International Journal of Database Theory and Application
Vol.9, No.4 (2016), pp.23-44
http://dx.doi.org/10.14257/ijdta.2016.9.4.02

A Hybrid Approach of Clustering and Time-Aware Based Novel Test Case Prioritization Technique

Geetanjali Chaurasia and Sonali Agarwal

Department of Information Technology, Indian Institute of Information Technology, Allahabad-211012, India
geetanjalichaurasia@gmail.com
sonali@iiita.ac.in

Abstract

Regression testing is an activity during the maintenance phase to validate the changes made to the software and to ensure that these changes would not affect the previously verified code or functionality. Often, regression testing is performed with limited computing resources and time budget. So, fully comprehensive testing is not possible at this stage. Test-case prioritization techniques are applied to ensure the execution of test cases in some prioritized order and to achieve some specific goals in minimum possible time like, increasing the rate of fault detection, detecting the most critical faults as early as possible etc. The main objective of this paper is to achieve higher value of average percentage of faults detected, execute the higher priority test cases before lower priority test cases and also we target to decrease the execution time for achieving the maximum value of average percentage of faults detected. We proposed a new prioritization technique that uses a clustering approach and also considers various factors like, execution time of every test case, code coverage metric, fault detection ratio, test case failure rate and code complexity metric to reorder the execution of test cases. The results of this research work will show the importance of clustering technique and various factors taken into consideration, for achieving effective prioritization of test cases. The results of implementation will subsequently show that the proposed approach is more effective than the existing coverage and clustering based prioritization techniques. From the experimental results, we found that our proposed approach achieved higher value of average percentage of faults detected than other clustering based and coverage based techniques. Also, this approach reduces the execution time taken by the prioritized test cases.

Keywords: *Clustering; Regression testing; test case prioritization; test suite*

1. Introduction

At the time of the formal testing phase software testers develop the test suites. These test suites often keep save to reuse them from future's perspective. "Regression testing" is an activity during the maintenance phase to validate the changes made to the software and to ensure that these changes would not affect the previously verified code or functionality. Often, regression testing is performed with limited computing resources and time budget. Executing the whole test suite at the time of regression testing is infeasible because a test suite contains large number of test cases and rerunning all the test cases would take enormous amount of time and resources. Rerunning such test suites can cost one half of the total cost requires spending in maintenance phase and take unjustifiable excessive amount of time [1-3]. For example, an industry reported that executing all the test cases at the maintenance phase required seven weeks for a product having 20000 lines of code [4]. There are also other challenges that one has to face at this phase like, Software development team often thinks that any change applied to a part of code will affect only that part of code and thus regression testing focusing only that area is sufficient and only corresponding test cases are sufficient to be executed. But, actually changes made in one part can draw impact on any other area and the whole code is needed to be tested again and again. This is a very difficult issue while performing regression testing. While making regression testing plans, testing team often starts execution with simplest tests and moves towards the complex and critical tests' execution covering complex functionalities at last. This flow of executing test cases totally divert the goal of regression testing to find most critical defects and bugs at the earliest. To get meaningful results from this phase is a very slow process. But sometimes project managers do not have that much time to wait for the results. They want to know whether current version is working correctly or not as early as possible. Limitation of resource availability is another main challenge of regression testing. Testing team cannot spend much cost, time and effort at the regression testing phase as it can spend on formal testing phase of SDLC. But in reality, regression testing requires maximum resources allocated to the project. One of the main challenges of regression testing is the repetition of regression testing cycle again and again. Whenever a change is applied to the software program, regression testing test cases has to get executed to fix the bugs. It is very difficult to manage all the test cases in large test suites for each run of the regression testing cycle.

In these situations, testers may re-arrange the test cases to assign the priority to each test case so as to execute them in a specific order aiming to achieve desired performance goals. The performance goals include increasing the rate of fault detection, detecting the severe faults as early as possible, reliability *etc.* Test case prioritization is different from test case selection and minimization in the sense that it overcomes the drawbacks of these two by not discarding the test cases because sometimes discarding of the test cases is not acceptable [5].

The first test case prioritization technique was proposed by Rothermel *et al*. in 1999 that only considered code coverage metric information for the prioritization [5]. After that many test case prioritization techniques have been presented that incorporated various factors like time, cost, faults, risks, history of executing the test cases *etc.* There are various techniques of test case prioritization are available in literature like, Coverage based, Distribution based, Human based, History based, Risk based, Fault-aware based, Requirements based *etc.* Although various clustering based techniques have also been proposed by researchers, but still improvements are required to make it more effective in terms of maximum rate of fault detection in minimum execution time and other factors.

In this paper, we present a hybrid approach of clustering and time-aware based novel test case prioritization technique that first clusters the test cases on the basis of their common property using Fuzzy C-means and then performs intra and inter- cluster prioritization based on coverage, complexity metric, execution time and test execution history. This work is an attempt to propose a better technique that can overcome the shortcomings of the existing ones and includes execution time factor while prioritizing the test cases. The comparative analysis of the results obtained by the proposed technique and the existing clustering based and coverage based prioritization techniques will show the effectiveness of the proposed one.

The rest of the paper is divided into seven sections. Section II presents the background of the task under consideration and describes general definition of test case prioritization. Section III presents related research works that have been done for test case prioritization problem. Section IV describes our proposed approach in detail. Section V describes the experimental studies performed for carrying out this research work including object of analysis, data collection, variables and measures *etc.* Section VI discusses and analyzes the experimental results obtained from the proposed approach. Section VII discusses various threats to validity related to this research work. Section VIII includes conclusion and future work.

2. Background

2.1 Methodologies Use for Regression Testing

Four methodologies have been proposed to handle the regression testing challenges:

2.1.1 Retest All

In this method, the test cases that are not applicable to the current modified version of the software are rejected and the other left over test cases are applied to test the modified version.

2.1.2 Regression Test Selection

While following the previously discussed method testing team has to apply all the test cases to test the modified version of the software, which takes a lot of effort and time and hence it is very expensive. But, in test selection approach subset of test cases are selected on the basis of the requirements of the modified version. So this method uses the information of the modifications applied in the current version to select the test cases [6].

2.1.3 Test Suite Reduction

This method reduces the size of the test suite by discarding the test cases that are not meaningful for the current version of the program by using the information of the modifications applied in the current version [6]. It also removes the redundant test cases. It is different from the previous one in the sense that test selection only chooses the test cases and does not discards the left over test cases. The main benefit of this method is the ease of handling the large test suites by reducing its size. But sometimes the fault detection capability of the reduced test suite is questioned.

2.1.4 Test Case Prioritization

In this method test cases are executed in prioritized order. Each test case gets a priority on the basis of some criteria and then they are applied in prioritized fashion from highest to lowest priority. Test case prioritization is different from test case selection and minimization in the sense that it overcomes the drawbacks of these two by not discarding the test cases because sometimes discarding of the test cases is not acceptable. But, prioritization can also be used in conjunction with selection and minimization. Before prioritization, test case minimization is done, *i.e.*, to select a minimum number of test cases that can fulfil the intension of performing regression testing. Then, we can prioritize them with goals like, to detect faults as soon as possible, to detect the most destructive faults first, and to attain certain coverage criterion *etc.*

This research work mainly focuses on the problem of Test Case Prioritization during Regression testing phase. The general definition of test-case prioritization can be given as [6]:

Given: TSU is a test suite.
PMT is a set of all possible permutations of test cases in TSU
To map PMT to real numbers a function f is defined.

Problem: Find TSU'€ PMT such that (∀TSU") (TSU"€ PMT)
(TSU"≠ TSU') [f(TSU') ≥ f(TSU")]

Here, PMT denotes the set of all probable orderings of TSU and f is a function that when applied to the orderings, provides an award value corresponding to that ordering. So, we want to have an arrangement of test cases that would achieve the highest award value among all the possible orderings. There can be many possible award values- rate of fault detection, coverage criteria, reliability *etc*. There are two types of test case prioritization [5]-

1) General Prioritization- In this, the same reordering of the test cases would be used in every upcoming version of the program or software, *i.e.*, in this type of prioritization without any knowledge of the modifications being applied to the program, the same solution is used for each subsequent version of a program or software.

2) Version Specific Prioritization- In this, we use the knowledge of changes being applied to a program or software to rearrange the test cases according to those specific changes and for each successive version the reordering is changed.

2.2 Clustering Based Test Case Prioritization

Clustering based test case prioritization technique considers the common property of the test cases and clusters them on the basis of their common features. Test cases are clustered in the sense that test cases within a cluster may have somewhat identical capability of detecting the faults and if a software testing team has lack of time to consider all the test cases to be executed at the time of regression testing then they can execute few of them from each cluster and can achieve approximately similar results. Also with the help of clustering we can find out any exceptional condition covered by a test case. Software artifacts are used to find out the common features of the test cases. For example, Coverage commonality among the test cases can be used to cluster them.

This approach includes two types of prioritization: Intra-cluster prioritization and Inter-cluster prioritization. Intra-cluster prioritization is done to get the prioritized list of test cases within each cluster, so that higher priority test cases can be executed first from each cluster. Inter-cluster prioritization prioritizes the overall test cases in all the test cases.

3. Related Works

In 1999 Rothermel *et al*. proposed various code coverage based test case prioritization techniques [5]. They proposed statement based, branch based, fault exposing potential (fep) based techniques. Rothermel and Elbaum suggested two main strategies for test case prioritization [7, 8]. Both strategies can be applied on any coverage criterion. Total strategy simply arranges the test cases in non-increasing order according to the number of statements covered by them, whereas, additional strategy sorts the test cases in decreasing order of covering the maximum statements not yet covered by previously executed test cases before any other previously unexecuted test cases. The techniques proposed by them include both total as well as additional strategies. They proposed total statement coverage, additional statement coverage, total branch coverage, additional branch coverage, total fep, additional fep based prioritization techniques. In 2002, some researchers have proposed function level prioritization in addition to statement level prioritization [9]. Li *et al*. suggested a framework that produces the test cases covering the points given higher preference by computing preferences using code coverage capability [10]. Aggrawal *et al*. suggested an approach based on code coverage criteria but this technique especially focuses upon version specific test case prioritization [11]. Srivastava *et al*. proposed a test case prioritization technique on the basis of coverage criteria to increase the value of APFD (Average Percentage of Faults Detected) metric [12]. In 2007 Bryce *et al*. proposed a test case prioritization technique using interaction coverage which was especially presented for Event-driven software [13]. In this test cases are prioritized on the basis of event interaction coverage and it uses the concept of software interaction testing. Zhang *et al*. proposed a new group of coverage based ART techniques [14]. This technique is based on the white-box coverage information. White-box ART prioritization technique at each step chooses the next test case that is farthest away from the already prioritized test cases. Harrold *et al*. presented modified condition/ decision coverage based prioritization approach [15]. MC/DC is a very powerful technique for verification and it is found that if test cases meet MC/DC coverage need then they can detect important errors that can't be found by functional testing. Jeffrey *et al*. presented a method of prioritization based on the relevant slices of outputs of various test cases [16]. This technique is a combination of total statement coverage criterion and also depends upon the number of statements covered by a test case that influence the output generated by that test case.

Under the umbrella of Human-based approaches Tonella *et al*. suggested case based ranking methodology which is based on machine learning algorithm that uses the user knowledge for prioritization [17]. This approach provides ranking with the help of two inputs- 1) a set of initial indicators of priority and 2) pairwise comparisons of test cases obtained from the expert. To enhance the scalability of human based approach, Yoo *et al*. proposed an approach that combines the pairwise comparison of test cases with clustering technique [18]. In this technique also prioritization is based on comparisons performed by human testers but test cases are clustered according to their similarity and these clusters are provided to the testers. The inter-class prioritization is performed by the tester whereas; intra-class prioritization is done on the basis of coverage criterion. Kim and Porter proposed the test case prioritization technique based on historical information about the execution of test cases in resource constrained environment [19]. Khalilian *et al*. proposed an approach on history based test case prioritization that uses the historical fault detection performance information of test cases for prioritization [20].

Srivastava *et al*. proposed a combination of requirement based and risk based prioritization where the task on hand is performed on the basis of identified requirements and risk factors [21]. It follows two level strategies for prioritizing the test cases which includes two priority factors- 1) Priority based on requirements provided by customers, developers and managers.2) severity of risks exist in requirements. The risk based prioritization technique proposed by Kavitha *et al*. attains the objective by ordering the test cases in a way to achieve maximum rate of fault identification with most severe defects identified at the earliest by test cases [22]. Krishnamoorthi *et al*. proposed a model for prioritization based on software requirement specification uses six factors to prioritize the test cases: Customer preference, modifications in requirement, complexity in implementation, completeness, traceability and influence of fault [23]. A research has been performed to propose fault based prioritization technique that incorporates fault localization approach [24]. This approach focuses on not only detecting the faults but also finding out where the set of faults located in a program so as to provide ease to debugging activity. Yu *et al*. proposed a fault based prioritization strategy for specification based testing that prioritizes the test cases based on their potential of detecting the faults [25]. The information about test cases and corresponding faults covered by them has been derived using specifications. So, this technique can also be used when source code is unavailable.

Mirarab and Tahvildari used Bayesian networks based on probability theory for prioritizing the test cases at regression testing phase [26]. This approach takes an advantage of augmenting different sources of information together into a single model for prioritization. It also prioritizes the test cases on the basis of their success probability. Korel *et al*. proposed a model based technique in which differences between the original and modified models are used to prioritize the test cases [27]. George *et al*. proposed heuristic methods for model based test case prioritization to overcome the complexity of model dependence based prioritization and to provide simple as well as efficient methods for the same [28]. Cost aware based techniques proposed by Elbaum *et al*. considers that the severity of each fault as well as cost of executing every test case together are two important factors for prioritization [29]. Thus main goal of cost aware based techniques is to execute test cases having minimum cost and highest rate of fault detection with detecting most severe faults earlier at the time of regression testing. A very effective technique of test case prioritization with the time budget factor using Integer Linear Programming (ILP) was suggested by Zhang *et al*. [30]. The two main goals to achieve while using this technique are- 1) we need to select the test cases, so that their execution time should not go beyond the time budget. 2) We need to prioritize the test cases so that they can detect the faults as early as possible. It has two main steps- 1) ILP is applied for test case selection. 2) Then for prioritizing the test cases any traditional technique can be used.

Various clustering based techniques have also been proposed by researchers. Table 1 summarizes these techniques.

Table 1. Summary of Existing Clustering Based Approaches

Title/Author	**Clustering Technique Used**	**Technique for prioritization**
Carlson *et al.* "A clustering approach to improving test case prioritization: An industrial case study" [31].	Agglomerative Hierarchical Clustering	1. Code Coverage- Method coverage 2. Code Complexity Metric- LOC and Method dependency count 3. Fault history information- Fault detection rate
Arafeen *et al.* "Requirement based clustering approach" [32].	K-Means	1. Lines of code 2. Nested block depth 3. McCabe's cyclomatic complexity
Jacob and Ravi "A Novel Approach For Test Suite Prioritization" [33].	K-Means	Cyclomatic Complexity
Badwal *et al.* "Test Case Prioritization using Clustering" [34].	Agglomerative Hierarchical Clustering	1. Statement coverage 2. Number of function calls

4. Proposed Methodology

The detailed steps of the proposed approach are shown in Figure 1. In this approach first we have selected the datasets, that is, a software application for which test cases were already provided. For clustering the test cases code coverage commonality information has been used between the test cases and Fuzzy C-means has been used as a clustering technique. After getting the clusters of test cases, fault history information, source code information including code complexity, test case execution history and execution time taken by the test cases have been used to re-arrange the test cases within the cluster. All these factors are necessary to assign the priority to the test cases. No technique available so far have used execution time factor for this task. So this proposed approach combines the idea of code coverage, fault-aware based, cost-aware based and clustering based approaches with the inclusion of execution time factor. After performing prioritization within the cluster, next step is to perform inter-cluster prioritization. For this we have used one-per cluster sampling approach in round-robin manner to decide the priority of test cases across the clusters. Using these final priorities, test cases are executed in that order to minimize the time, cost and effort and to maximize the rate of fault detection.

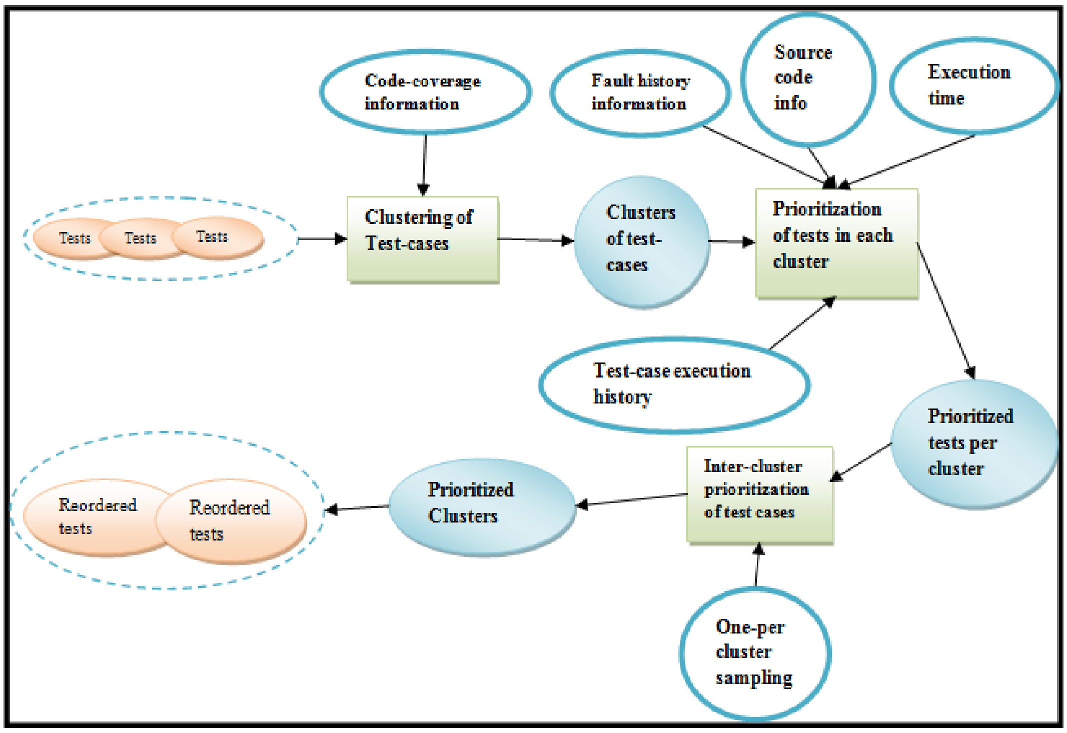
Code-coverage information
Fault history information
Source code info
Execution time
Tests
Tests
Tests
Clustering of Test-cases
Clusters of test-cases
Prioritization of tests in each cluster
Test-case execution history
Prioritized tests per cluster
Inter-cluster prioritization of test cases
Prioritized Clusters
Reordered tests
Reordered tests
One-per cluster sampling

Figure 1. Detailed Steps of Proposed Approach

4.1 Clustering of Test Cases

The first step of the proposed approach is clustering of the test cases on the basis of some common feature. The test cases have been clustered on the basis of their common coverage features. Line coverage, method coverage and branch coverage have been considered to group the test cases, in the sense that if test cases have some coverage commonality then they are somewhat similar type of test cases and they also have same potential of detecting the bugs. For clustering, Fuzzy C-means clustering method has been used [35-36]. The reason behind using FCM is that it may be the possibility while clustering the test cases that we may not get firm boundaries of the clusters. Test cases may belong to one or more clusters according to its commonality with test cases in different cluster.

4.1.1 Fuzzy C-Means

The other clustering techniques proposed so far are referred to as hard or crisp clustering, which means that each data point is allocated to only one cluster. For fuzzy clustering, this constraint is diminished, and the data point can be affiliated to all of the clusters depending on its membership value for each cluster. This concept is exceptionally helpful when there is ambivalence among the clusters' boundaries and they are not well defined. Furthermore, this approach is quite useful to identify more delicate relations between a data point and its corresponding clusters. The steps of algorithm used for clustering are:

Step 1: Initialize number of clusters K and cluster centre matrix M(0). Also initialize fuzzification parameter m=2.

Step 2: Use following equations to evaluate membership matrix W(0) :

$$W_{pq}^{(t)} = \frac{1}{\sum_{r=1}^{K} \left(\frac{d_{pq}}{d_{pr}}\right)^{\frac{2}{m-1}}} \quad (1)$$

If $d_{pq} = 0$ then $w_{pq} = 1$ and $w_{pq} = 0$ for $q \neq r$ (2)

Where, p=1,..,N and q, r= 1,..,K

Step 3: At each iteration increase t by 1 and recalculate centre matrix M(t) by using:

$$M_j^{(t)} = \frac{\sum_{p=1}^{N} (w_{pq}^{(t-1)})^m x_p}{\sum_{p=1}^{N} (w_{pq}^{(t-1)})^m} \quad (3)$$

Where, Xp is a data point.

Step 4: Recalculate membership matrix using equation (1) and (2).

Step 5: If | W(t)-W(t-1) | < ε then stop further iterations otherwise move back to step 3.

According to the membership value of each test case in each and every cluster, we assigned test cases to the cluster. Cluster centre matrix has been initialized using random generator function. While considering the distance function we used "Aggregate Quantifiable metric" defined in equation (4). This metric was proposed by Vangala *et al.* to be used especially in test case comparison and clustering task [37].

$$d(T_p, T_q) = \sum_{k=1}^{m} \frac{W_k * d_k(T_p, T_q)}{m} \quad (4)$$

Where, $d_k(T_p, T_q)$) illustrates the value and Wk depicts weight of the kth metric.

4.2 Intra Cluster Prioritization

After having the clusters of test cases, a new intra-cluster prioritization algorithm has been applied to assign priorities to the test cases within a cluster. For this purpose we have used the following metrics:

4.2.1 Code Coverage Metric

Code coverage metric provides the statistics about the part of code covered by a single test case from the total covered code by all the test cases. We have considered statement, method and branch coverage and computed the coverage ratio of each test case. This metric has been used because larger the part of code covered by a test case, higher should be its priority.

4.2.2 Test-Case Failure Rate

This metric has been considered because here the task is to prioritize the test cases at the regression testing phase where the information of the previous running status of the test cases is already available and if a test case has failed many times then the fault covered by it, is still unresolved, and thus it should get the higher priority to be executed for the next time.

4.2.3 Fault Detection Ratio

Fault detection ratio of each test case depicts the capability of detecting faults by a test case and a test case is more important to execute if it has more fault detection capability.

4.2.4 Execution Time

Execution time taken by each test case is also very important information to be included while performing the test case prioritization and to get good results. If time taken by a test case is low then it should get higher priority to get executed.

4.2.5 Code Complexity Metric

It provides the information about the complexity level of the code. Higher complex code requires more vigorous testing efforts. The formula of each metric is shown in Table 2.

Table 2. Metrics and Related Formulae

Metric	Formula
Code Coverage Metric (Statement Coverage)	$\frac{\text{Statements covered by a test case}}{\text{total number of covered statements}}$
Test-case failure rate	$\frac{\text{Number of times test case has failed}}{\text{No. of times it has been executed}}$
Fault detection ratio	$\frac{\text{Number of detected faults by a test case}}{\text{Total number of faults}}$
Code Complexity Metric	Complexity of a test case is equal to the average of the complexities of classes covered by that test case.

4.3 Algorithm for Intra-Cluster Prioritization

```
Input: Clusters C1,C2,..,Cn having t1, t2,..,tn test cases in each.
Output: A prioritized list of test cases.
1. Start
2. For each cluster, do
3.      for all t in C do
4.      TFRt← Calculate test-case failure rate from previous execution history.
5.       Calculate coverage ratio CRt← coverage of (line + branch + method)/3
6.        Et← Determine execution time of test case t.
7.        CCt← Compute code complexity for t.
8.        FDRt← Compute fault detection ratio for t.
9.      for all t in C do
10.      Normalize TFR, CR, E, CC, FDR.
11.      Priorityt= (TFRt + CRt + CCt + FDRt) - Et
12.     Sort test cases according to their test priority.
13. End
```

4.4 Inter Cluster Prioritization

After getting the prioritized test cases, the test cases have been executed in round-robin pattern from each cluster. We started this process with the highest priority test case from first cluster and executed it, and then the highest priority test case from second cluster has been selected and so forth. After executing one test case from each cluster the process has been repeated for the second highest priority test cases. In this way inter-cluster prioritization has been achieved and if an organization cannot spend its time to execute all the test cases, it can skip the execution of low priority test cases from each cluster. This step would finally decide the priority of a test case for execution. Figure 19 is showing this inter cluster prioritization approach.

5. Experimental Studies

Some empirical studies have been performed to assess the potential of the proposed approach in prioritizing the test cases. The plot of conducting the experiments is illustrated here.

5.1 Research Questions

This approach has been proposed to handle the challenges of regression testing in best possible ways, to perform the regression testing with minimum resources like, time and cost and to achieve higher average percentage of faults detected. This work considers and will answer the following research questions:

RQ1: How to increase the rate of fault detection at the time of regression testing?

RQ2: Whether clustering approaches perform better than non-clustering approaches?

RQ3: What is the effect of considering other important factors over considering only code coverage metric for prioritization?

RQ4: How to reduce the rate of execution time?

5.2 Variables and Measures

5.2.1 Independent Variables

The independent variable for conducting this study is test case prioritization method. To determine the effectiveness of our proposed approach six control techniques and the proposed technique have been considered to be compared. These seven techniques are mentioned in Table 3.

Table 3. Description of Techniques Under Experiment

Label	Description
Torig	Random ordering of Test Cases
Tcodecov	Prioritized using code coverage only without clustering
Tallfac	Prioritized using all metrics mentioned in Chapter 3 without clustering
Tkmcodecov	Clustering using K-means and prioritized using code coverage only
Tkmallfac	Clustering using K-means and prioritized using all factors
Tfcmcodecov	Clustering using FCM and prioritized using code coverage only
Tfcmallfac	Clustering using FCM and prioritized using all factors

These techniques have been considered to compare the results of using K-means and FCM clustering techniques, to compare the results between clustering and non-clustering based techniques and to compare the results between techniques using code coverage only and using all factors. The motive is to show the effect of clustering based approach over non-clustering based approaches, chosen clustering technique and effect of considering all metrics over code-coverage based only.

5.2.2 Dependent Variables

To answer our research question 2 and to prove the validity and benefits of the proposed technique over the previous available techniques, it is required measuring the value of rate of fault detection. So, it is needed to evaluate the value of APFD (Average percentage of faults detected) metric [38]. Higher APFD value indicates better performance of the corresponding prioritization technique. Suppose "TS be a test suite having m test cases and F be a set of n faults. Let TCFi be the first test case in prioritized order which is covering fault i." The APFD is expressed as:

$$APFD = 1 - \frac{TCF_1 + TCF_2 + \cdots + TCF_n}{mn} + \frac{1}{2m} \qquad (5)$$

To answer the research question 4, the considered dependent variable is execution time taken by the prioritized test cases that are contributing in achieving the higher APFD value. As a part of this research work, the execution time taken by the techniques under consideration has been evaluated and compared in order to access the effectiveness of the proposed technique in reducing the execution time.

5.3 Object Programs

Two software applications Apache ant and JMeter have been selected to perform this work. Two applications have been used to evaluate and also validate the results. Table 4 shows the subject programs used to perform this experiment. Apache ant is a Java library and build tool similar to make [39]. JMeter is a pure Java application used to perform load and performance testing of web applications [40]. These programs and their test cases have been taken from Software Artifact Infrastructure Repository [41]. So, both are Java based applications and their test cases' available in the repository are JUnit test cases.

Table 4. Datasets and Related Information

Subject	Versions	Size	Test Classes	Faults
Ant	9	627	150	21
JMeter	6	389	28	9

Ant dataset has 9 original and fault seeded version. Here, number of test cases in dataset are 150 and number of faults seeded are 21. For JMeter dataset, 6 original and fault seeded versions are available with 28 test cases and 9 faults. Their fault seeded versions have been used to perform this work. For each version different numbers of test cases are applicable. Maximum number of test cases applicable for any version is mentioned in Table 4.

5.4 Experiment Setup and Procedure

5.4.1 Experiment Environment

The environment to perform these experiments consist of a machine having Windows 7 as an operating system, Intel Core i3 processor with 4 GB RAM. As a programming language, Java has been used throughout this experimental study.

5.4.2 Data Collection

After finalizing the object of analysis, the next step is to collect data to perform clustering and prioritization. Data that need to be collected are: JUnit test suite, Coverage information, execution time, test case failure rate, and fault detection ratio and code complexity information.

After having datasets with JUnit test cases, test suites with 150 test cases in Apacahe Ant application and 32 test cases in JMeter application using Eclipse IDE have been developed. For developing these test suites JUnit environment has been used and included junit3.8.1 library in our project. The test cases added in this suite can be executed together by only executing this test suite using JUnit. The test cases are executed in the order in which they have been added in this suite. Initially they have been added in random order.

To perform clustering of test cases we have used Java programming language in Eclipse IDE. Eclipse IDE has been used throughout the implementation process. Two clustering techniques have been implemented to compare the results- K-Means and Fuzzy C-means. K-means have been used in previous researches of test case prioritization and my proposed approach considers FCM for this task. To group the test cases, their coverage ratio feature have been used. So, at this stage the first task is finding out the coverage ratio of each test case. To get the coverage information, EMMA code coverage tool has been used. This tool has been integrated with Eclipse and JUnit framework in our project by installing its plug-in.

We ran this tool with JUnit test suite, and collected the coverage information for all the test cases. Covered instructions, branches and methods by each test case have been gathered and the statement, branch and method coverage ratios have been calculated using the formula mentioned in Table 2. EMMA tool has also been used to collect complexity metric values for each test case. This tool provides Complexity values as a total complexity of class and how much complexity has been covered by test cases covering that class. Complexity of a test case is equal to the average of the complexities of classes covered by that test case. We have generated complexity report in XML file and converted into CSV file to further use this information for applying intra-cluster prioritization approach.

To obtain test case failure ratio first data from test case execution history of previous runs of test cases have been gathered through previous versions. We obtained information about how many times each test has been executed and how many times it has been failed through JUnit test suite execution of previous versions. This test suite execution history is available in XML file which contain information about all the previous runs of test cases and their pass status has been shown through value 1 and value 0 for a fail test case. The data of this xml file has been used to calculate test case failure ratio of each test case. The result of this module has been given as input to the intra-cluster prioritization algorithm.

To calculate the fault detection ratio, first the fault matrix using Gen_fault_matrix tool has been produced. This tool generates fault matrix in the form of Universe file which can be opened in Notepad. This matrix shows the fault id and test number. In this, 0 indicates that this fault is not covered by the mentioned test case and 1 indicates fault covered by this test case. The formula mentioned in Table 2 has been used to calculate fault detection ratio. The result of this module has been given as input to the intra-cluster prioritization algorithm. To obtain the execution time taken by each test case, simply the JUnit test suite execution report has been used and exported in XML format.

5.4.3 Procedure

Figure 2 shows the blueprint of the experimental setup. This Figure presents an abstract view of the implementation process. After gathering all the required data, the test cases have been clustered by K-means and FCM using coverage commonality. “Aggregate Quantifiable metric” has been used as a similarity metric and to calculate the distance between the two test cases. The difference in their statement; branch and method coverage ratio has been used and their weightage have been taken equally as 1. The experiment has been performed by varying cluster numbers for Apache ant as 5 and 10 and for JMeter 4 and 6 numbers of clusters have been taken.

After clustering intra and inter cluster prioritization have been performed and APFD value has been computed. First intra-cluster prioritization has been performed using K-means clustering result and the proposed algorithm, and then the same task has been performed using FCM and the proposed algorithm. We have calculated priority metric value of each test case by using the formula mentioned in proposed algorithm and rearranged the test cases in each cluster. After having prioritized test cases within each cluster, final priorities of the test cases have been obtained across the cluster by selecting one test case from each cluster starting with highest priority test case from 1st cluster and so on in round robin manner.

After getting final priorities of the test cases, APFD value has been calculated and a new test suite has been developed in which test cases have been added in final priority order. We have executed this test suite and evaluated execution time to compare it with the execution time taken by the random order test suite.

6. Experimental Results

6.1 Answer to RQ1: APFD Value

To answer the research question 1, we started our analysis with APFD value comparison of all the considered techniques for Apache Ant dataset for all the 8 versions. The results will clearly show that Tfcmallfac outperforms over all other techniques and the proposed approach is able to achieve higher value of average percentage of faults detected than other considered techniques. The lowest value of APFD is given by Torig, that is, when any particular ordering on the test cases has not been applied. This comparison has been performed for cluster numbers 5 and 10 and for both; the pattern of results is same. APFD value for Torig is 51.55% for version 1 which is the lowest among all other techniques and APFD value for Tfcmallfac is 87.35% which is the highest among the others.

Approximately across all the versions the results are same. Also the cluster numbers has been varied and it was found that there is no improvement in APFD value even if the cluster numbers are increased. The results of Apache Ant dataset with cluster number 5 and 10 are shown in Figure 3 and 4.

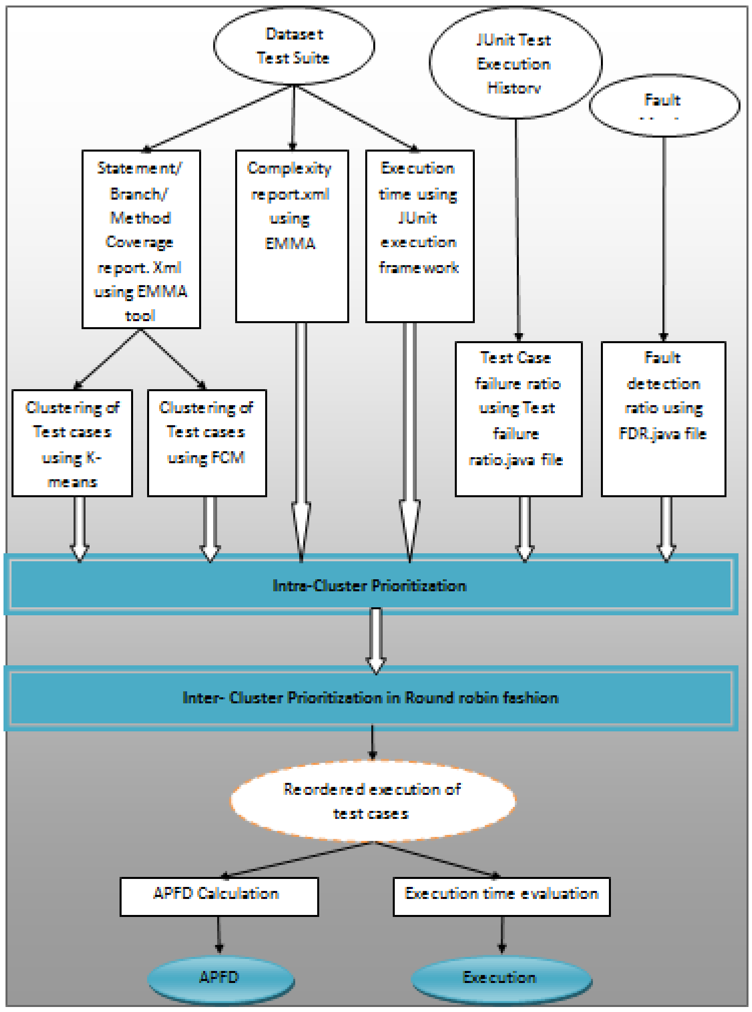
Dataset Test Suite
JUnit Test Execution History
Fault
Statement/ Branch/ Method Coverage report. Xml using EMMA tool
Complexity report.xml using EMMA
Execution time using JUnit execution framework
Test Case failure ratio using Test failure ratio.java file
Fault detection ratio using FDR.java file
Clustering of Test cases using K-means
Clustering of Test cases using FCM
Intra-Cluster Prioritization
Inter- Cluster Prioritization in Round robin fashion
Reordered execution of test cases
APFD Calculation
Execution time evaluation
APFD
Execution

Figure 2. Experimental Setup

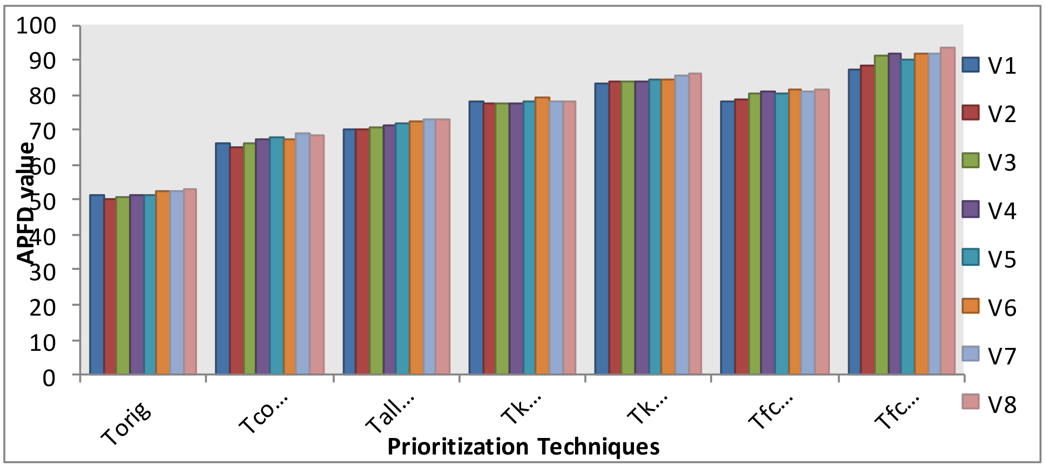

Figure 3. APFD Values for Apache Ant with Cluster Numbers 5

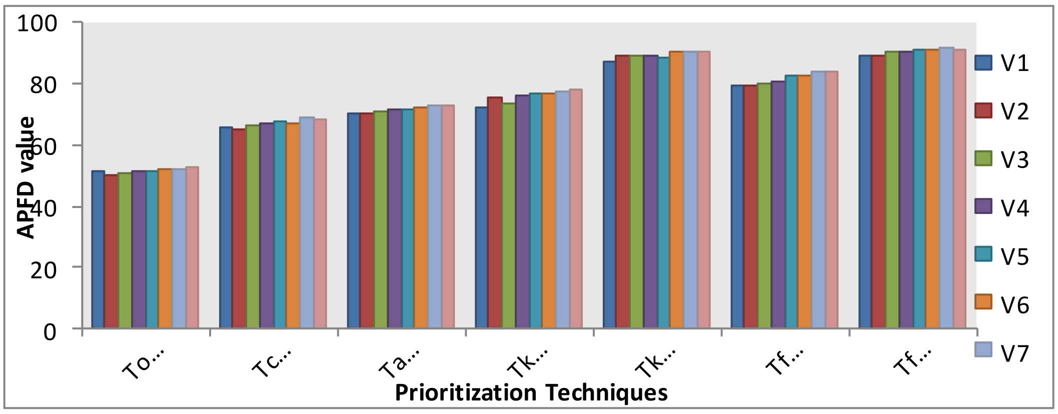

Figure 4. APFD Values for Apache Ant with Cluster Numbers 10

In the above Figures, it can be seen that the APFD values achieved by Tkmallfac is less than Tfcmallfac but higher than Tfcmcodecov only. Also Tallfac is having higher value than Tcodecov.

To validate these results, same experiment has been performed with JMeter dataset across all 5 versions having cluster numbers 4 and 6 and results obtained are shown in Figures 5 and 6. For version1 and cluster numbers 4 the APFD value achieved by Tcodecov is 66.12% whereas Tfcmallfac gives 90.75%. Here, also Tkmallfac outperforms Tfcmcodecov.

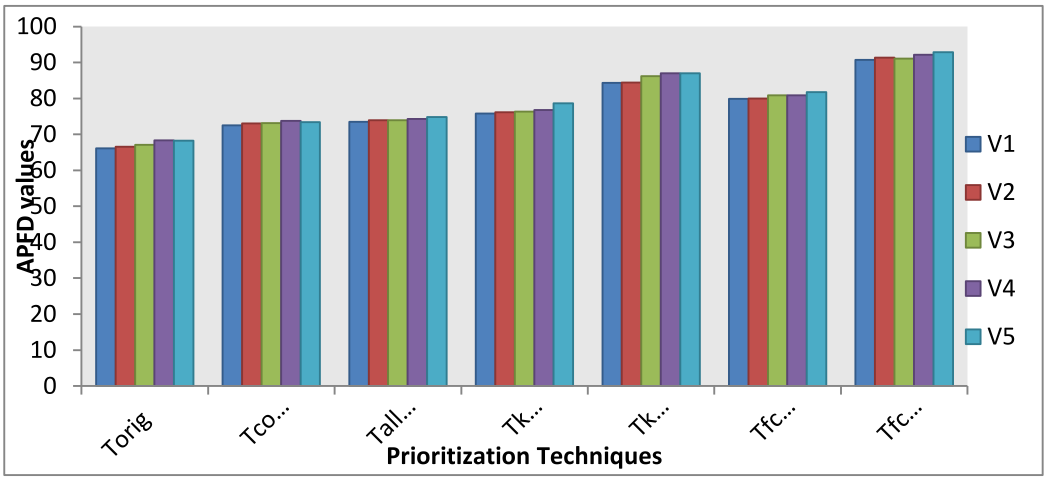

Figure 5. APFD Values for JMeter with Cluster Numbers 4

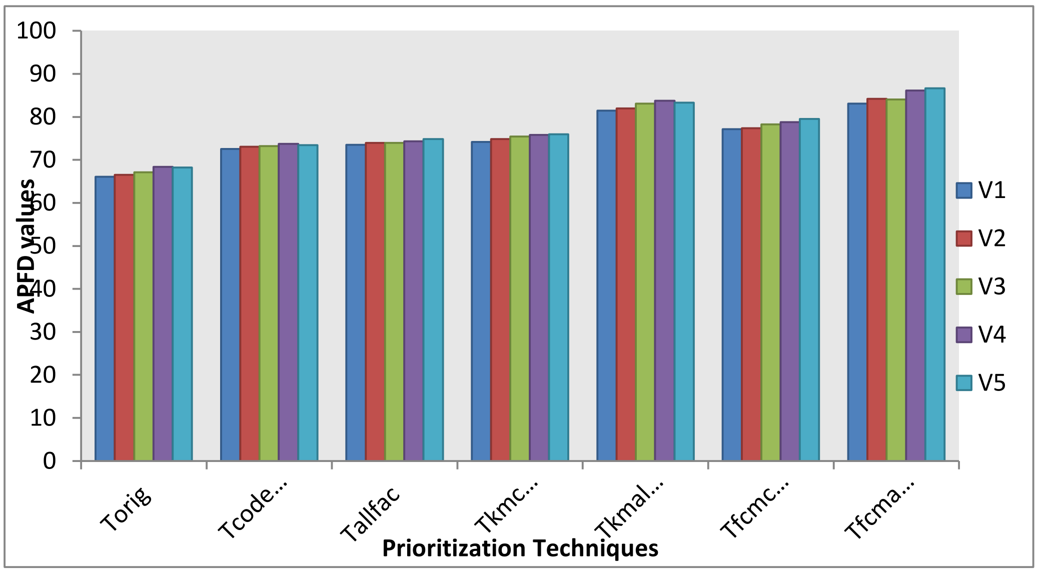

Figure 6. APFD Values for JMeter with Cluster Numbers 6

Table 5 shows the improvement of Tfcmallfac over all other considered techniques. It can be seen that results are improved with Tfcmallfac. The maximum improvement has been achieved over Torig. Results have also been improved over Tkmallfac. Also, from the results it is clear that by increasing the cluster numbers the results are not getting more improved. The similar improvement patterns have been observed for all other versions of both datasets.

Table 5. Improvement % of Tfcmallfac over Other Techniques

Dataset	Cluster	Torig	Tcodecov	Tallfac	Tkmcodecov	Tkmallfac	Tfcmcodecov
Ant Version-8	**5**	77.51	36.70	28.09	19.69	9.11	14.65
	10	72.69	32.99	24.61	16.19	1.02	8.59

JMeter Version-5	**4**	36.15	26.49	24.14	18.11	6.79	13.59
	6	26.99	17.98	15.78	14	8.95	3.95

6.2 Answer to RQ2

To answer the research question 2, clustering based techniques and non-clustering based techniques have been compared. Table 6 represents that clustering improved the APFD value over all non-clustering techniques irrespective of the metrics considered for prioritization. The similar improvement patterns have been observed for all other versions of both datasets.

Table 6. Improvement % of Clustering Techniques over Non-clustering Techniques

Dataset	**Ant Version-8**				**JMeter Version-5**			
Cluster	5		10		4		6	
Techniques	**Tcodecov**	**Tallfac**	**Tcodecov**	**Tallfac**	**Tcodecov**	**Tallfac**	**Tcodecov**	**Tallfac**
Tkmcodecov	14.21	7.01	14.46	7.25	7.09	5.11	3.49	1.56
Tkmallfac	25.29	17.39	31.65	23.36	18.44	16.24	13.50	11.39
Tfcmcodecov	19.24	11.72	22.48	14.75	11.36	9.29	8.28	6.27
Tfcmallfac	36.70	28.09	32.99	24.61	26.49	24.14	17.98	15.78

6.3 Answer to RQ3

The next comparison has been made in between techniques including code coverage, complexity, execution time, test case failure ratio and fault detection ratio factors and techniques including code coverage only. Table 7 shows the results of this comparison and it can be concluded that Clustering is more important to be considered over all factors because Tallfac gives negative improvement over Tkmcodecov and Tfcmcodecov.

But, it is also showing that techniques considering both clustering and all factors are better than anyone. So, from the results, it can be clearly concluded that the prioritization technique considering FCM and all metrics mentioned in Table 2 gives best results over all other techniques.

Table 7. Comparison of Techniques Including Test Case Failure Ratio, Fault Detection Ratio and Execution Time and Techniques Excluding these Factors

	Excluding all factors		
	Dataset	**Ant Version-8**	**JMeter Version-5**

Including all factors	Techniques	Tcodecov	Tkmcodecov	Tfcmcodecov	Tcodecov	Tkmcodecov	Tfcmcodecov
	Tallfac	6.73	-6.56	-1.05	1.39	-4.86	-8.50
	Tkmallfac	25.29	9.70	5.08	18.44	10.59	6.36
	Tfcmallfac	36.70	19.69	14.65	26.49	18.11	13.59

6.4 Answer to RQ4: Execution Time

If the execution time taken by all the techniques is compared then it is found that Tcodecov is taking maximum time as 131.35 seconds to execute all the test cases for Apache ant version 8 whereas time taken by Torig is 109.02 seconds. Tfcmallfac is taking 10.48 seconds with number of clusters 5 and 17.84 seconds with cluster numbers 10. It is clear from the results that all techniques those including execution time factors for prioritization are taking less execution time than techniques that do not consider this factor. Figure 7 shows the graph of execution time taken by each technique to execute the test suite.

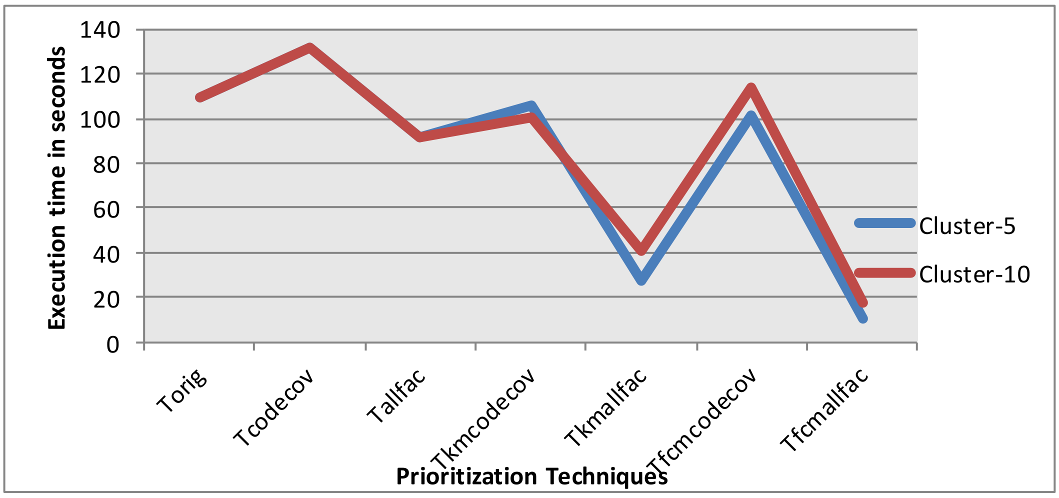

Figure 7. Execution Time for Apache Ant Version-8

The results are same with JMeter also. Here, time taken by Torig is 13.53 seconds and by Tallfac is 10.65 seconds. Figure 8 shows the results of JMeter execution time analysis with all the techniques under experiment.

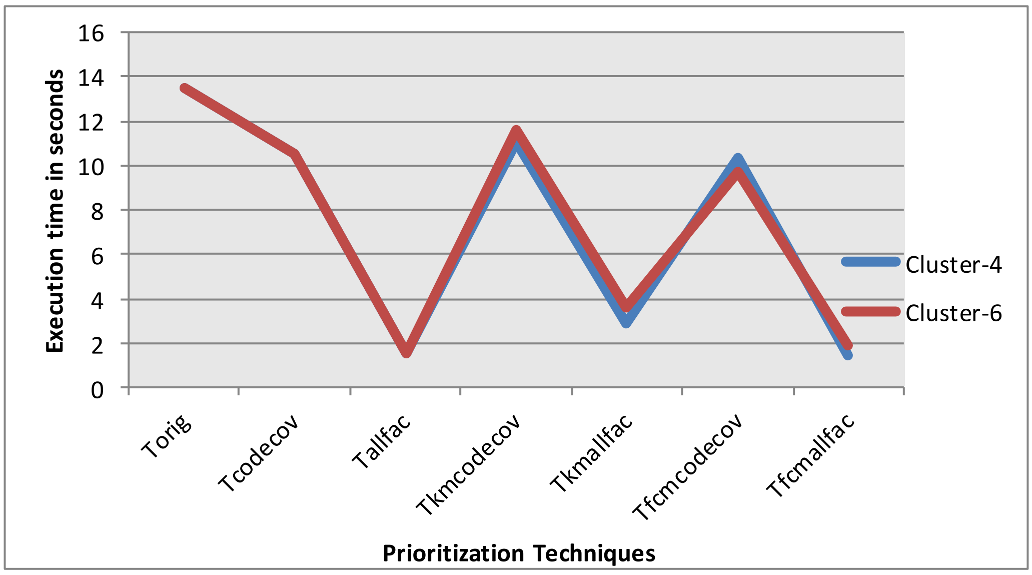

Figure 8. Execution Time for JMeter Version-5

7. Threat to Validity

7.1 Threat to Internal Validity

Every research work faces some limitations to its validity and this may face the same. There are some threats to internal validity that involves considering the accuracy in performing the experiments. The perfection of experiment depends on the accuracy in the results provided by tools used, metrics considered, precision in performing clustering task and other data. The results of EMMA tool and gen_fault_matrix tool have a great impact on the internal validity. Also the clustering technique FCM affects the performance of the experiment. Considering this, widely acceptable FCM clustering algorithm has been used. The execution time and test case failure ratio have been obtained through JUnit execution report. So, accuracy of these data depends upon the performance of JUnit.

7.2 Threat to External Validity

Threat to external validity is all about the universalization of the results of the research work that is up to which level the results can be generally applicable. This includes conditions for which results can be generalized and limitations on these results. Here, in this approach the combination of clustering technique and time aware based technique with intra and inter-cluster prioritization has been proposed which is itself a novel approach. We have used FCM for clustering and code coverage, execution time, test case execution history, complexity and fault detection ratio for prioritization. But other combinations of using different clustering technique and different metrics may produce different results. So, these results are generally applicable if only using FCM and mentioned metrics. Also, the experiment has been performed only with JUnit test cases based on Java applications. So, the main threat to external validity is to use this approach with other programming language applications and other type of test cases. The size of the dataset, that is, the number of test cases varied from lesser to large number of test cases. So, results can be generalized with any number of test cases.

8. Conclusion and Future Work

Regression testing is performed at maintenance phase which is a very crucial phase of the whole Software Development Life Cycle. So, it is important that cost and time effective testing techniques be applied during maintenance. Test case prioritization is an efficient way to perform the regression testing. Various test case prioritization techniques are currently being used by many organizations. Clustering based test case prioritization techniques are the best among others because this technique can be used with the combination of other techniques but the existing clustering based techniques have some shortcomings. This research work is an attempt to improve the clustering approach of test case prioritization and used FCM clustering technique to cluster the test cases and performed intra and inter- cluster prioritization using the important factors like test case failure ratio, code coverage ratio, fault detection ratio and complexity. In this approach along with other factors, we have also considered the execution time of each test case while prioritizing them.

The main aim of this work is to obtain higher APFD value in minimum execution time. We have calculated the APFD values of the systems under test across all their versions using all the techniques under consideration with proposed one and it was found that the proposed approach provides better APFD value than other approaches. The results have been compared between clustering and non-clustering techniques and conclusion has been drawn that including clustering in prioritization improves result over non-clustering ones. It is also concluded that FCM performs better than K-means when both are applied with all factors and importance of clustering is more than considering all factors without clustering. In the proposed approach execution time has been used as one of the factors to prioritize the test cases. The test cases having lesser execution time will get the higher priorities and will be executed first. So, more test cases can be run in less time. Eventually execution time would be reduced and it was found that this approach takes least execution time. It is also concluded that techniques including execution time to prioritize the test cases take less time than techniques without considering execution time factor.

In the future work this technique can be applied with applications developed in other programming languages. This research has been performed with JUnit test cases, which can be extended with other programming languages like C, C++ *etc.* and test cases like TestNG, TSL *etc.* In future research, other important factors can also be considered for prioritization and more sophisticated features can be selected to cluster the test cases.

References

[1] K. Onoma, W. T. Tsai, M. Poonawala and H. Suganuma, “Regression testing in an industrial environment”, Comm. of the ACM, vol. 41, no. 5, **(1988)**, pp. 81–86.
[2] B. Beizer. “Software Testing Techniques”, Van Nostrand Reinhold, New York, NY, **(1990)**.
[3] H. Leung and L. White, “Insights into Regression Testing”, In Proceeding of the Conf. on Software Maintenances, **(1989)**, pp. 60-69.
[4] G. Rothermel, R. H. Utnch, C. Chu and M. J. Harrold, “Test Case Prioritization: An Empirical Study”, Proceedings of the International Conference on Software Maintenance, Oxford, UK, September, IEEE, **(1999)**.
[5] G. Rothermel, R. H. Utnch, C. Chu and M. J. Harrold, “Test Case Prioritization Technical report”, GIT-99-28, **(1999)**.
[6] S. Yoo and M. Harman, “Regression Testing Minimization, Selection and Prioritization”, Software Testing, Verification and Reliability, **(2007)**.
[7] E. Sebastian, G. M. Alexey and G. Rothermel, “Prioritizing Test Cases for Regression Testing”, ISSTA '00, Portland, Oregon, ACM, **(2000)**.
[8] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, “Prioritizing Test Cases For Regression Testing”, CSE Journal Articles. Paper 9, **(2001)**.

[9] E. Sebastian; A. G. Malishevsky and G. Rothermel, “Test Case Prioritization: A Family of Empirical Studies”, CSE Journal Articles, **(2002)**.

[10] J. J. Li, D. Weiss and H. Yee, “Code-coverage Guided Prioritized Test Generation”, Information and Software Technology (Elsevier), **(2006)**.

[11] K. K. Aggrawal, Y. Singh and A. Kaur, “Code Coverage Based Technique For Prioritizing Test Cases For Regression Testing”, ACM SIGSOFT Software Engineering Notes, **(2004)**.

[12] P. R. Srivastava, “Test Case Prioritization”, Journal of Theoretical and Applied Information Technology ©2005, **(2008)**.

[13] R. C. Bryce and A. M. Memon, “Test Suite Prioritization by Interaction Coverage”, DoSTA’07, September 4, 2007, Dubrovnik, Croatia, ACM, **(2007)**.

[14] B. Jiang, Z. Zhang, W. K. Chan and T. H. Tse, “Adaptive Random Test Case Prioritization”, IEEE/ACM International Conference on Automated Software Engineering, **(2009)**.

[15] J. A. Jones and M. J. Harrold, “Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage”, College of computing, Georgia Institute of Technology.

[16] D. Jeffrey and N. Gupta, “Test Case Prioritization Using Relevant Slices”, Journal of Systems and Software, vol. 81, no. 2, **(2008)**.

[17] P. Tonella, P. Avesani and A. Susi, “Using the Case-Based Ranking Methodology for Test Case Prioritization”.

[18] S. Yoo, M. Harman, P. Tonella and A. Susi, “Clustering Test Cases to Achieve Effective & Scalable Prioritizations Incorporating Expert Knowledge”, ISSTA’09, July 19–23, 2009, Chicago, Illinois, USA, ACM, **(2009)**.

[19] J. M. Kim and A. Porter, “A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments”, 1CSE'02, May 19-25, 2002, Orlando, Florida, USA. ACM, **(2002)**.

[20] Y. Fazlalizadeh, A. Khalilian, M. A. Azgomi and S. Parsa, “Incorporating Historical Test Case Performance Data and Resource Constraints into Test Case Prioritization”, Springer-Verlag Berlin Heidelberg, **(2009)**.

[21] P. R. Srivastva and K. K. G. Raghurama, “Test Case Prioritization Based on Requirements and Risk Factors”, ACM SIGSOFT Software Engineering Notes, vol. 33, no. 4.

[22] R. Kavitha and Dr. N. Sureshkumar, “Test Case Prioritization for Regression Testing based on Severity of Fault”, (IJCSE) International Journal on Computer Science and Engineering, vol. 2, no. 5, **(2010)**, pp. 1462-1466.

[23] R. Krishnamoorthi and S. A. S. A. Mary, “Factor Oriented Requirement Coverage based System Test Case Prioritization of New and Regression Test Cases”, Information and Software Technology 51 (Elsevier), **(2009)**.

[24] S. Kim and J. Baik, “An Effective Fault Aware Test Case Prioritization by Incorporating a Fault Localization Technique”, ESEM’10, September 16–17, 2010, Bolzano-Bozen, Italy, ACM, **(2010)**.

[25] Y. T. Yu and M. F. Lau, “Fault-based Test Suite Prioritization for Specification based Testing”, Information and Software Technology 54 (Elsevier), **(2012)**.

[26] S. Mirarab and L. Tahvildari, “A prioritization approach for software test cases based on bayesian networks”, Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering, Springer–Verlag, **(2007)**, pp. 276–290.

[27] B. Korel, L. H. Tahat and M. Harman, “Test Prioritization Using System Models”.

[28] B. Korel, G. Koutsogiannakis and L. H. Tahat, “Model-Based Test Prioritization Heuristic Methods and Their Evaluation”, AMOST'07, July 9-12, 2007, London, UK, ACM, **(2007)**.

[29] A. G. Malishevsky, J. R. Ruthru, G. Rothermel and S. Elbaum, “Cost-cognizant Test Case Prioritization”, Technical Report TR-UNL-CSE-2006-0004, Department of Computer Science and Engineering, **(2006)**.

[30] L. Zhang, S. S. Hou, C. Guo, T. Xie and H. Mei, “Time-Aware Test-Case Prioritization using Integer Linear Programming”, ISSTA’09, July 19–23, 2009, Chicago, Illinois, USA, ACM, **(2009)**.

[31] R. Carlson, H. Do and A. Denton, “A Clustering approach to improving Test Case Prioritization: An Industrial Case Study”.

[32] Md. J. Arafeen and H. Do, “Test Case Prioritization Using Requirements-Based Clustering”.

[33] T. P. Jacob and T. A. Ravi, “A Novel Approach for Test Suite Prioritization”, Journal of Computer Science, vol. 10, no. 1, **(2014)**, pp. 138-142.

[34] J. Badwal and H. Raperia, “Test Case Prioritization using Clustering”, International Journal of Current Engineering and Technology, vol. 3, no. 2, **(2013)**.

[35] J. C. Bezdek, “Cluster validity with fuzzy sets”, Cybernetics, vol. 3, **(1974)**.

[36] J. C. Bezdek, “Pattern recognition with fuzzy objective function algorithm”, Plenum Press, New York, **(1981)**.

[37] V. Vangala, J. Czerwonka and P. Talluri, “Test case comparison and clustering using program profiles and static execution”, ESEC/FSE ’09, ACM.

[38] S. Elbaum, A. G. Malishevsky and G. Rothermel, “Test Case Prioritization: A family of Empirical Studies”, CSE Journal Articles. Paper 8, **(2002)**.

[39] Apache Ant, “http://ant.apache.org.”, accessed on 15th March, **(2015)**.

[40] JMeter, “http://jakarta.apache.org/jmeter”, accessed on 20th March, **(2015)**.

[41] Software Artifact Infrastructure Repository, “http://sir.unl.edu/portal/index.php”, accessed on 10th March, **(2015)**.

Authors

Ms. Geetanjali Chaurasia, She is perusing M. Tech. Software Engineering from Department of Information Technology, Indian Institute of Information Technology, Allahabad. Her area of interest in research includes Software Engineering, Software Testing specially in the field of test case prioritization using machine learning approaches. Her research topic in M. Tech. is — A Clustering based Novel Test Case Prioritization Approach for Regression Testing

Dr. Sonali Agarwal, is working as an Assistant Professor in the Information Technology Department of Indian Institute of Information Technology (IIIT), Allahabad, India. Her primary research interests are in the areas of Data Mining and Software Engineering. Her current focus in last few years is on the research issues in Twin Support vector machine, Big Data Mining and Stream Computing.