

A PROPOSED FRAMEWORK FOR TEST SUITE PRIORITIZATION AND REDUCTION USING THE CLUSTERING DATA MINING TECHNIQUE

¹ KHALID ELDRANDALY, ² MAHMOUD ABD ELLATIF, ³ NORA ZAKI

¹ Professor of Information Systems, Faculty of Computers and Informatics, Zagazig University.

² Professor of Information Systems, College of Business, Jaddah University, SA & Faculty of
Computers and Informatics, Helwan University.

³ Assistant Teacher, Faculty of Computers and Informatics, Zagazig University.

E-mail: ¹eldrandaly66@gmail.com, ²mmlatif@uj.edu.sa, ³omarsabryel@gmail.com .

Abstract

Software testing is one of the most critical phases of the software development life cycle. The primary purpose of software testing is to check the produced program or application before delivering it to the target customer and to discover the hidden faults or errors that lead to system failure. The time and cost consumed by software testing are one of the most critical limitations of software testing. The most consumed time in software testing results from executing a large number of redundant and inefficient test cases. Therefore, the automatic generated test cases should be filtered before executing them. Test suite reduction TSR and test suite prioritization TSP are considered as a management method for test suites. They provide efficient management for test suites by reducing and prioritizing the number of test cases. The main goal of this research is to propose a framework for improving the software testing process by using the clustering-based test suite prioritization and reduction techniques. The main objective of the proposed framework is to generate an optimal set of test cases from the original set. The optimal set of test cases can be efficiently executed in less time and cost. A case study is conducted to estimate the performance of the proposed framework. The results show that the proposed framework is robust and valuable for software testing process under the limited time and provides testers with some guides to obtain maximal benefit of the proposed framework.

Keywords: *Test Suite Reduction, Test Suite Prioritization, Code Coverage, Clustering Data Mining, Proposed Framework.*

1. INTRODUCTION

Test suite reduction and test suite prioritization are the most Challenge of software testing. Especially when time and cost are limited. Test suite reduction techniques are based on applying selective approaches to pick a smaller number of test cases from the original test suite to generate a reduced set (RS) of the test suite. Wherever, the test cases in the reduced set should cover all the requirements as covered by the original one. Although, during the reduction process, it is possible to eliminate test cases that have a high probability of fault detection. Test suite prioritization techniques overcome this limitation by presenting test cases in the best order to improve the rate of fault detection [1], [2].

cost. Therefore, they applied reduction techniques to select a small set of test cases that cover the

This research proposes a framework for managing the test suite by investigating the test suite prioritization and reduction techniques where the priority set (PS) of test cases is determined first. Then the reduced set (RS) of test cases is generated from the priority set in order to optimize the rate of fault detection. This combination of test suite prioritization and reduction techniques would be valuable for testers who are under pressure because of the limited time and cost of the testing process.

Recently, some studies have begun to combine test suite prioritization and reduction techniques. [3] Supposed a strategy to order test cases in the reduced test suite to increase the effectiveness of the testing process when testing stops suddenly because of the limited time and testing requirement from the original set and then order the reduced test cases. They confirmed that

such an order is likely to improve the rate of fault detection. Also, it guarantees that the most effective test cases in the reduced suite would already be executed in the case of early stopping of test execution. They considered four well-known reduction heuristics and applied them on two real-world applications, and they used APFD to measure their rate of fault detection. However, they did not show how the order of test cases in a reduced suite is performed. Furthermore, they compared between test suites of the same size. Therefore, the used APFD metric was suitable for their evaluation. However, APFD metric may be unsuitable for comparing the rate of fault detection for test cases of different sizes.

In [4], they inspected the previous strategy for a web application's domain. However, they applied several prioritization approaches criteria that are experimentally verified in web application. They tried to order the test cases in RS based on prioritization. Further, they developed a new metric *Mod_APFD_C* that is used to measure the rate of fault detection for test cases of different sizes. The results ensured that the rate of fault detection was improved when an ordered reduced test suite was executed, especially when the tester does not have sufficient time to run all test cases in RS. Also, the tester can get the most effective test cases for early execution.

This research differs from these works; because they focus on applying the prioritization techniques for RS. But, this research focuses on using the reduction techniques for PS. It hybrid the prioritization techniques and reduction techniques in one component in the proposed framework. So that, the prioritization and reduction process is performed respectively not separately. However, this work is similar to the work in [5] where they focused on prioritization of user session-based test cases for the web application. They proposed a tool that is called CPUT. It is implemented by applying the prioritization techniques that is experimentally verified effectiveness in even driven software. They implemented CPUT by four prioritization criteria to allow the user to select one of these four criteria. However, in the proposed CPUT, the reduction process is optional for the user where he can choose whether to reduce PS or not. The prioritization and reduction process in CPUT tool are performed separately. Therefore, this is the main difference between the proposed framework and the proposed tool CPUT. Although, CPUT tool used different criteria to complete the prioritization process such as (Length Gets/Posts, Number of parameters,

2way combinatorial, and random). Whereas, the proposed framework uses other criteria for the prioritization process.

Furthermore, the proposed framework is based on applying coverage-based TSP and TSR techniques because coverage metrics are valuable to assess the quality of software testing. The code coverage metric is widely used to measure the percentage of code that has been covered and tested by a test case. Therefore, the proposed framework is similar to the work in [6] where they proposed TestOptimizer framework that is based on applying a code coverage based TSR and TSP techniques. They combined *TestFilter* and *St-Total* techniques to generate optimal test cases. Further, the proposed TestOptimizer includes an optimizer component that is responsible for computing the optimization time based on the total time that is required for running the implemented TSP or TSR techniques. Based on the allowed optimization time, the optimizer decides which test suite management to implement: TSP, or TSR, or TSP/TSR, or TSR/TSP.

Although, the proposed framework investigates data mining techniques by applying clustering-based TSP and TSR techniques. Clustering technique is the most useful data mining that is widely used for test case reduction and test-suite prioritization due to its ability to reduce the size of the test suite by removing the redundancy from the test suite. Therefore, the proposed framework is different from other works that are tried to combine TSP and TSR techniques because this work integrates between TSP and TSR where the priority of test cases is determined at the first step, and then the RS is generated from PS. An optimal set of test cases with a high percentage of fault detection will be created as a result of this combination. This optimal list of test cases is beneficial for execution under a limited time constraint. Moreover, TSP and TSR are implemented by applying the most active code coverage-based TSP and TSR techniques that use clustering data mining technique.

The Contribution of this research:

- 1- A proposed model that orders and reduces test suites by using multiple coverage criteria and different prioritization criteria.
- 2- An empirical comparison between the optimal set of test cases against the original set of test cases using some measurement metrics.
- 3- Provide a guide to testers on which the best prioritization criteria to be selected based on observed results of the empirical study.

2. BACKGROUND AND RELATED WORK

2.1 Test Suite Reduction (TSR):

The automatic generation of the test cases results in the vast number of test cases. Therefore, the test suite size becomes too long and may contain unnecessary and redundant test cases. At this point, test cases are difficult to be managed and costly to be executed [2]. Test suite size problem has been addressed by proposing a test suite reduction TSR approach [7] that also named test suite minimization approach. The goal of this approach is to select a minimal subset from the original test suite that covers a set of test requirements. The definition of test suite reduction problem can be summarized as follow [8]:

Given: a test suite TS that contains a set of test cases $\{tc1, tc2, tc3, \dots, tcn\}$ that can cover a set of requirements $Req = \{R1, R2, R3, \dots, Rn\}$.

Problem: find the minimal subset $RS \subseteq TS$ that satisfies all of the requirements Req where each requirement can be covered by at least one test case.

The test suite reduction is an essential challenge in software testing because the test suite size directly affects the time and cost of the testing process and hence, its effectiveness. Therefore, it is useful and profitable to have as a small set of test cases as could reasonably be expected [9], [10]. The number of test suite reduction techniques have been widely proposed and experimented. The main objective of TSR techniques is to reduce the size of the test suite based on some criteria to limit the executing time and its costs. The optimal minimal test suite should be satisfied with some criteria, such as maximum coverage, high fault detection, and minimum execution time [11].

The data mining techniques play a vital role in test reduction due to its ability to extract hidden patterns of test cases by detecting the similarities between the test cases and deleting the redundant ones [12], [13]. Many different studies use data mining techniques in test suite reduction. The most applied data mining technique to test suite reduction is a clustering approach [2], [12], [14], [15].

2.1.1 Clustering-based test suite reduction technique:

Cluster analysis is an essential domain in data mining. The primary task of the clustering analysis in test case reduction is to divide a set of test cases within single test suite into clusters where each cluster contains the test cases that have the same or

similar characteristics and different from the test cases in other clusters [9]. The cluster algorithms divide the set of test cases based on the similarity or dissimilarity metrics that are used to measure the similarity between test cases. These metrics such as Minkowski metric, Euclidean distance and Supermun distance [16].

Clustering technique is the most useful data mining that is widely used for test case reduction due to its ability to eliminate the redundancy from the test suite and reduce the size of the test suite. Clustering-based test case reduction techniques can be classified into Coverage based test case reduction, Similarity-based test case reduction, and Density-based test case reduction [17].

2.1.2 Coverage based test case reduction techniques:

Code coverage is a useful metric to assess the quality of software testing. It is used to measure the percentage of code that has been covered and tested by a test suite. Code coverage can be measured by using many metrics such as [12]:

1. **Function coverage:** the number of called function or subroutine in the code.
2. **Statement coverage:** the number of executed statements in the code.
3. **Branch/ path coverage:** the number of IF statements that have both true and false path.
4. **Conditional / decision coverage:** the number of Boolean expressions that are evaluated to both true and false.

The test case that is able to exercise the maximum percentage of the code in the software under test SUT is efficient test case. The output of code coverage measurement provides information about the area that is not covered and also this information can be used to direct the test generators to create a test case that able to test area that has not been covered before [18]. Therefore, code coverage is widely used in regression testing, test case reduction, and test case prioritization.

The main objective of coverage-based test suite reduction techniques is to reduce the size of the test suite by eliminating the redundant test cases and select the optimal test case that has a maximum percentage of code coverage. The most proposed coverage-based test case reduction techniques use K-Means clustering algorithm concerning different code coverage metrics. In this regard, [19] proposed a technique for test case reduction by using the K-Means algorithm based on branch coverage. They

performed white box testing and black-box testing by generating a set of test cases by using Pex software tool. Then, they applied K-Means clustering to reduce the number of test cases based on branch coverage as the code coverage metric. The results show that the K-Means clustering reduces the test suite size and keeps the same coverage in reduced test suit. Other studies [20], [21], [22] also used the K-Means clustering algorithm but, for path coverage.

An empirical study presented in [23] suggested that using multi coverage criteria rather than single coverage is more effective in selecting test cases that can expose different faults. Therefore, some studies use more than one coverage metric to improve fault detection capability. In [24], they proposed an approach for test case reduction based on multi coverage criteria such as function, statement, and branch. They applied hierarchical clustering algorithm. The proposed approach can reduce the size of the test suite and at the same time, maintaining the ratio of code coverage and fault prediction capability of the reduced suite. The limitation of the proposed approach is the complexity of the hierarchical clustering algorithm.

On the other side, [12] proposed the same approach for test case reduction to decrease the time and cost of executing them. But, they used a k-mean clustering algorithm instead of hierarchical clustering to group several test cases into clusters and then remove the redundant test cases that have equal distances to cluster centre point. The proposed approach applied for test case reduction based on the most two attributes of test cases: coverage (branch, path, and method coverage) and cyclomatic complexity CC. They also used multi coverage criteria in addition to CC attribute of test cases. The advantage of using two different attributes of test cases (coverage and CC) rather than single attribute (coverage) is to increase the chance of selecting test cases that have a high ability to detect different faults. So, their proposed approach can reduce the size of the test suite by removing only the test cases that are not necessary for testing. Also, the coverage after the reduction process still offered good results.

The advantage of the coverage-based test suit reduction techniques is the high rate of test suit reduction and the saved time [17]. Therefore, the coverage aspect is a vital issue that needs to be considered in test suite reduction. So, the proposed framework is built on code coverage-based test suit reduction technique that uses multi coverage criteria rather than single coverage. Although, the

main limitation of the coverage-based techniques is that the path coverage consumes more time to calculate the coverage from source code, especially in the extensive system [8].

2.1.3 The benefit and limitation of test case reduction techniques:

As evident, using test-suite reduction (TSR) techniques have a good impact on the effectiveness of the testing process in terms of time and cost. They help the manager to (1) execute the test cases in less time. (2) Observe the test results. (3) Handle the testing data.

Although, the primary limitation of these techniques is the significant decrease of fault detection capability because some of the removed test cases that are eliminated during the reduction approach have a latent ability to reveal the faults. Also, the rate of fault detection of the reduced test suite may be less than the rate of fault detection of the original test suite [13]. Therefore, the tradeoff between the size of test cases and their fault detection capability must be taken into account when applying test suite reduction techniques. To overcome this limitation, test suite prioritization (TSP) techniques should be used to improve the rate of fault detection.

2.2 Test Suite Prioritization (TSP):

Another real challenge of software testing is the order by which test cases are executed. Test suite prioritization TSP [25] aims to schedule test cases in perfect order so that some objective function such as rate of fault detection can be maximized during the execution. The main goal of TSP is to execute the test cases that have the maximum fault detection capability at a first position to minimize the cost and time of execution by early detection of faults. TSP provides many advantages: increasing the rate of fault detection, reducing the costs of testing, increasing the reliability of SUT, and increasing the coverage of the code [26].

Many researchers proposed many techniques for test case prioritization [27], [1], [28] to help the testers to solve TSP problem. The main goals of TSP techniques are to enhance the fault detection rate and reduce the time and cost of execution. Most proposed techniques prioritize the test cases based on the code coverage criteria [29] where the test cases can be ordered based on the number of statements, or the number of functions, or a number of blocks that are covered by each test cases. Also, several studies [30], [31] that have been applied code coverage based prioritization

techniques showed good results for the

To evaluate the rate of fault detection that is considered as the essential objective function of TSP techniques, the number of faults that are detected by each test case should be measured. Different metrics are used to measure how prioritized test cases revealed faults. APFD (Average Percentage of Faults Detected) is a popular metric that is used for a long time for measuring the rate of fault detection [32]. The values of APFD metric are ranged between 0 to 100, where the maximum value of APFD means a high rate of fault detection. The limitation of APFD metric is it assumes that all faults have the same severity and all test cases have the same costs.

The severity and costs of faults may vary according to different factors of the testing environment. Therefore, [33] presented a new metric to overcome the limitation of APFD metric. The proposed metric is termed as APFD per cost (APFD_c). It considers the different severity of test cases and different costs of faults. Other metrics are introduced, such as saving factor (SF), normalized APFD (NAPFD), the average severity of faults detected (ASFD), etc. Catal and Mishra concluded the distribution of these metrics through systematic study. They stated that APFD metric is quite dominant of all wherever it is used by 34% of the chosen papers [34].

2.2.1 Clustering-based test suite prioritization techniques:

Clustering analysis plays a vital role in test case prioritization as well as in test case reduction. It enhances the rate of fault detection and execution time. So that the most important test cases will be identified and executed early. Several approaches for clustering-based test suite prioritization are proposed [35], [36], [37]. The basic concept of these approaches is based on applying a clustering data mining technique to test suite and then prioritize test cases in each cluster based on different types of information. The most types of information that are used by the proposed TSP techniques are:

1. Code coverage: is one of the basic metrics that is used to measure the effectiveness of test cases. It shows how many lines, statements, branches, and decisions are covered by each test case. Code coverage is mostly used in TSP techniques due to its simplicity and its effectiveness where the better coverage gives a greater rate of fault detection [38], [18].

effectiveness of regression testing.

2. Cyclomatic complexity (CC) [39]: is another metric that used to measure the number of independent paths through a given method in the program. This number of independent paths determines the maximum number of test cases that should be executed to test the program code. Cyclomatic complexity can be calculated manually from the flow chart or flow graph that is developed from the source code. It is calculated using the following equation: $CC = E - n + 2$, Where E is the number of edges and n is the number of nodes. Another method for CC calculation is: $CC = (\text{the number of loops} + \text{the number of conditionals}) + 1$

3. Fault history information: it contains information about the number of faults that are detected by each test case during the first execution of SUT. Each executed test case and its related fault history information are stored in the test repository. [29] Assumed that if test cases have detected faults in a previous execution, then it could have a high possibility to detect faults when they will be re-executed in the current version of SUT. They tried to explore this assumption by implementing TSP technique based on the fault history information of test cases. The results ensured that fault detection history is vital information for TSP techniques and could improve the effectiveness of the prioritization process. Furthermore, the fault history information does not require prior knowledge of source code; unlike coverage, require detailed analysis of source code.

Recently, some researches use other types of information such as requirement coverage prioritization, cost-aware prioritization, time aware prioritization and model-based prioritization. Also, [29] used the combination information of code complexity and fault history information to prioritize the test cases per clusters. The effectiveness of TSP technique can be improved by using two sophisticated information such as code complexity and fault history information. Mukherjee and Patnaik summarized the different approach of TSP through a systematic survey by reviewing 90 scholarly articles from 2001 to 2018 [40].

Among this different information, the proposed framework uses the code coverage, cyclomatic complexity, and fault history information for prioritization process because this three information is familiar to the tester and can be collected and calculated easily as well as their effective relation to test cases.

3. THE PROPOSED FRAMEWORK:

The proposed framework is inspired by [5], [6], [12], [16], [19], [22], [29]. It supports test-suite prioritization and reduction using clustering data mining technique. It helps the tester to generate the optimal set of test cases that can be executed in less time and cost. The optimal set should achieve higher detection of faults with maximum code coverage. The optimal set of test cases is generated by applying the clustering-based TSP and TSR technique to the original set of test cases. Then, selecting the high order test cases that satisfy the user requirements and covers most of the code with minimal repetition of test cases.

The proposed framework can be applied for any software application. It only needs some prior information from the test leader about code coverage and fault history information. Then, it automatically handles a vast number of test cases according to coverage and prioritization criteria. Figure1 shows the high-level structure of the proposed framework that consists of four basic components: (I) Test repository (II) Organizer (III) TS-clustering (IV) Prioritizer and Reducer (PR) engine. The issues and design options of these components are discussed as follow:

I. Test repository:

It contains essential information that has been driven by test leader such as test suites, source code of SUT, coverage criteria and fault history information. It also includes the gathered information that is calculated by software tools such as code coverage and cyclomatic complexity. All this information is retrieved by the Organizer component to complete the function of the proposed framework.

II. Organizer:

It monitors and organizes the prioritization and reduction process of test cases by retrieving the needed information from the test repository and then calling the clustering mechanism, and the PR engine, respectively.

III. TS-clustering:

The main purpose of this component is to classify the input data set (test cases) into the number of clusters. It is implemented by applying the K-means clustering algorithm. This algorithm is considered one of the simplest and most popular unsupervised learning algorithms for data clustering. It is used extensively in practical implementation due to its simplicity. K-means clustering algorithm follows a simple and straightforward approach. First, it groups the input

data sets into K groups where “K” is initially selected to represent cluster centres. After that, the distance between cluster centres and each data points are measured. All data points can be moved to the cluster centre whose distance to the centre is the nearest. After that, the new cluster centre can be calculated by the mean value of the data in each group and again the distance value between each data points and new centres is recalculated in order to assign the data points to the closest cluster centre. Summarize of K-means clustering algorithm [39]:

1. Select K as initial cluster centres randomly;
2. Calculate the distance between each data points and cluster centres;
3. Assign each data points to the nearest centres;
4. Recalculate the new centres of each cluster;
5. Repeat from step2 until no data points are reassigned;

The necessary inputs to this procedure are the set of test cases and “K” initial specified number of clusters. While the output is the number of clusters where each cluster contains similar test cases that have the same features, this means that each cluster displays some redundant test cases. The problem of test cases repetition can be handled by the PR engine.

IV. The PR engine:

It is the core component of the proposed framework. It handles the problem of test case reduction and prioritization by combining clustering-based TSP and TSR techniques. Therefore, the PR engine includes clustering-based TSP and TSR techniques that have been experimentally validated and proven effective in test case prioritization and reduction problem. The main objective of the PR engine is to represent the test cases in optimal order without redundancy for effective execution. The PR engine performs two basic processes: the prioritization process and reduction process

1. The prioritization process:

This process is initiated after creating the clusters of test cases to order the test cases per each cluster based on some criteria with the objective function of early detection of faults. The prioritization process is implemented by three prioritization criteria (code coverage, cyclomatic complexity, or fault history information) that are mentioned previously in section 2.2.1. The proposed framework allows the tester to select one from the three prioritization criteria based on the available information about code coverage and fault detection. The ranked test cases per clusters are

exported to the buffer until the reduction process is called by the organizer.

2. The reduction process:

This process is initiated after prioritizing the test cases per cluster. The objective of this

process is to eliminate the redundant test cases from each cluster by selecting unique and high-level priority test case from each cluster. So, in order to

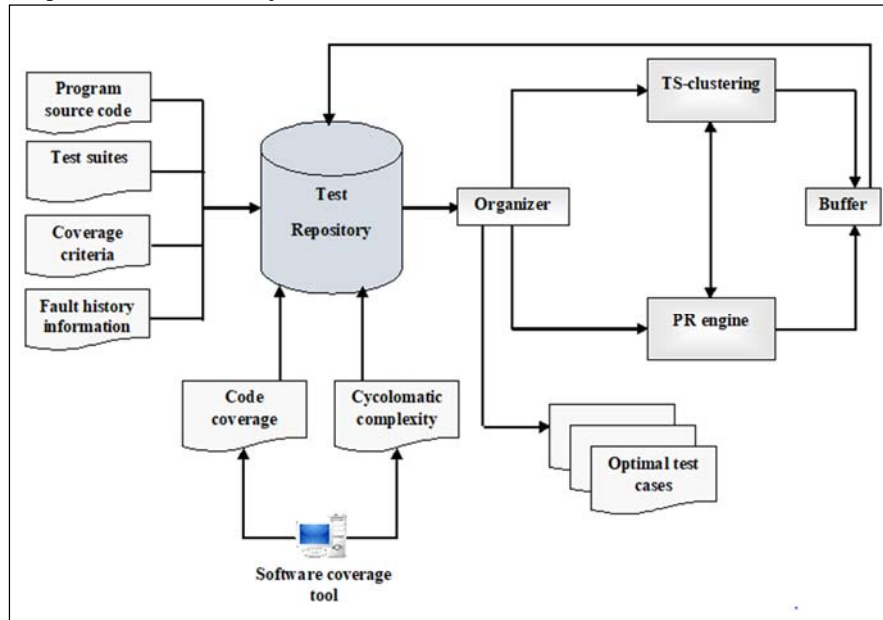


Figure 1. The Proposed Framework.

select the high-level priority test case, an effective selection approach is implemented in the reduction process.

The selection approach visits each cluster to pick the first-order test case. The selection approach may visit the clusters in a random order of the clusters or in the primary order in which the clusters are generated by clustering tool. But, to optimize the priority of test cases, a feature is added to the PR engine where the clusters are ranked in an order based on the selected prioritization criteria. Then the selection approach visits the clusters in the new order. This feature enables the PR engine to produce an optimal set of high order test cases. The picked test cases from each cluster are exported to the buffer.

3.1 Assumption:

For proper operation of the proposed framework, the following assumption must be taken into account when implementing the proposed framework:

ASS1: unavailability of time to execute more test cases and fault detection is a critical issue for a test manager.

ASS2: apply prioritization process before the reduction process to optimize the rate of fault detection.

ASS3: the prioritization process can be implemented by any proposed TSP techniques as well as the reduction process can be performed by any proposed TSR techniques.

3.2 The Function Of The Proposed Framework:

This section illustrates how the proposed framework functions under the previous assumption. In the beginning, all the required inputs are accepted from a test leader. Next, code coverage and cyclomatic complexity are calculated by software tool, and they are stored in the test repository. After that, all the required information is forwarded to the organizer. Next, the organizer begins in monitoring and organizing the process of the PR engine through the following steps. Figure 2 summarizes the steps of the proposed approach for the PR engine:

Step1: Apply the clustering approach:

Once all the required inputs are available to the organizer. Then, it calls the TS-clustering to apply the K-means clustering in order to group test cases into clusters. The necessary inputs to TS-

clustering are test cases with code coverage and CC attributes and the initial “K”. Based on these inputs, the K-means start to perform the clustering approach. K-means clustering can be clustered test cases according to only code coverage or both cyclomatic complexity and code coverage.

Code coverage-based clustering is applied when the tester selects either the code coverage or cyclomatic

complexity criteria for prioritization while code coverage and cyclomatic complexity-based clustering are used when the tester selects fault history information for prioritization. The output of this step is “k” number of clusters that contains redundant test cases.

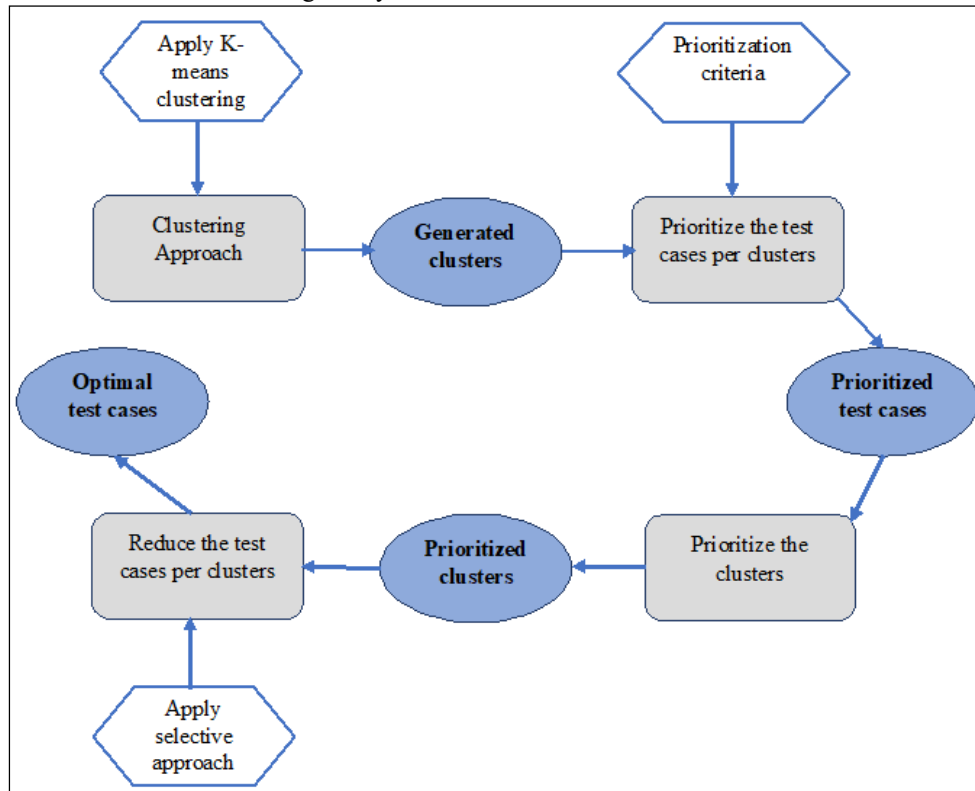


Figure 2. The Proposed Approach For The PR Engine.

Step 2: Prioritize the test cases per clusters:

Once clusters of test cases are created. Then, the organizer calls the PR engine to start its function. It begins with the prioritization process. Prioritization process takes created clusters and prioritization criteria as inputs in order to arrange test cases per clusters. In the proposed framework, the prioritization process is implemented by three methods:

1. Code coverage-based test suite prioritization (cov-TSP): test cases can be ordered in descending order according to the number of code statements, branches, paths, or methods. These code coverage information is computed at the beginning for each test case by a software coverage tool.

2. Cyclomatic complexity-based test case prioritization (cyc-TSP): test cases can be ordered in descending order according to the code

complexity metrics. This metric is calculated for each test case by a software coverage tool.

3. Fault history information-based test case prioritization (fault-TSP): test cases can be ordered in descending order according to the number of faults that are detected by each test case in the previous executions. The numbers of faults that are detected by each test case are retrieved from test repository, and then the ratio of fault detection for each test case is calculated by dividing the number of faults that detected with each test case by the total number of faults. The values of fault detection ratio range between zero and one and they are used to order test cases.

The prioritization process is done by following only one of the above criteria. The output of the prioritization process is the same number of input clusters but with prioritized test cases. The

prioritized test cases per clusters are handed over to the buffer.

Step 3: Prioritize the clusters:

Before the reduction process start, the selective approach should follow a way to visit the clusters. The tradition way that is frequently used is to visit clusters in the same order that is generated by the clustering tool. To optimize the effectiveness of the selected test cases, it is useful to visit clusters according to the higher priority of

In this step, the organizer calls the reduction process to remove the redundant test cases. The test cases within the same cluster are assumed to have the same behavior. Whereas the test cases that have the same distance values within this cluster are supposed to be repetitive and must be removed from this cluster. The reduction process is implemented by applying an effective selection approach. The procedure of the selective approach is as follow: first, the selection approach visits clusters by their new order. It selects the first test case from the first order cluster and put it in an empty list. Then, it removes this selected test case from the original list of test cases. Next, the second-order test case is retrieved, and its distance value compares with the distance value of previously added test cases in the list. If the distance value of the retrieved test case is equal to the distance value of any test cases within the list. Then, this test case is not added into the list in order to avoid the redundant test cases. Else, it is added into the list. The same process is repeated until the last test case in that cluster is retrieved. Second, the procedure is moved to the second-order cluster and begins to retrieve the test cases with the same process. Third, it moves to the next order cluster and so on. This procedure continues until all clusters are visited.

The output of this step is the optimal set of test cases that contains a high priority and unique test cases. This optimal set of test cases can be effectively executed in SUT and accordingly improving the software testing process.

4. CASE STUDY:

To verify the proposed framework, an empirical study is conducted through the following phases:

Phase 1: Test case generation and data collection

A java source code for Fee Report (Student Management System) [41] is downloaded. The fee management system is used to maintain the records of students for a long time and make a straightforward calculation of students' fee where the accountant can be added, viewed, or deleted by admin. And then the accountant can add, remove,

clusters. Therefore, this step is necessary to be performed to prioritize clusters. This can be done by ordering the clusters based on the maximum value of prioritization criteria in each cluster. The priority of clusters gives the PR engine a feature that increases the efficiency of the proposed framework. The output of this step is a new order for the clusters.

Step 4: Reduce the test cases per clusters:

edit, or view student to check paid and a due fee of the student. The source code of the Fee System consists of 1,696 Lines of code, 15 classes, and different methods per each class.

Eclipse is used to run the Fee System [42]. Eclipse is a Java integrated development environment (IDE) that is widely used for developing Java applications. It contains many features and plug-ins that present all the functionality for any application written in Java programming language. The following steps are performed by using Eclipse:

1. Generate a set of test cases for the source code of the Fee System using CodeProAnalytix tool:

CodeProAnalytix is a free automated software testing tool that was developed as a plug-in for Eclipse. It has many features, such as code metrics and code coverage calculation. The main feature of this tool is the Junit test case generation, where different test cases can be generated automatically for each input class. By this tool, 142 automated test cases are generated for some classes of Fee System, and 9 test cases are generated manually for other classes. Table 1 represents the number of automatic and manual test cases that are generated for each test suite with 151 test cases as a total.

Table 1. Number Of Generated Test Cases For Different Test Suite.

Test Suite_Name	Number of generated test cases
Test Suite_Accountant	12
Test Suite_Accountant_Dao	26
Test Suite_Student	27
Test Suite_Student_Dao	77
Test Suite_Accountant_login	4
Test Suite_Add_Accountant	2
Test Suite_Add_Student	3
Total	151

2. Calculate the code coverage data for the generated test cases using EclEmma tool:

EclEmma is a free testing tool for Java code coverage in Eclipse, available under the Eclipse marketplace. It automatically calculates code coverage for test cases directly into Eclipse. It supports four types of code coverage that are listed in a view menu: Instructions counters, Branch counters, Line counters, and Method counters.

3. Calculate cyclomatic complexity for the generated test cases using JaCoCo metrics:

JaCoCo metrics is a plug-in in EclEmma for code coverage and the cyclomatic complexity calculation. It is used to calculate the cyclomatic complexity for test cases.

The generated code coverage data (Instructions, Branch, Line, and Method) and cyclomatic complexity for 151 test cases that are generated by Junit tool are collected and stored in Excel file with CSV extension. Table 2 shows a sample of test cases with six attributes: TC_ID, Instruction, Branches, Lines, Methods, and C_Complexity.

Phase 2: case study setup

To understand the function of the proposed framework, three case scenarios are presented to help in understanding the work of the basic components of the proposed framework based on the prioritization criteria that is selected by the tester. The proposed framework allows the tester to choose from three prioritization criteria: code coverage, cyclomatic complexity, or the number of detected faults.

1. Case Scenario A: code coverage as a prioritization criteria

Suppose the tester selects code coverage data as prioritization criteria. Based on the coverage data that is collected for the generated test cases, four types of code coverage data are calculated (Instructions, Branch, Line, and Method). The tester should select one from these four coverage data as prioritization criteria. Then, the organizer calls the TS clustering to perform multi-code coverage based clustering using other types of code coverage data and cyclomatic complexity. For example, if the tester selects Instruction code coverage as prioritization criteria, then code coverage based clustering is performed based on (Branch, Line, and Method) and cyclomatic complexity. Therefore, there are four case studies in this case scenario based on the selected type of code coverage data.

Table2. Sample Set For Fee System's Test Cases Coverage Data.

TC_ID	Instruction	Branches	Lines	Methods	C_Complexity
1	203	1	76	12	3
2	200	1	85	12	3
3	210	0	67	12	3
4	217	0	96	12	3
5	270	2	96	12	3
6	270	0	96	12	3
7	270	0	96	10	3
8	270	0	96	10	3
9	270	0	100	10	1
10	270	0	100	10	1
11	270	0	100	3	1
12	190	0	100	3	1
13	190	0	85	4	1
14	190	0	85	3	1
15	270	1	85	2	1
16	270	1	100	10	5
17	270	1	100	2	3
18	203	0	75	2	5
19	203	0	75	0	5
20	0	0	75	0	5
21	0	0	75	0	5
22	0	0	75	0	9
23	190	1	85	3	11

One case study is selected where the tester selects Instruction code coverage as prioritization criteria, and the proposed approach for the PR process is implemented through the following steps:

1. Apply the K-means algorithm:

SPSS [43] is used to perform a code coverage based clustering using K-means algorithm. SPSS is statistical analysis software with a fast and powerful solution for statistical problems. It is used by many researchers such as market, education, health and data miner researchers. It supports clustering analysis using K-means algorithm and hierarchal algorithm. SPSS clustering analysis is used because it uses a simple interface for K-means clustering and can calculate the distance of each input data from the cluster centre and shows the calculated distance in the output of the clustering process. Distance data is essential for a later reduction process.

In K-means cluster analysis window, only Branch, Line, Method and C_Complexity variables are selected for clustering, and the number of clusters is decided to be four clusters (K=4) as illustrated in figure 3. After applying the K-means algorithm, the result of clustering analysis is exposed as in figure 4. Figure 4 represents a list of test cases with new variables: cluster number and distance. The distance variable contains the distance value of each test case from its cluster centre. The test cases in each cluster are stored in tabular form within the SQL database in four tables: cluster_1, cluster_2, cluster_3, cluster_4.

Each one has three columns: TC_ID, prioritization criteria (Instruction), and Distance

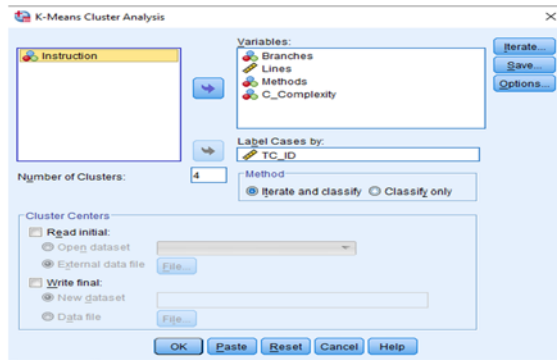


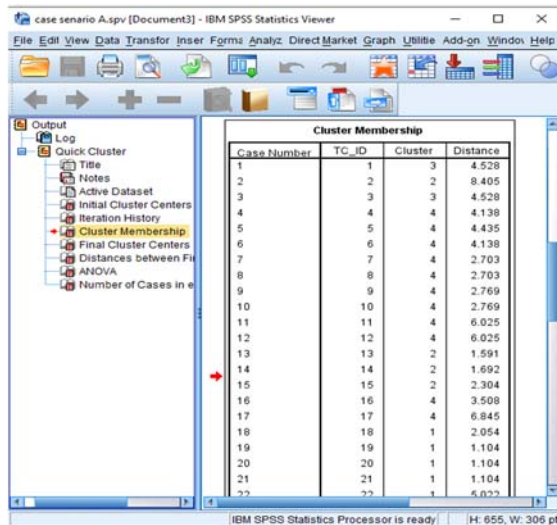
Figure 3. K-Means Clustering Analysis For Scenario A.

2. Prioritize the test cases in each cluster:

The records inside each cluster table are ordered in descending order based on values of prioritization criteria (Instruction attribute). So, the test cases in each cluster are prioritized based on Instruction code coverage data, and a new order of test cases is created.

3. Prioritize the clusters:

The applied K-means clustering creates the four clusters in this order: cluster 1, cluster 2, cluster 3, and then cluster 4. But, to select a high priority of test cases. The clusters must be ordered based on the maximum value of Instruction attribute in each cluster. This can be done by calculating the maximum value of Instruction attribute in each cluster table and then order these values in descending order. A new order of the clusters is created as C1, C2, C3, and C4.



Case Number	TC_ID	Cluster	Distance
1	1	3	4.528
2	2	2	8.405
3	3	3	4.528
4	4	4	4.138
5	5	4	4.435
6	6	4	4.138
7	7	4	2.703
8	8	4	2.703
9	9	4	2.769
10	10	4	2.769
11	11	4	6.025
12	12	4	6.025
13	13	2	1.591
14	14	2	1.692
15	15	2	2.304
16	16	4	3.508
17	17	4	6.845
18	18	1	2.054
19	19	1	1.104
20	20	1	1.104
21	21	1	1.104
22	22	1	6.022

Figure 4. Sample Of K-Means Clustering Output For Scenario A.

4. Reduce the test cases per clusters:

In this step, the redundant test cases are removed. To generate the optimal set of high priority and unique test cases, the selective approach that is mentioned in section 3.2 is used to retrieve the ordered test cases from each cluster by visiting the clusters in order that results in the previous step (step 3). The procedure of the selective approach can be illustrated by an example. See figure 5 and suppose ten test cases are clustered into two clusters.

2. Case Scenario B: Cyclomatic complexity as a prioritization criteria

In this case scenario, the tester is supposed to select cyclomatic complexity as prioritization criteria. Consequently, a multi-code coverage based clustering is processed using only the four types of code coverage data (Instructions, Branch, Line, and Method). The previous four steps are implemented respectively for this scenario. At the end of this scenario, an optimal list of high priority and unique test cases is created based on cyclomatic complexity prioritization criteria. The optimal list of this scenario contains only 63 test cases compared to the optimal list of scenario A that includes 70 test cases.

3. Case Scenario C: Faults history as prioritization criteria.

In this case scenario, the tester is supposed to select faults history as prioritization criteria. Consequently, a multi-code coverage based clustering is processed using the four types of code coverage data (Instructions, Branch, Line, and Method) and cyclomatic complexity. To implement the steps of the proposed approach for this scenario, a history of the number of real faults is required. Faults that are supposed to have been detected by each test case during the first execution of the Fee System. But, there are no faults history data for the Fee System. Therefore, the number of faults is seeded manually into the source code of the Fee System. And then execute each test case against these faults to determine which faults can be detected by each test case. 15 faults are seeded in a different position in the source code.

The numbers of faults that are detected by each test case are represented in table 3. Based on these number of faults, the average value of fault detection for each test cases is calculated by dividing the number of faults that are detected by each test case (No.faults) by the total number of faults (15 faults). The average values of fault

detection represent the rate of fault detection. Then, these values are used as prioritization criteria for this scenario. Respectively, the four steps of the proposed PR process are implemented for this scenario. Accordingly, an optimal list of high priority and unique test cases is generated based on the rate of fault detection as prioritization criteria. The optimal list contains only 68 test cases. Figure 6 shows the number of test cases before and after the PR process for each case scenario.

Table 3. Sample Data For The Number Of Detected Faults.

TC ID	No.faults	Average no.faults
1	2	0.13
2	1	0.07
3	0	0.00
4	0	0.00
5	1	0.07
6	3	0.20
7	1	0.07
8	2	0.13
9	2	0.13
10	0	0.00

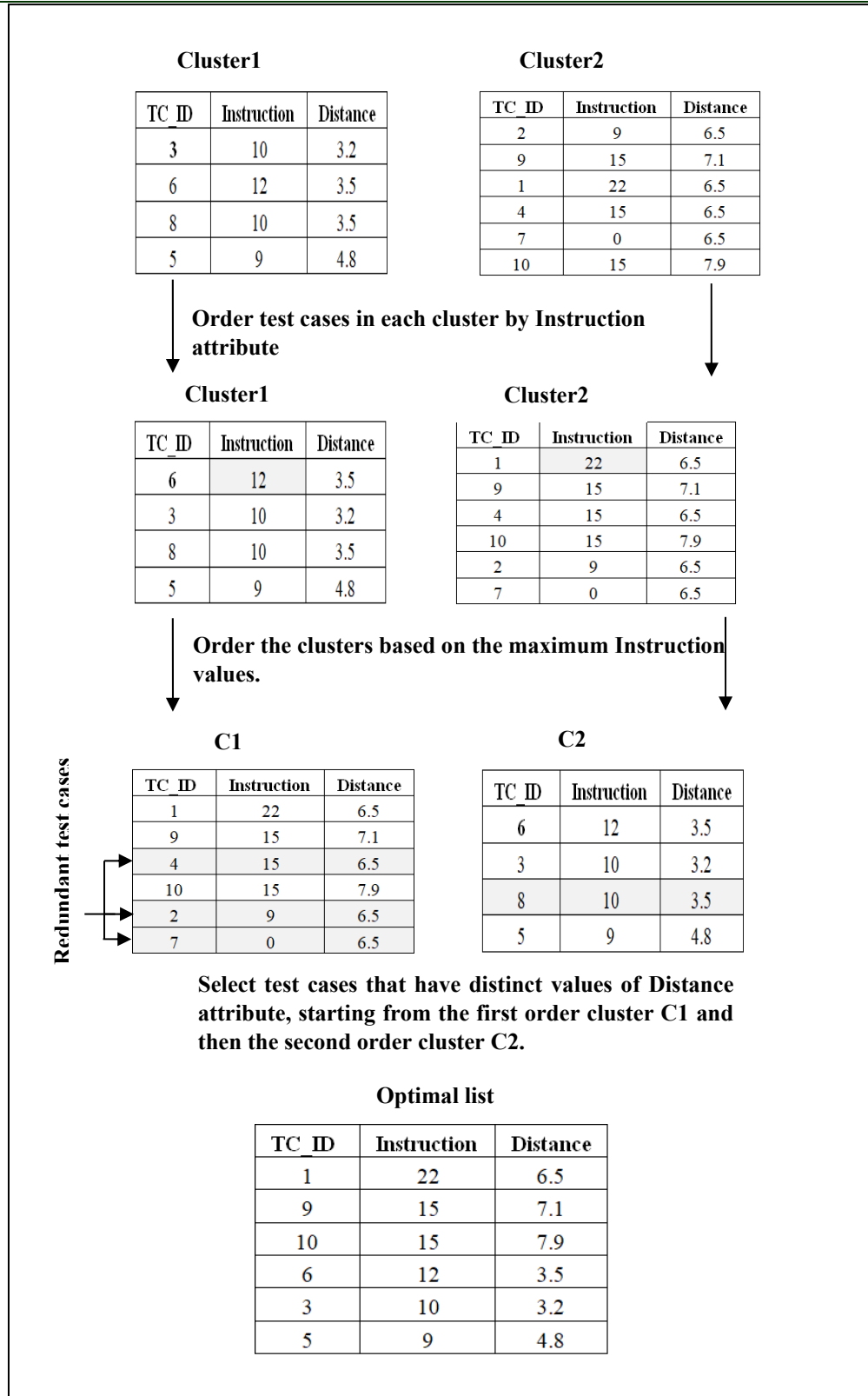


Figure 5. Example Of The Selective Procedure For Reduction Process.

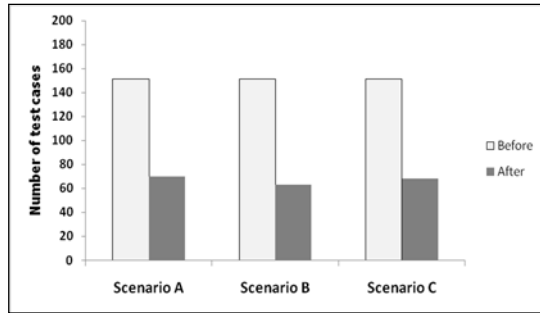


Figure 6. Number Of Test Cases For Scenario A, B, And C.

Phase three: measurement metrics

The measurement metrics that are used to evaluate the proposed framework are code coverage and APFD. The code coverage is recomputed using Eclemma tool for the optimal set of test cases that are generated in each case scenario. Table 4, Table 5, and table 6 represent the average coverage before and after the PR process for each case scenario. And figure 7 shows the average of code coverage for the three scenarios before and after the PR process.

Table 4. The Code Coverage Before And After The PR Process For scenario A.

Code coverage	Before	After
Instructions	85.00%	89.00%
Branches	70.20%	70.00%
Lines	80.00%	78.30%
Methods	93.00%	90.6%
Average	82.05%	81.97%

Table 5. The Code Coverage Before And After The PR Process For Scenario B.

Code coverage	Before	After
Instructions	85.00%	80.00%
Branches	70.20%	69.00%
Lines	80.00%	78.60%
Methods	93.00%	91.20%
Average	82.05%	79.70%

The second measurement is the APFD metric that is mentioned in section 2.2. The APFD is calculated by the following Equation [44].

$$APFD = 1 - \frac{TF_1 + TF_2 + \dots + TF_m}{nm} + \frac{1}{2n}$$

Where, $T \implies$ a set a test suite, $F \implies$ a set of faults, $n \implies$ number of test cases in T, $m \implies$ number of faults in F, and $TF_i \implies$ the position of the first test in T that catches fault i. Figure 8, figure 9, and figure 10 show the APFD for each scenario.

Table 6. The Code Coverage Before And After The PR Process For Scenario C.

Code coverage	Before	After
Instructions	85.00%	84.00%
Branches	70.20%	70.20%
Lines	80.00%	78.00%
Methods	93.00%	93.00%
Average	82.05%	81.30%

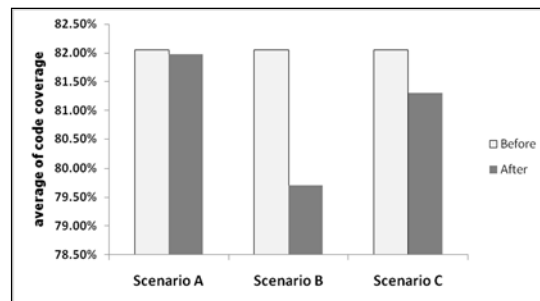


Figure 7. The Average Of Code Coverage For Scenario A, B, And C.

5. RESULTS

The results show that:

- According to figure 6 the size of test suites is reduced in each case scenario after implementing the proposed approach for the PR process by 54%, 58%, and 55% for scenario A, scenario B, and scenario C respectively. Subsequently, the execution time of the testing process for the Fee System will be reduced, and results in lower costs. The findings indicate that the proposed framework can effectively reduce the size of test suites by more than 50%. This contributed to confirm the importance of using the TSR techniques in the proposed PR model.
- The average of the code coverage for each scenario after implementing the PR process is 81.97% for scenario A, 79.70% for scenario B, and 81.30% for scenario C. as illustrated, the average of the code coverage still achieving a

good result compared to the average of the code coverage before PR process that is 82.05%.

- From figure 7, the results indicate that the percentage of code coverage reduction for each tester. For scenario A where the instruction coverage data is used as prioritization criteria, the percentage of code coverage reduction is 0.1% that is best over the two other scenarios (3% and 0.9% for scenario B and scenario C, respectively). However, the percentage of code coverage reduction for scenario A and scenario C are very close to each other (0.1% and 0.9%). The findings indicate that there may be a significant relationship between the percentage of code coverage reduction and the selected prioritization criteria. This contributed to determine which the best prioritization criteria to be selected.

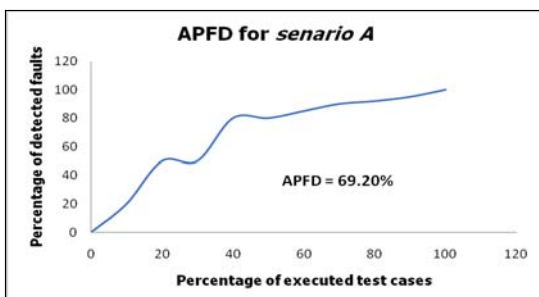


Figure 8. APFD For The Optimal Set Of Scenario A.

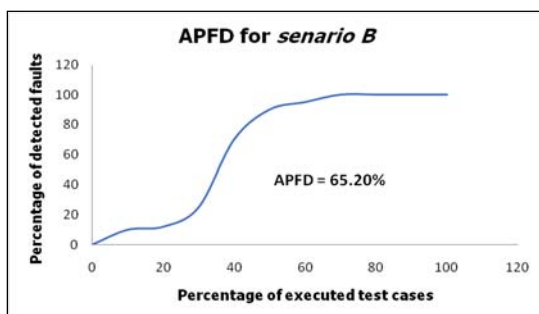


Figure 9. APFD For The Optimal Set Of Scenario B.

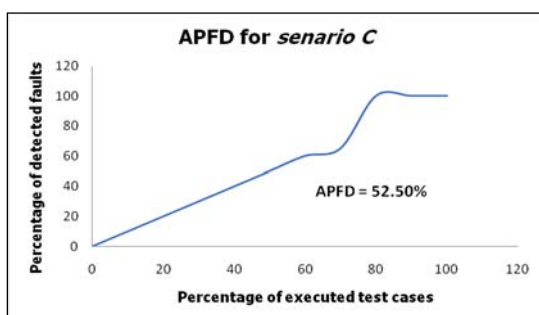


Figure 10. APFD For The Optimal Set Of Scenario C.

scenario is different according to the prioritization criteria that are selected by the

- The results of APFD for each scenario indicate that scenario A has a higher value of 69.20% compare to scenario B and scenario C (65.20% and 52.50%, respectively). Although, the scenario C has the lowest value of fault detection rate. But, this result may be differed for real faults history.

6. DISCUSSION:

Based on the hand results, this section discusses the fundamental trends that are observed from results analysis as well as the related points concerning software testing area of research. The following issues are concluded:

- The proposed framework can effectively prioritize and reduce the test cases and provide the tester with the optimal set of test cases that will be executed in less time without more losing in code coverage rate. Subsequently, the proposed framework is useful for a tester when he hinders with the time and cost of the testing process.
- Investigate both test suite prioritization and reduction techniques in the proposed framework improve its effectiveness by combining them in a way (the PR process) that is efficient for different testing situations.
- Implementing multi-code coverage based clustering for prioritization and reduction techniques optimize the effectiveness of both techniques. As illustrated from the results: in scenario A where the tester selects instruction code coverage as a prioritization criterion, the clustering approach is based on multi-code coverage (Branch, Line, and Method) and cyclomatic complexity as another attribute of test cases. In this scenario, both prioritization and reduction technique is based on code coverage data. Consequently, scenario A has the highest value of code coverage. As well as in scenario C, where the tester selects the average of fault detection as a prioritization criterion, also the clustering approach is based on multi-code coverage and cyclomatic complexity as another attribute of test cases. Therefore, the average value of code coverage for scenario C is very close to the average value of code coverage for scenario A. but for scenario B, the average value of code coverage is less than other two scenarios because the clustering approach in this scenario is based only on code coverage data without any additional attributes of test cases.

- The value of APFD for each scenario is not high enough, but they still yielded good results for the fault detection rate.
- Based on the results, the tester can take maximal advantage of the proposed framework if he follows some guides. The first choice for prioritization criteria should be a code coverage data; this can be developed in the proposed framework by making the code coverage as the default choice for prioritization criteria. The second choice is fault history, and cyclomatic complexity is the last choice. The tester may follow these guidelines when all these data are available in the test repository. Otherwise, the tester makes the appropriate decision based on the available data.

The main limitations of this study should be highlighted as follow:

- The study is conducted with only one SUT that is a java source code application, and not be generalized for a different sizes and types of application. Therefore, future work should consider other applications for more generalization.
- The prioritization criteria are confined to only three types of information: code coverage, cyclomatic complexity, and faults history.
- The manual method that is used for fault seeding, the limited number of seeded faults, and the distribution of faults in source code are the major limitation of this study.
- The APFD measurement metric is not accurate to measure the rate of fault detection when fault severity and costs of test cases are considered. Therefore, this study assumed that all faults have the same severity, and all test cases have the same cost. But, when fault severities and costs are considered, this study should provide additional measurement metrics for the rate of fault detection.

7. CONCLUSION AND FUTURE WORK

The proposed framework is practical for improving the software testing process in terms of time and cost. It uses a clustering-based test suite prioritization and reduction techniques. It aims to automatically generate an optimal set of test cases by combining test suite prioritization and reduction techniques based on clustering data mining technique. The optimal set of test cases can be effectively executed under a limited time constraint.

Moreover, the proposed framework can order and reduce test suites by using multiple code coverage criteria and different prioritization criteria. Therefore, the proposed framework can improve the software testing process when the time is limited, and fault detection is a critical issue for a test manager.

Furthermore, the proposed framework is evaluated by conducting an empirical study. The results indicate that the proposed framework can effectively prioritize and reduce the size of test suites with a high percentage of code coverage for each scenario (81.97% for scenario A, 79.70% for scenario B, and 81.30% for scenario C) and good value of APFD for each scenario (69.20% for scenario A, 65.20% for scenario B, and 52.50%, for scenario C). Therefore, the proposed framework is robust and valuable for software testing process.

For future work, it is essential to evaluate the proposed framework for different types and sizes of SUT. It is required to use other criteria for prioritization and conduct additional case studies for further estimation of the proposed framework. It is needed to make a questionnaire to take the opinions of the testers for the proposed framework.

REFERENCES

- [1] Sharma N, Purohit GN. Test case prioritization techniques “an empirical study”. In 2014 International Conference on High Performance Computing and Applications (ICHPCA) 2014 Dec 22 (pp. 1-6). IEEE.
- [2] Coutinho AE, Cartaxo EG, Machado PD. Test suite reduction based on similarity of test cases. In 7th Brazilian workshop on systematic and automated software testing—CBSOFT 2013.
- [3] Bertolino A, Cartaxo E, Machado P, Marchetti E, Ouriques JF. Test suite reduction in good order: comparing heuristics from a new viewpoint. On Testing Software and Systems: Short Papers. 2010 Oct:13.
- [4] Sampath S, Bryce RC. Improving the effectiveness of test suite reduction for user-session-based testing of web applications. Information and Software Technology. 2012 Jul 1;54(7):724-38.
- [5] Sampath S, Bryce RC, Jain S, Manchester S. A tool for combination-based prioritization and reduction of user-session-based test suites. In 2011 27th IEEE International Conference on Software Maintenance (ICSM) 2011 Sep 25 (pp. 574-577). IEEE.

- [6] Khan SU, Lee SP, Parizi RM, Elahi M. A code coverage-based test suite reduction and prioritization framework. In 2014 4th World Congress on Information and Communication Technologies (WICT 2014) 2014 Dec 8 (pp. 229-234). IEEE.
- [7] Harrold MJ, Gupta R, Soffa ML. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology (TOSEM)*. 1993 Jul 1;2(3):270-85.
- [8] Alian M, Suleiman D, Shaout A. Test case reduction techniques-survey. *International Journal of Advanced Computer Science & Applications*. 2016 May 1;1(7):264-75.
- [9] Ilkhani A, Abaee G. Extraction test cases by using data mining; reducing the cost of testing. In 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM) 2010 Oct 8 (pp. 620-625). IEEE.
- [10] Singh R, Santosh M. Test case minimization techniques: a review. *International Journal of Engineering Research & Technology (IJERT)*. 2013 Dec;2(12).
- [11] Shrivathsan AD, Ravichandran KS, Sekar KR. Test suite reduction mechanisms: a survey. *International Journal of Applied Engineering Research*. 2015;10(18):39841-8.
- [12] Saifan AA, Alsukhni E, Alawneh H, Sbah AA. Test case reduction using data mining technique. *International Journal of Software Innovation (IJSI)*. 2016 Oct 1;4(4):56-70.
- [13] Raamesh L, Uma GV. Reliable mining of automatically generated test cases from software requirements specification (SRS). arXiv preprint arXiv:1002.1199. 2010 Feb 5.
- [14] Kansomkeat S, Thiket P, Offutt J. Generating test cases from UML activity diagrams using the Condition-Classification Tree Method. In 2010 2nd International Conference on Software Technology and Engineering 2010 Oct 3 (Vol. 1, pp. V1-62). IEEE.
- [15] Dubey Y, Singh D, Singh A. Amalgamation of Automated Test Case Generation Techniques with Data Mining Techniques: A Survey. *International Journal of Computer Applications*. 2016 Jan;975:8887.
- [16] Khan FA, Gupta AK, Bora DJ. An efficient approach to test suite minimization for 100% decision coverage criteria using K-Means clustering approach. *IJAPRR International Peer Reviewed Refereed Journal*. 2015;2(VII):18-26.
- [17] Mottaghi N, Keyvanpour MR. Test suite reduction using data mining techniques: A review article. In 2017 International Symposium on Computer Science and Software Engineering Conference (CSSE) 2017 Oct 25 (pp. 61-66). IEEE.
- [18] Shahid M, Ibrahim S, Mahrin MN. A study on test coverage in software testing. *Advanced Informatics School (AIS), Universiti Teknologi Malaysia, International Campus, Jalan Semarak, Kuala Lumpur, Malaysia*. 2011.
- [19] Chantrapornchai C, Kinputtan K, Santibowanwing A. Test Case Reduction Case Study for White Box Testing and Black Box Testing using Data Mining. *International Journal of Software Engineering and Its Applications*. 2014;8(6):319-38.
- [20] Muthyala K, Naidu RA. A novel approach to test suite reduction using data mining. *Indian Journal of Computer Science and Engineering*. 2011 Jul;2(3):500-5.
- [21] Dash R, Dash R, Siksha IT. Application of K-mean algorithm in software maintenance. *International Journal of Emerging Technology and Advanced Engineering*. 2012 May;2(5).
- [22] Subashini B, JeyaMala D. Reduction of test cases using clustering technique. In International Conference on Innovations in Engineering and Technology (ICIET'14) 2014.
- [23] Jeffrey D, Gupta N. Improving fault detection capability by selectively retaining test cases during test suite reduction. *IEEE Transactions on software Engineering*. 2007 Jan 8;33(2):108-23.
- [24] Prasad S, Jain M, Singh S, Patvardhan C. Regression Optimizer A Multi Coverage Criteria Test Suite Minimization. *International Journal of Applied Information Systems (IJ AIS), Foundation of Computer Science FCS, New York, USA*. 2012 Apr;1(8).
- [25] Kumar A, Singh K. A Literature Survey on test case prioritization. *An international journal of advanced computer technology. CompuSoft*. 2014 May 1;3(5):793.
- [26] ur Rehman I, Malik SU. The impact of test case reduction and prioritization on software testing effectiveness. In 2009 International Conference on Emerging Technologies 2009 Oct 19 (pp. 416-421). IEEE.
- [27] Zhou ZQ, Sinaga A, Susilo W. On the fault-detection capabilities of adaptive random test case prioritization: Case studies with large test suites. In 2012 45th Hawaii International Conference on System Sciences 2012 Jan 4 (pp. 5584-5593). IEEE.

- [28] Jiang B, Zhang Z, Chan WK, Tse TH. Adaptive random test case prioritization. In Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering 2009 Nov 16 (pp. 233-244). IEEE Computer Society.
- [29] Carlson R, Do H, Denton A. A clustering approach to improving test case prioritization: An industrial case study. In 2011 27th IEEE International Conference on Software Maintenance (ICSM) 2011 Sep 25 (pp. 382-391). IEEE.
- [30] Shahid M, Ibrahim S, Mahrin MN. A study on test coverage in software testing. Advanced Informatics School (AIS), Universiti Teknologi Malaysia, International Campus, Jalan Semarak, Kuala Lumpur, Malaysia. 2011.
- [31] Do H, Mirarab S, Tahvildari L, Rothermel G. The effects of time constraints on test case prioritization: A series of controlled experiments. IEEE Transactions on Software Engineering. 2010 Jun 7;36(5):593-617.
- [32] Rothermel G, Untch RH, Chu C, Harrold MJ. Test case prioritization: An empirical study. In Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No. 99CB36360) 1999 Aug 30 (pp. 179-188). IEEE.
- [33] Elbaum S, Malishevsky A, Rothermel G. Incorporating varying test costs and fault severities into test case prioritization. In Proceedings of the 23rd International Conference on Software Engineering 2001 Jul 1 (pp. 329-338). IEEE Computer Society.
- [34] Catal C, Mishra D. Test case prioritization: a systematic mapping study. Software Quality Journal. 2013 Sep 1;21(3):445-78.
- [35] Medhun Hashini DR, Varun B. Clustering approach to test case prioritization using code coverage metric. International journal of engineering and computer science. 2014.
- [36] Arafeen MJ, Do H. Test case prioritization using requirements-based clustering. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation 2013 Mar 18 (pp. 312-321). IEEE.
- [37] Pathania Y, Kaur G. Role of Test Case Prioritization based on Regression Testing using Clustering. International Journal of Computer Applications. 2015 Jan 1;116(19).
- [38] Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. IEEE Transactions on software engineering. 2001 Oct;27(10):929-48.
- [39] Upadhyay AK, Misra AK. Prioritizing test suites using clustering approach in software testing. International Journal of Soft Computing and Engineering (IJSCE). 2012 Sep;2(4).
- [40] Mukherjee R, Patnaik KS. A survey on different approaches for software test case prioritization. Journal of King Saud University-Computer and Information Sciences. 2018 Oct 3.
- [41] <https://www.javatpoint.com/free-java-projects>.
- [42] <https://marketplace.eclipse.org/content/ibm>.
- [43] <https://www.ibm.com/analytics/spss-statistics-software>.
- [44] Pradeepa R, VimalDevi K. Effectiveness of test case prioritization using APFD metric: Survey. In International Conference on Research Trends in Computer Technologies (ICRTCT—2013). Proceedings published in International Journal of Computer Applications@ (IJCA) 2013 (pp. 0975-8887).