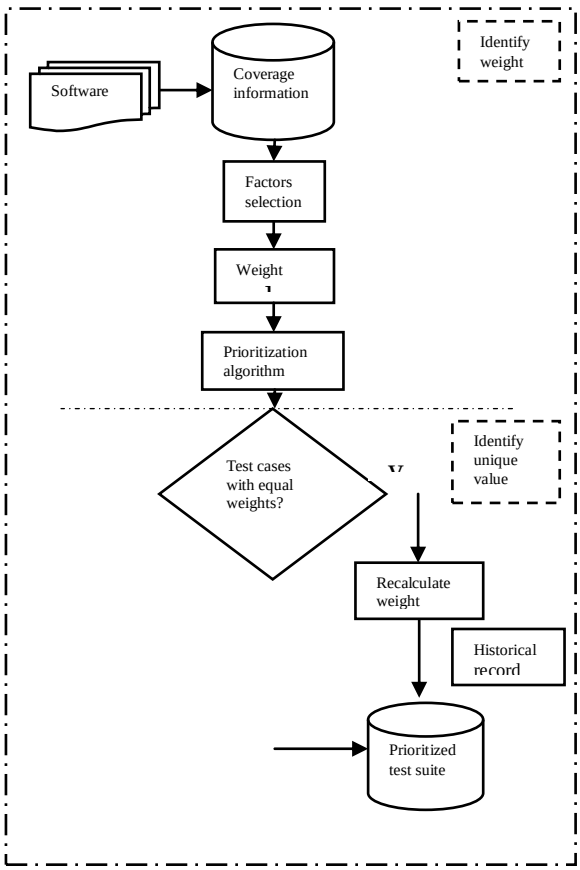

Fig. 2: Process of the enhanced weighted method UniVal

- Threats to internal validity: According to [35], threats to internal validity refer to the factors that can affect independent variables without a researcher's knowledge, which might put the causal relationship between the treatment and the outcome in question. In this research, the main concerns are the seeded faults and generation of test cases for object programs. To overcome this issue, an important criterion in choosing the object programs was whether it comes with seeded faults and test cases together which are generated by the tester/ researcher.
- Threats to construct validity: It refers to the extent to which the experiment setting truly reveals the construct under study. In this study, threats to construct validity were not considered critical. The dependent APFD measures used to evaluate the effectiveness had some limitations as they assumed that faults were with equal severities, and test cases were with equal costs. However, in this experiment, these assumptions were acceptable. Furthermore, it is a common measure for effectiveness of the prioritization methods and techniques.
- Threats to external validity: This is concerning the scope to which the outcome can be generalized. It is influenced by the selected experimental design and objects in the experiment. In this research, threats to external validity might be the selection of the object programs for the experiment that may not be representative of the population. However, the process of selecting the objects programs went through many criteria which made it difficult to choose the most appropriate program for this study. Besides, by comparing the number of samples (object programs) in this study with similar studies in this field, the threats to external validity were considered reasonable in generalizing the results.

4. Enhanced Weighted Method Using Unique Priority Value

The literature review shows that code coverage based technique is more effective in prioritizing test cases. Six factors employed in developing UniVal which are: statement coverage, function coverage, branch coverage, path coverage, fault coverage and test cases ranking from previous execution. Two types of equality in test case priority values appeared: (1) test cases covered the same segment of code, and (2) test cases covered different segments. For each situation, the enhanced weighted method used different formulas to generate the unique priority value. The process of the enhanced weighted method is shown in Figure 2.

4.1. Identify Weight

a) Factors selection

Five code coverage factors and history of previous execution were used to prioritize test cases. These factors are described below:

Statement coverage

This is the basic measurement of code coverage calculation process which depends on the number of statements covered by each test case. [3], [7], [10] and [25] used this factor to rank test cases.

Function coverage

It refers to the number of functions or methods executed completely or partially by a test case [6], [25] and [15].

Branch coverage

Branch coverage is also known as Decision coverage or all-edges coverage. It covers both the true and false conditions. A branch is the outcome of a decision, so branch coverage simply measures which decision outcomes have been tested. It was employed by [7] and [25].

Path coverage

Path coverage refers to performance of designing test cases such that all linearly independent paths in the program are executed at least once. A linearly independent path can be defined regarding to the control flow graph of an application [7] and [25].

Fault coverage

Fault coverage refers to the percentage of fault that can be detected during the test of any system. High fault coverage is particularly valuable during software testing [5], [36], [26] and [8].

History of previous execution

This factor refers to the documented record of the behavior of test cases from the last execution session. In this research, the previous order of test cases was employed to calculate unique priority values [24], [37], [38] and [17].

b) Weight calculation

As mentioned earlier, this method uses five coverage criteria with one criterion from the history of previous execution. For the purpose of illustrating the detailed unique priority value calculation, an object program of this experiment, which is the calculator program, was included within the weight calculation process. The weight was calculated as follows:

Statement coverage calculation

The weight WstTi for test case Ti was computed by dividing the number of statements Nst covered by test case Ti to the highest number of statements Mst covered by any test case, Ti.

$$W_{st}T_i = \frac{N_{st}}{M_{st}} \times 10 \quad (2)$$

Function coverage calculation

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

$$W_{fn}T_i = \frac{N_{fn}}{M_{fn}} \times 10 \quad (3)$$

Branch coverage calculation

The weight WbrTi for test case was calculated by dividing number of branches covered by test case Ti by the maximum number of branches covered by any test case Ti, then the weight for the test case, Ti.

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

Path coverage calculation

The weight for the path coverage WptTi for the test case Ti was calculated by dividing the number of paths covered by test case Ti to the maximum number of paths covered by any test case, Ti.

$$W_{pt}T_i = \frac{N_{pt}}{M_{pt}} \times 10 \quad (5)$$

Fault overage calculation

The weight for the fault coverage WflTi for the test case Ti was 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.

$$W_{fl}T_i = \frac{N_{fl}}{M_{fl}} \times 10 \quad (6)$$

c) Prioritization algorithm

For n test cases and m coverage criterion, the priority value for each test case Ti P was as follows.

$$T_iP = \frac{\sum_{j=1}^{m} T_i WX_j}{m} \quad (7)$$

Where: WXj = weight of criterion j

m = total number of criteria

Test cases were sorted based on their priority value, from the maximum to the minimum value. Table 2 shows the test cases weight at this stage before applying the unique priority value procedure.

Table 2: test cases order after coverage weight calculation

Test case	Priority value
T33	10
T34	10
T24	7.802
T27	7.802
T38	7.508
T37	7.308
T16	7.256
T20	7.256
T25	7.256
T26	7.256
T36	7.256
T15	7.056
T19	7.056
T21	7.056
T22	7.056
T35	7.056
T8	6.496
T14	6.31
T17	6.31
T7	6.296
T12	6.294
T28	6.148
T11	6.094
T23	5.948
T4	5.828
T32	5.828
T3	5.628
T31	5.628
T6	5.55
T5	5.35
T10	5.348
T39	5.298
T18	5.202
T9	5.148
T13	5.002
T2	4.882
T1	4.682
T29	1.368
T30	1.368
T40	1.368

4.2. Identify Unique Value

a) Test cases with equal weights

For this object program, many test cases recorded equal weight values.

b) Weight recalculation

UniVal continued the prioritization process with the following steps:

Groups for test cases with equal weight values were created.

Test cases within each group were rearranged based on the previous execution order.

It was noticed that when two or more test cases still have equal weight after the weight calculation, it could cover the same segment of the code. Based on this fact, there were two cases to deal with which are:

Test cases covered the same code segment

In this case, one of the test cases was chosen while the rest was delayed until the last test case in the initial order being executed. This helped in avoiding any redundancy issue when testing the same code segment. Thus, the first test case in the group kept its original weight. The other test cases within the group were delayed until the last test case in the initial order being executed. The formula for calculating the new weight for the amended test cases is:

$$T_iW_{new} = T_iW - (\frac{j}{n}) \quad (8)$$

Where:

T_iW = weight of the last test case in theinitial order

j =position of test case Ti within a group G

Then the test cases were re-ordered after each group weight was calculated.

Test cases covered different code segments

In this case, these test cases still had a higher priority over the next test cases in the table. Similar to the previous case, the first test case in the group kept its original weight. These test cases were given higher priority value than the next test case in the initial table. The order of test cases depended on the previous execution order, and the formula to calculate the test cases weight is:

$$T_iW_{new} = T_nW + (\frac{j}{n}) \quad (9)$$

Where:

T_nW = weight of the next test case in theinitial order

j = position of test case Ti within a group G

Then the test cases were re-ordered after each group weight was calculated.

The unique priority value for each test case was calculated using equations (8) and (9). The table was updated after the calculations of each group.

c) Prioritized test suite

After calculating the new priority values for all test cases, the enhanced weighted method sorted test cases in a descending order based on the new priority values. Table 3 shows test the priority values after applying the unique priority value procedure.

Table 3: Enhanced weighted method ranking with the unique priority values

Test case	Priority value	Unique Priority value
T33	10	10
T24	7.802	7.802
T38	7.508	7.508
T37	7.308	7.308
T16	7.256	7.256
T20	7.256	7.106
T15	7.056	7.056
T19	7.056	6.546
T8	6.496	6.496
T14	6.31	6.31
T7	6.296	6.296
T12	6.294	6.294
T28	6.148	6.148
T11	6.094	6.094
T23	5.948	5.948
T4	5.828	5.828
T3	5.628	5.628
T6	5.55	5.55
T5	5.35	5.35
T10	5.348	5.348

T39	5.298	5.298
T18	5.202	5.202
T9	5.148	5.148
T13	5.002	5.002
T2	4.882	4.882
T1	4.682	4.682
T29	1.368	1.368
T34	10	1.318
T27	7.802	1.268
T25	7.256	1.193
T26	7.256	1.168
T36	7.256	1.143
T21	7.056	1.068
T22	7.056	1.043
T35	7.056	1.018
T17	6.31	0.968
T32	5.828	0.918
T31	5.628	0.868
T30	1.368	0.818
T40	1.368	0.793

Hence, the priority value for each test case held a unique value.

4.3. APFD Calculation

To measure the effectiveness of the proposed method UniVal, a modified Average Percentage of Fault Detection (APFD) metrics was used. These measurements are listed below:

- Average percentage of statement coverage

$$AP_{st}C = 1 - \frac{\sum_{i=1}^{m} Tst_i}{nm} + \frac{1}{2n} \quad (10)$$

- Average percentage of function coverage

$$AP_{fn}C = 1 - \frac{\sum_{i=1}^{m} Tfn_i}{nm} + \frac{1}{2n} \quad (11)$$

- Average percentage of branch coverage

$$AP_{br}C = 1 - \frac{\sum_{i=1}^{m} Tbr_i}{nm} + \frac{1}{2n} \quad (12)$$

- Average percentage of path coverage

$$AP_{pa}C = 1 - \frac{\sum_{i=1}^{m} Tpa_i}{nm} + \frac{1}{2n} \quad (13)$$

- Average percentage of fault coverage

$$APFD = 1 - \frac{\sum_{i=1}^{m} Tf_i}{nm} + \frac{1}{2n} \quad (14)$$

The effectiveness of UniVal based on the calculator program is shown in Table 4.

Table 4: Units for Effectiveness of UniVal

	$AP_{st}C$	$AP_{fn}C$	$AP_{br}C$	$AP_{pa}C$	$APFD$
Calculator program	96.28	98.75	93.36	93	89.41

5. Effectiveness of the Enhanced Weighted Method

The goal of this section is to demonstrate statistically the effectiveness of the Enhanced weighted method by comparing it against the weighted method by [25] with respect to five measurements that were mentioned earlier using equations (10) to (14)
The experiment involved two main phases. The first phase collected coverage information for each test case using Jacocoverage tool, calculated test cases weight as well as generated the unique priority value for each test case under test suite. In the second phase, the effectiveness of the average percentage of coverage for each method was determined and the results were compared to determine the effectiveness of each approach in terms of the highest coverage percentage. The measurements of effectiveness comprised Average Percentage of Fault Coverage APFD and the derived measurements.
Eventually, a paired t-test was conducted to analyze the differences between the population means for the two methods for each criterion. The results of the statistical test provided an accurate and deep intuitive understanding into the effectiveness of the two methods of test case prioritization.

5.1 Experimental Results

The argument of the outcomes is based on a quality factor, which is the effectiveness of each method in prioritizing test cases. The assessment is relying on the capability of the weighted and enhanced weighted method in prioritizing test cases and recording a better coverage rate. In the general assessment, both methods used the same set of test cases in evaluating effectiveness. Since the advanced assessment is significant to this research, statistical tests were conducted to compare the distribution of data collected. The complete results were calculated using equations (10) to (14). The complete results are shown in Table 5 and Figures 3 to 7.

Table 5: effectiveness results (all measurements in a percentage % form)

	APcdC		**APfnC**		**APbrC**		**APpaC**		**APFD**	
	WM	UV	WM	UV	WM	UV	WM	UV	WM	UV
Queue	92.1	95.6	98.48	98.48	91.72	93.57	94.5	96.45	76.36	86.63
calc	94.44	96.28	98.75	98.75	89.9	93.36	89.1	93	84.08	89.41
ATM	93.5	95.5	94.5	98.75	85.95	87.9	88.19	92.42	69.59	80.3

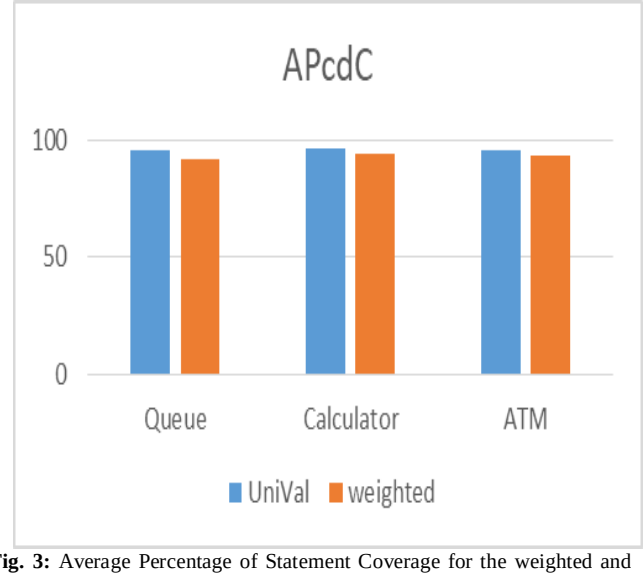

Fig. 3: Average Percentage of Statement Coverage for the weighted and enhanced weighted methods

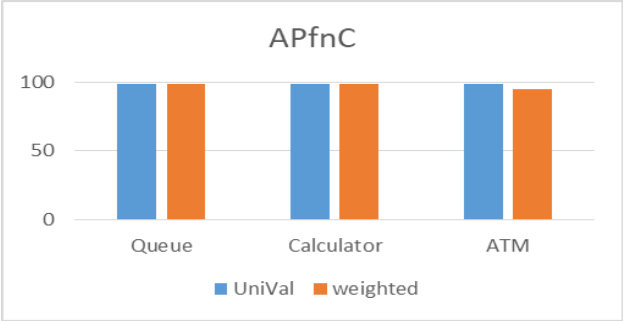

Fig. 4: Average Percentage of Function Coverage for the weighted and enhanced weighted methods

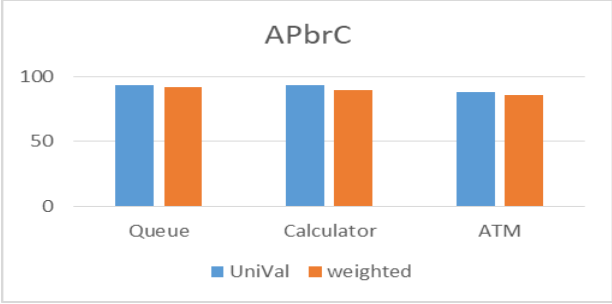

Fig. 5: Average Percentage of Branch Coverage for the weighted and enhanced weighted methods

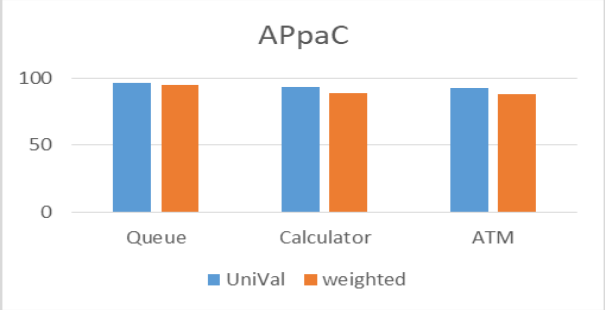

Fig. 6: Average Percentage of Path Coverage for the weighted and enhanced weighted methods

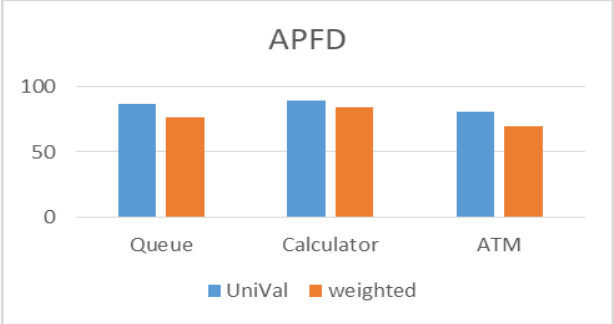

Fig. 7: Average Percentage of Fault Coverage for the weighted and enhanced weighted methods

The bar graphs indicate the difference between the weighted method and the enhanced weighted method with regard to effectiveness. The graphs show that the enhanced weighted method comprises the sector with a higher percentage over the weighted method. However, both of them are almost the same in terms of Average Percentage of Function Coverage. It is obvious that for two out of three object programs, the two methods recorded the same percentage of 98.48 and 98.75, respectively. The reason behind these results is that these two programs have relatively few numbers of functions that could be covered by one test case. Hence, if this test case recorded the highest coverage number, it would have the highest priority based on the prioritization methods. Therefore, the two methods show the same trend. When an object program has a larger number of functions as in ATM, the enhanced weighted method shows a higher percentage over the weighted method.

The most significant difference between the two methods is recorded by the Average Percentage of Fault Coverage APFD for ATM and Queue, respectively. On the other hand, it shows the lowest percentage among the five coverage criteria. For the weighted method, the lowest percentage was scored for ATM, with 69.59% when it was 80.3 for the same object program recorded by UniVal.

To prove this claim, a statistical test was carried out for the five coverage criteria. Five hypothesises were tested using paired t-test, a parametric test, to determine if the enhanced weighted method could have significant improvements in terms of coverage criteria over the weighted method. The paired t-test is a parametric statistical hypothesis test for any population distribution of two samples when the assumption of independent samples is intentionally violated. The distribution of the collected data that was normally distributed led to the selection of the parametric statistical test. The paired t-test was selected because the two samples were not independent and were in the form of ratio scales. Furthermore, it is well-known that the paired t-test is most appropriate when the sample size is small (i.e. < 100), which also applicable to this study.

The statistical test was applied for the average percentage of coverage. In order to to see if there is a difference in the two population means µ1-µ2, the null and alternative hypotheses were formulated as shown previously in section 3.3.

The first step was to calculate the difference between the two methods, next was the calculation of the mean, standard deviation of differences, t- distribution and significance level α=0.05. Summary of results of paired t-test is shown in Table 4.

Table 4: paired t-test results

measurement	df	$\bar{d}$	s_d	t	Critical t values	Reject H_0?	P value
$APsaC$	2	2.44	0.9	4.69	4.303, -4.303	Yes	0.0213
$APfnC$	2	1.416	2.45	1.001	4.303, -4.303	No	0.211
$APbrC$	2	2.42	0.902	4.64	4.303, -4.303	Yes	0.022
$APpaC$	2	3.36	1.23	4.72	4.303, -4.303	Yes	0.021
$APFD$	2	8.65	2.87	5.22	4.303, -4.303	Yes	0.017

The null hypothesis could not be rejected with regard to the average percentage of function coverage as no differences were recorded between the methods under comparison for the two object programs. However, based on the results obtained from the statistical tests, there is sufficient evidence to reject the null hypothesis at 0.05 significance level for four out of five measurements. Hence, it is statically proven that the enhanced weighted method is more effective than the weighted method in terms of average percentage of coverage. At this point, the results show that the research objective has been achieved since the enhanced weighted method shows better performance over the weighted method by generating unique priority values.

6. Conclusion

This study presents the enhanced weighted method for prioritizing test cases by assigning unique priority value to each test case. The method ranks test cases using code coverage criteria and the history of previous execution. The code coverage criteria are the main elements in achieving better effectiveness in term of average percentage of coverage while the history of previous execution helps

in assigning unique priority values. By having a unique priority value, the random sorting technique could be avoided.

Acknowledgement

The authors gratefully acknowledge the Ministry of Higher Education Malaysia (MoHE) for the financial support under the Fundamental Research Grant Scheme (FRGS); Project code-08-01-15-1723FR.

References

[1] Beizer, B. (1990). *Software Testing Techniques (2Nd Ed.)*. New York, NY, USA: Van Nostrand Reinhold Co.

[2] Chaudhary, N., Sangwan, O. P., & Singth, Y. (2012). Test Case PrioritizationUsing Fuzzy Logic for GUI based Software. *International Journal of Advanced Computer Science and Applications(IJACSA)*, *3*(12), 222–227. Retrieved from http://ijacsa.thesai.org/

[3] Cho, Y., Kim, J., & Lee, E. (2016). History-Based Test Case Prioritization for Failure Information. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)* (pp. 385–388). IEEE. https://doi.org/10.1109/APSEC.2016.066

[4] Collis, J., & Hussy, R. (2003). *Business research: a practical guide for undergraduate and postgraduate students / Jill Collis and Roger Hussey*. *- Version details - Trove*. Retrieved from http://trove.nla.gov.au/work/13991820?q&sort=holdings+desc&_=1494530661049&versionId=220777880

[5] Elbaum, S., Rothermel, G., Kanduri, S., & Malishevsky, A. G. (2004). Selecting a cost-effective test case prioritization technique. *Software Quality Journal*, *12*, 185–210. https://doi.org/10.1023/B:SQJO.0000034708.84524.22

[6] Fazlalizadeh, Y., Khalilian, A., Abdollahi Azgomi, M., & Parsa, S. (2009). Incorporating historical test case performance data and resource constraints into test case prioritization. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5668 LNCS*, 43–57. https://doi.org/10.1007/978-3-642-02949-3_5

[7] Felderer, M., & Fourneret, E. (2015). A systematic classification of security regression testing approaches. *International Journal on Software Tools for Technology Transfer*. https://doi.org/10.1007/s10009-015-0365-2

[8] 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), 53–60. https://doi.org/10.5121/ijcsea.2012.2305

[9] Harikarthik, S. K., & Palanisamy, V. (2014). Improving Quality of Software Testing Process by Test Case Prioritization, 245–248.

[10] Hashini, M., & Varun, B. (2014). Clustering Approach to Test Case Prioritization Using Code Coverage Metric, (May), 3–6.

[11] Jorgensen, P. C. (2014). *Software Testing A Craftsman's Approach* (4th ed.). crc press. Retrieved from http://202.191.120.147:8020/greenstone/collect/ebooks/index/assoc/HASH017c.dir/doc.pdf

[12] Jyoti, & Solanki, K. (2014). A Comparative Study of Five Regression Testing Techniques : A Survey. *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*, *3*(8). Retrieved from www.ijstr.org

[13] Kaur, N., & Mahajan, M. (2014). Prioritization of Test Cases using Branch Coverage with Multiple criteria for Regression Testing, *5*(2), 972–974.

[14] Khalilian, A., Abdollahi Azgomi, M., & Fazlalizadeh, Y. (2012). An improved method for test case prioritization by incorporating historical test case data. *Science of Computer Programming*, *78*(1), 93–116. https://doi.org/10.1016/j.scico.2012.01.006

[15] Khandelwal, E., & Bhadauria, M. (2013). Various Techniques Used For Prioritization of Test Cases, *3*(6), 3–6.

[16] Korel, B., Koutsogiannakis, G., & Tahat, L. H. (2007). Model-based test prioritization heuristic methods and their evaluation. *Proceedings of the 3rd International Workshop on Advances in Model-Based Testing - A-MOST '07*, 34–43. https://doi.org/10.1145/1291535.1291539

[17] Kumar, A., & Singh, K. (2014). A Literature Survey on test case prioritization. *COMPUSOFT*, *3*(5). Retrieved from https://ijact.in/index.php/ijact/article/viewFile/320/271

[18] Lawanna, A. (2012). The Theory of Software Testing, *16*(1), 35–40.

[19] Mei, L., Zhang, Z., Chan, W. K., & Tse, T. H. (2009). Test case prioritization for regression testing of service-oriented business applications. *Proceedings of the 18th International Conference on World Wide Web - WWW '09*, 901. https://doi.org/10.1145/1526709.1526830

[20] Muthusamy, T. (2014). A New Effective Test Case Prioritization for Regression Testing based on Prioritization Algorithm, *6*(7), 21–26.

[21] Muthusamy, T., & Seetharaman, K. (2013). A Test Case Prioritization Method with Weight Factors in Regression Testing Based on Measurement Metrics, (12), 390–396.

[22] Muthusamy, T., & Seetharaman, K. (2014). Comparisons of Test t Case Prioritization Algorithm with Random Prioritization, *5*(5), 6814–6818.

[23] Orso, A., & Rothermel, G. (2014). Software testing: a research travelogue (2000–2014). *Proceedings of the on Future of Software Engineering - FOSE 2014*, 117–132. https://doi.org/10.1145/2593882.2593885

[24] Ponaraseri, S., Susi, A., & Tonella, P. (2008). Using the Planning Game for Test Case Prioritization. *Test*.

[25] Prakash, N. (2013). Potentially Weighted Method for Test Case Prioritization, *18*, 7147–7156. https://doi.org/10.12733/jcis5860

[26] Prakash, N., & Rangaswamy, T. R. (2013). WEIGHTED METHOD FOR COVERAGE BASED TEST CASE PRIORITIZATION, *56*(2), 235–243.

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

[28] Rothermel, G., Untch, R. H., Chu, C., Harrold, M. J., & Society, I. C. (2001). Prioritizing Test Cases For Regression Testing Prioritizing Test Cases For Regression Testing. *IEEE Transactions on Software Engineering*, *27*(10), 929–948. https://doi.org/10.1145/347324.348910

[29] Shadish, W. R., Cook, T. D., & Campbell, D. T. (2002). *EXPERIMENTAL AND QUASI-EXPERIMENTAL DESIGNS FOR GENERALIZED CAUSAL INFERENCE*. Retrieved from http://impact.cgiar.org/pdf/147.pdf

[30] Shelke, S., & Nagpure, S. (2014). The Study of Various Code Coverage Tools. *International Journal of Computer Trends and Technology (IJCTT)--Volume*, *13*(1), 46–49.

[31] Srikanth, H., Banerjee, S., Williams, L., & Osborne, J. (2014). Towards the prioritization of system test cases. *Software Testing, Verification and Reliability*, *24*(4), 320–337. https://doi.org/10.1002/stvr.1500

[32] Tanwani, L., & Waghire, A. (2016). Test Case Prioritization for Regression Testing of GUI. *International Acadmey of Engineering and Medical Research*, (1). Retrieved from http://www.iaemr.com/wp-content/uploads/2016/11/test-case-prioritization-regression-testing-gui.pdf

[33] Tonella, P., Avesani, P., & Susi, A. (2006). Using the case-based ranking methodology for test case prioritization. *IEEE International Conference on Software Maintenance, ICSM*, 123–132. https://doi.org/10.1109/ICSM.2006.74

[34] Wang, X., & Zeng, H. (2014). Dynamic Test Case Prioritization based on Multi-objective, (61073050), 0–5.

[35] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29044-2

[36] Yadav, D. K., & Dutta, S. (2017). Regression Test Case Prioritization Technique Using Genetic Algorithm (pp. 133–140). https://doi.org/10.1007/978-981-10-2525-9_13

[37] Yoo, S., & Harman, M. (2007). Regression Testing Minimisation, Selection and Prioritisation : A Survey. *Test. Verif. Reliab*, *0*, 1–7. https://doi.org/10.1002/000

[38] Zou, Y., Chen, Z., Zheng, Y., Zhang, X., & Gao, Z. (2014). Virtual DOM Coverage : Drive an Effective Testing for Dynamic Web Applications Categories and Subject Descriptors. *Issta*.