symmetry

MDPI

Article

Novel Fuzzy Clustering Methods for Test Case Prioritization in Software Projects

A. D. Shrivathsan [1], K. S. Ravichandran [1], R. Krishankumar [1], V. Sangeetha [1], Samarjit Kar [2,*], Pawel Ziemba [3] and Jaroslaw Jankowski [4]

1 School of Computing, SASTRA University, Thanjavur 613401, TN, India; shrivathsan.a.d@mca.sastra.edu (A.D.S.); raviks@sastra.edu (K.S.R.); krishankumar@sastra.ac.in (R.K.); sangeetha@sastra.ac.in (V.S.)
2 Department of Mathematics, National Institute of Technology, Durgapur, West Bengal 713209, India
3 Faculty of Economics and Management, University of Szczecin, Mickiewicza 64, 71-101 Szczecin, Poland; pawel.ziemba@usz.edu.pl
4 Department of Computer Science and Information Technology, West Pomeranian University of Technology, Zolnierska 49, 71-210 Szczecin, Poland; jjankowski@wi.zut.edu.pl
* Correspondence: samarjit.kar@maths.nitdgp.ac.in

Received: 9 August 2019; Accepted: 5 October 2019; Published: 12 November 2019

Abstract: Systematic Regression Testing is essential for maintaining software quality, but the cost of regression testing is high. Test case prioritization (TCP) is a widely used approach to reduce this cost. Many researchers have proposed regression test case prioritization techniques, and clustering is one of the popular methods for prioritization. The task of selecting appropriate test cases and identifying faulty functions involves ambiguities and uncertainties. To alleviate the issue, in this paper, two fuzzy-based clustering techniques are proposed for TCP using newly derived similarity coefficient and dominancy measure. Proposed techniques adopt grouping technology for clustering and the Weighted Arithmetic Sum Product Assessment (WASPAS) method for ranking. Initially, test cases are clustered using similarity//dominancy measures, which are later prioritized using the WASPAS method under both inter- and intra-perspectives. The proposed algorithms are evaluated using real-time data obtained from Software-artifact Infrastructure Repository (SIR). On evaluation, it is inferred that the proposed algorithms increase the likelihood of selecting more relevant test cases when compared to the recent state-of-the-art techniques. Finally, the strengths of the proposed algorithms are discussed in comparison with state-of-the-art techniques.

Keywords: regression testing; test case prioritization; grouping technology; clustering; WASPAS

1. Introduction

The quality of software can be improved through a software testing process. However, a software testing process is a tedious process, and it consumes more testing time. Strenuous efforts need to be vested if testing happens to be in the cycle, which is otherwise known as regression testing (RT) [1–4]. RT is a continuous process, and it has to be stopped at an apt time, where a reasonable number of bugs have been fixed. During every cycle of testing, bugs are found and fixed. Debugging may impact the lines of code that result in new errors. These new errors are fixed in subsequent iterations. RT ascertains that changes in software during bug fixing will not affect the overall quality and performance of the software product being developed [5].

Chittimalli and Harrold [6] proved that the RT is more expensive, and it requires more than 33% of the cumulative expenses of the software. Yoo and Harman [7] examined various regression test cases and showed the importance of a reduction of time in the test suite in RT. The reduction in time

for software testing mainly depends on any one of the following strategies: (i) Test Suite Minimization (TSM)—a methodology of removal of repetitive experiments to reduce the number of tests to run [8], (ii) Test Case Selection (TCS)—another way of reducing the number of tests to run [9] by way of modifications, and (iii) Test Case Prioritization (TCP) [10,11]—the most significant approach to reduce the number of tests to run by way of optimization.

RT should not be obviated because of its substantial impact on a time constraint. Rather, test cases in the test suite should be prioritized. Many ways of prioritizing test cases are based on their fault, exposing potential. The factors towards fault exposure are purely a white-box approach having a focus on branch coverage, and statement coverage [12]. Huang et al. [5] focused on the past data, such as the severity of the fault identified, the time spent for exercising the relevant test case, and the very first test case that identified specific severe fault, which is the backbone for finding the cost-cognizant average of faults detected. Keeping this as a fitness function and applying genetic algorithm resulted in optimized ordering of test cases which was a breakthrough in regression test prioritization.

The historical information about each test case reveals the number of times exercised, code coverage, and the iteration number in which the fault was uncovered, ends up in discovering an innovative equation setting up the priority numbers to the test cases [12]. Further, Huang et al. [13] prioritized test cases for GUI using weighted methods. Dennis Jeffrey et al. [8] viewed the programs written as slices based on their data and control flow. Every slice may represent multiple specifications. A single test case may cover a few of the requirements in a particular slice. Moreover, the output influencing statements of the slice and expressions present in the slice is the area in which attention must be paid. All of these aspects are applied heuristically to achieve prioritization. The workflow of the proposed methodologies is given in Figure 1. Figure 1 shows the various TCP methods used by researchers in RT, and in this paper, we have proposed two methods under the similarity/dominancy category. It is observed that the two clustering methods are developed to tackle two different perspectives. Similarity-based clustering is used in a situation where accuracy is more importance than time. In contrast, dominance-based clustering is used in situations where time is more importance than accuracy. Finally, after clustering, the clusters and the test cases are prioritized using the Weighted Arithmetic Sum Product Assessment (WASPAS) method.

The details of those methods will be discussed in Section 3. The construction of this paper is as follows. Section 2 discusses the previous studies related to TCP techniques. Section 3 provides the proposed methodology where the core contributions are discussed. Further, in Section 4, a numerical example is presented, and Section 5 gives the real case analysis where the strength of the proposed framework is discussed. Finally, in Section 6, concluding remarks are provided along with the future scope of research.

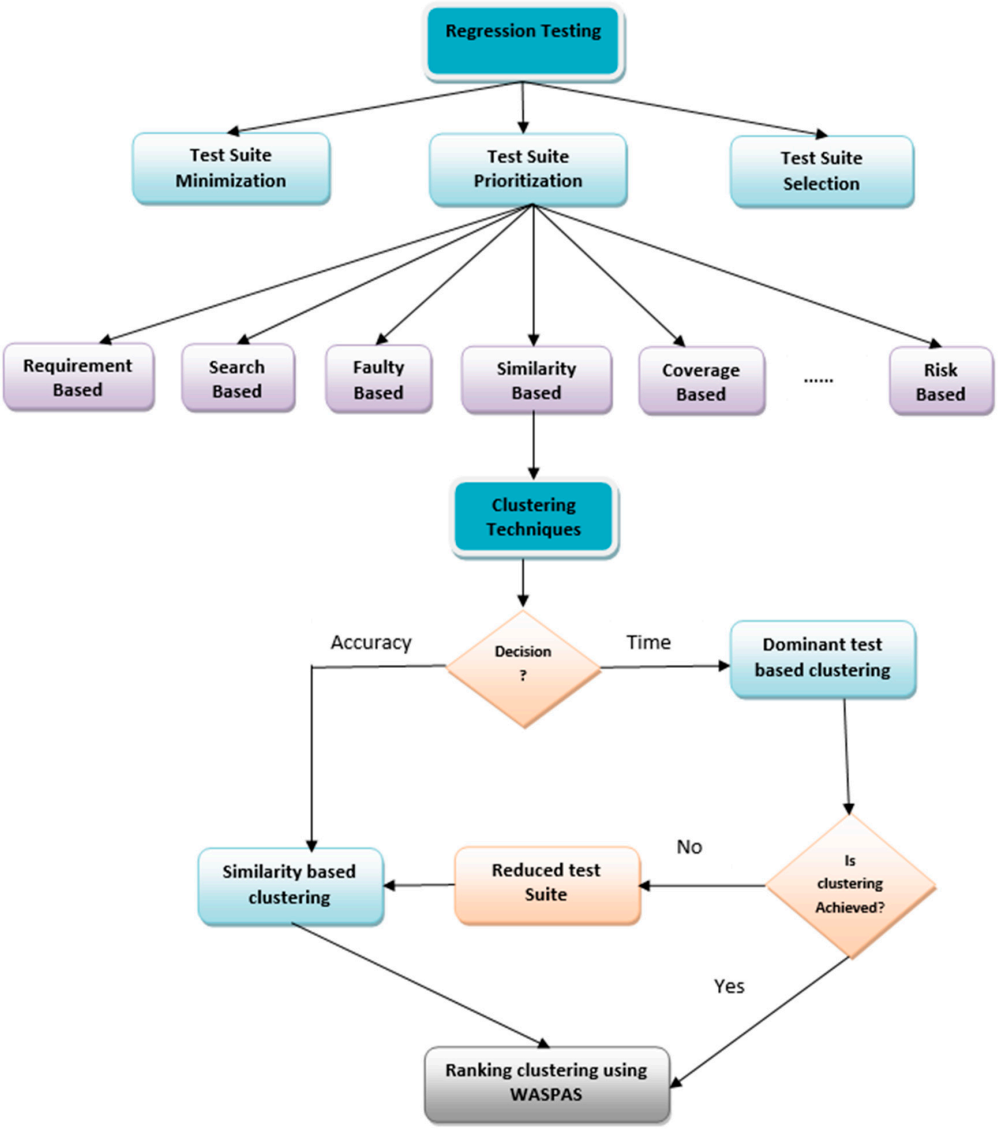

Figure 1. The workflow of the proposed similarity-based test case prioritization (TCP).

2. Related Studies of Test Case Prioritization (TCP)

Prioritizing test cases is a predominant optimization problem, in which the least prioritized test cases may be ignored to save time, while the quality remains intact [14]. Test case prioritization is an attempt to arrange the test cases, in such a way that highly formidable test cases lead the remaining ones based on the factors such as critical bugs' coverage and satisfying the customer's critical requirements. This leads to exemplary permutation of test cases, forming an archetypal test suite. The test cases in such test suites are used to uncover the bugs at the earliest, even though few of the trailing test cases are not exercised. Such prioritization is an optimization effort since it never undermines the power of testing. Several prioritization approaches have been developed for the researchers in the last decade, which is based on coverage, search, fault, risk, requirements, history, similarity, and others. From the survey published by Muhammad Khatibsyarbini et al. [15], more than 70% of the published works deals with STP (Software Testing Prioritization) which is based on search, coverage, faulty, requirement, and history, whereas the other methods occupy the rest of the percentage which is inclusive of similarity-based STP.

Singh [16] presented a detailed review of 65 studies in Regression-based TCP (RTCP) which covers the period of 1997–2011. This study covered the empirical study of eight main approaches

and discussed gaps, metrics, and artifact used. Catal and Misha [17] elaborated another review of 120 studies in RTCP from 2001 to 2011, which discusses the trends in TCP approaches, including evaluation metrics. Yoo and Harman [7] conducted a literature review of RTs over 159 studies over the period of 1977–2009. This study covers four important TCP approaches of RTs namely TCP, TSM & TCS.

Similarly, Refs. [18,19] discussed 19 and 90 studies of literature review in RTCP, respectively. Khatibsyarsini et al. [15] presented 80 studies in RTCP from 1998 to 2016. They selected 80 studies from 707 articles collected from the above said period for discussing various quality assessments. This review covers the types of RT, TCP, tools, and evaluation metrics used. The limitations of the above systematic literature review consist of: (i) there is no empirical evidence for other TCP approaches, (ii) gaps have not been identified regarding the usage of artifacts on TCP approaches, and (iii) reasons behind the importance of the evaluation metrics are not clearly stated.

In this paper, similarity-based approaches for TCP have been concentrated. This will provide priority to the test cases that are the most dissimilar from those already selected. Recently, many researchers have derived similarity-based approaches [12,13,20–22] for TCP. Thomas et al. [23] proposed text analysis method for static TCP. In this paper, the main objective of the similarity-based techniques is to maximize the commonality and minimize the diversity, which is the opposite to researchers that have been obtaining priority that is based on maximizing the diversity and minimizing the similarity [24–27]. In general, similarity-based TCP is mainly divided into two types, which are distribution based and ART (Adaptive Random Test) inspired. TCP for the first case is derived from clustered test cases through dissimilarities between test cases [28,29]. ART is used as an improvised methodology of random testing, and in almost all cases, it outperforms all the cases expect when the size of code is less [30]. ART-inspired is an extension method of ART in TCP, the test cases with higher similarity are only considered instead of considering all of them, and this provides better results than ART [26]. Yves Ledru et al. [31] proposed a TCP model using four similarity-based different string distances, and then it was been proved that the proposed method has better Average Percentage Fault Detection (APFD) than randomly ordered test suites.

Zhang et al. [32] and Jiang et al. [33] have proposed adaptive random testing based prioritization techniques guided by code-coverage. Breno Miranda et al. [34] have extended the concepts of [32,33] using FAST approach, and finally, they proved that the FAST method performs better scaling and timing. ART (Adaptive Random Testing) [2] is a variant of random test generation that tries to spread, as evenly as possible, the test inputs in the input domain. In the present experimentation, the FAST approaches have been compared against both [26,35], which are further described in Section 5. Also, the approach proposed by Fang et al. [36] is based on code coverage information, from which they exploit the execution frequency profiles. Among black-box methods, Ledru et al. [31] propose a similarity-based approach solely considering the strings that express the test cases, i.e., the input data or the JUnit test cases. Also, FAST has been compared against this approach (see Section 5). Noor and Hemmati [37] developed a history-based approach in which, among new or modified test cases, those that are the most similar to failing ones are prioritized. FAST does not currently use historical data.

In software testing, diversity is a useful measure for both selecting and prioritizing test cases, and several similarity-based approaches have been proposed [14,24,26,28,35]. In this, clustering plays the role of prioritizing test cases. ARS approach clusters the test cases according to the number of objects and methods using k-means and k-medoids clustering algorithms. Another method which clusters the test cases according to object and method invocation sequence similarity metric uses the k-medoids clustering algorithm [14].

Model-Based Black-Box Techniques

White-box clustering is also profoundly used in clustering. Program entities such as assignment statements and control structures are analyzed. These are categorized based on the frequency of execution with respect to test cases. Clusters are formed having the same amount of frequencies. Inner elements of the cluster are further clustered using a greedy approach. Ultimately, a representative

of each cluster gives an ordered sequence of test cases. This methodology uses both similarity and distance measures [36].

Event-oriented graph models and the importance of events lead to the clustering of events using unsupervised neural network and fuzzy c-means clustering in finding the frequently occurring event groups, thus providing high ranking to their respective test cases [38].

Various similarity measures over the test cases based on their code coverage have been studied and analyzed. Subsequently, many prioritization algorithms, such as Adaptive Random Testing and Global similarity-based algorithms, are scrutinized using various similarity measures. The apt prioritization algorithms with the best-fit similarity measures are suggested by [1,2]. The factors, namely, customer priority, implementation complexity, fault proneness, and volatility of the requirements are weighed, and their weight factors paved the way for prioritizing the associated test cases. Yuen Ta Kyu et al. [3] addressed the issues in transforming specifications into code, especially when the logical expressions are prone to drift away from the intended meaning. Different forms of the same expression result in multiple results, which may be undesirable. Test cases must be prioritized to obtain compelling test cases that can resolve such an issue.

Requirement based black box prioritization techniques were enhanced by introducing regression test factors such as the impact of fault in requirements, completeness, and traceability. Apart from the severity of faults detected, test efforts on a total and average basis are also used to discern the weight of test cases in attaining the ordering [22]. A new revolutionized approach in the field of prioritization has been introduced by Stephen W. Thomas et al. [23]. Their work focused on text analysis, wherein keywords, literals, comments, and identifiers were extracted from the documents. The test cases which are associated with the identified phrases or topics are considered, and the similarity among them was measured. The most dissimilar test cases were assigned with high ranking.

A novel method of prioritization was to introduce a mutant version of programs raising faults. These are generated through automation rather than hand seeded. Then the rate of fault detection was measured using block coverage, functional coverage, and random coverage and established that mutation-based prioritization was one of the best approaches [24]. Location services-based applications often suffer into the anomaly of rendering undesired locations. Hence, KeZhai et al. [25] introduced a new set of metrics and classified faults in five different categories. Based on these categories, test cases were prioritized and assessed by the harmonic mean of fault detection.

Li et al. [39] analyzed the search algorithms in prioritization which focused on coding facets, giving a stance over choosing the right algorithm from the greedy, meta-heuristic, or genetic and size up the factors determining the efficiency of the search algorithms. Elberzhager et al. [40] did a comprehensive survey of software testing articles and classified them. These classifications help the test manager in determining the required effort in testing, deciding over the stop point of testing and setting up the quality assurance aspects triggered by inspections and reviews. Furthermore, Elbaum et al. [41] presented a method for selection of test cases in a cost-effective manner.

Studies [11,40] presents a systematic review on TCP techniques and analysis its effects on software testing. Yoo and Harman [7], have also done survey and reviews on various test case optimization research methodologies. Do et al. [42] conducted a study on six different prioritization techniques and figured out the performance of the methods by varying time constraints imposed on them. Shrivathsan et al. [43] have done a substantial survey of test case prioritization techniques.

The advantages of the proposed similarity measures are: (i) similarity measures can be evaluated separately in the test cases, as well as the faulty items which are based on the grouping technology; (ii) it is based on the newly developed similarity measuring formula; and (iii) two-way similarity measures provide better classification accuracy than one-way similarity measures provide.

From the above literature survey, it shows that the improvement may take place by proposing an effective similarity-based TCP which will increase the testing accuracy and for that purpose, the present work proposes two similarity-based TCP and, finally, it is proved that the proposed method provides considerably good coverage of test cases.

From the investigation of the literature, the following challenges are encountered:

1. Clustering methods proposed in the literature are single-sided clustering, that is, clustering is done from the perspectives of test cases. This will cause potential information loss, as the interrelationship between test cases and faulty functions is ignored.
2. The similarity measures used in literature do not concentrate on the distribution of the data (input), which affects the convergence process. Moreover, literature studies do not provide methods pertaining to accuracy and time separately, and this is an open challenge to address.
3. Prioritization methods do not consider inter- and intra-clustering, which eventually causes loss of information.
4. Finally, customization on constraints is not dominantly provided in TCP. This would restrict the convergence and cause inaccuracies.

Motivated from these open challenges and to overcome them, the following contributions are made in this paper:

1. Challenge one is resolved by proposing a novel two-way clustering that clusters both test cases and faulty functions for better understanding the interrelationship among them.
2. Challenge two is addressed by proposing a new similarity measure that utilizes the power of exponent measure and provides smoothening of data for effective convergence. Further, the distribution of the data is also taken into consideration, and dominancy measure is proposed along with similarity measure for properly managing accuracy time trade-off.
3. The WASPAS method is extended for inter- and intra-clustering prioritization, which effectively prioritizes test cases.
4. Finally, the programming model is put forward for better customization of parameters to obtain an optimal test case for the set of faulty functions.

3. Proposed Methodology

Practically, in the context of Regression testing for software quality assurance, test case prioritization is a crucial aspect. In this paper, a fuzzy similarity-based test case prioritization model is proposed to improve the performance of regression testing compared to the existing literature. AbdurRahman et al. [44] proposed Call Dependency Graph (CDG) method, which uses the measure based on the similarity, wherein the degree of relationship between two test cases is measured through similarity. Boolean values are used to find the presence and absence of connectivity.

3.1. Proposed Fuzzy-Similarity Test Case Prioritization (TCP) Model (FSTPM)

All requirements are not equally important to the clients. Therefore, a software technique needs to be developed for selecting the requirements which are important to the clients. Hence, specific requirements need more attention from testers. Thus, the requirements clusters are prioritized, and their priority information is used to select test cases from each cluster, obtaining a complete set of reordered test cases. That is, the clusters with higher priority can be visited at an earlier time, and several test cases could be selected from it.

Proposed Fuzzy-Similarity Test Case Prioritization (TCP) Model (FSTPM) is a clustering model, and its solution procedure is based on grouping. Optimal clustering is obtained by using optimal grouping between the test cases and the faulty functions. The solution procedure for FSTPM is given below:

Given the fuzzy linguistic matrix $F = [L_{ij}]$, where the values of L_{ij} are linguistic and for the sake of convenience, the following linguistic levels are identified. The linguistic levels are low (LL), low-high (LH), medium (MM), medium-high (MH), high (HH), and very high (VH). Here, L_{ij} is represented as the fault f_j, which is identified by the test case TC_i. The Gaussian membership function is used to associate the truth value to the linguistic levels.

Modify the linguistic matrix F into fuzzy matrix F1 = [b_{ij}] where b_{ij} is the fuzzified value of the given linguistic levels in F. All the entries of the fuzzy matrix F1 lies within 0 and 1. This value is the degree of relationship between the test cases and the fault.

To convert the given fuzzy matrix F1 into 0–1 adjacency matrix F2 = [a_{ij}], whose elements are defined by:

$$a_{ij} = \begin{cases} 0 \; if \; b_{ij} \leq \vartheta \\ 1 \; if \; b_{ij} > \vartheta \end{cases}, \tag{1}$$

where the value of ϑ is chosen based on the testing-time constraint. If higher the testing-time then lower the ϑ value and lesser testing-time leads to higher ϑ value. The value of ϑ is calculated by:

$$\vartheta = min\left\{\frac{1}{\rho}, \frac{G_c}{N}\right\}, \tag{2}$$

where ρ = maximum number of permissible test cases in a group; G_C = number of permissible grouping and N = number of test-cases (or) a maximum number of faults identified by a test case. The value of ρ must satisfy the condition: $\rho = \frac{N+1}{G_c}$.

After finding the 0–1 adjacency matrix F2, grouping between the test cases is identified. The following procedure is used to find the grouping of similar test cases.

Construct a lower triangular similarity matrix for the given test cases from the 0–1 adjacency matrix F2 by using:

$TC_{sim} = \left\{S_{ij};\ i > j \; and \; \forall i,\ j = 1,\ 2,\ 3,\ \ldots,\ N\right\}$, where N = total number of test cases and

$$S_{ij} = \left(\frac{a\left(\frac{a}{ne^{\left(\frac{n-a}{n+a}\right)}} + d\right)}{\left[a\left(\frac{a}{ne^{\left(\frac{n-a}{n+a}\right)}}\right) + \ b + c + ad\right]}\right). \tag{3}$$

Here, a is the number of faults that exists on both the test cases TC_i and TC_j; b is the number of faults that exists only on TC_i; c is the number of faults that exists only on TC_j and d is the number of faults that do not exist on test cases TC_i and TC_j.

When test cases uncover faulty function, repetition of test cases may occur; if such a scenario exists, then any of the repeated test cases is removed before applying the grouping techniques. Suppose $S_{ij} = 1$, then TC_i and TC_j have the same faulty functions, and either TC_i or TC_j is removed, and these two test cases are tagged into one, namely TC_{ij}. This will reduce the processing time of the problem.

After modifying the similarity matrix, all dominant similarity coefficients are swapped into the first G_c columns (G_c—number of groups), because of the maximum number of clusters is G_c. For this purpose, the similarity matrix is further modified as follows: (i) if larger similarity coefficient value appears between the test cases in the first G_c column of each row, then all the values are retained as such; (ii) if the largest similarity value appears after G_c column, then that column is interchanged to any one of the first G_c columns; and (iii) if the largest similarity value occurs in various places including the first G_c column, then ε quantity is added to the highest similarity appeared in the first G_c column and this quantity lies typically between 0.001 and 0.005. The nature of the problem is not affected by this. A few problems require this modification.

For finding the optimal grouping of the test cases, the following 0–1 optimization problem is constructed:

The optimal clustering problem for grouping test cases which maximizes the sum of similarities S_{ij} is given by:

$$Max \; Z = \sum_{i=1}^{N} \sum_{j=1}^{N} S_{ij} y_{ij} \; for \; i > j, \tag{4}$$

where $y_{ij} = \begin{cases} 1;\ both \; the \; test \; cases \; TC_i \; and \; TC_j are \; mutually \; inclusive \\ 0;\ both \; the \; test \; cases \; TC_i \; and \; TC_j are \; mutually \; exclusive \end{cases}$

Subject to the constraints:

$$\sum_{i=1}^{N} y_{ij} = 1 \; for \; j = 1, \; 2, \; \ldots, \; N \; and \; i \geq j, \tag{5}$$

$$\sum_{\substack{j=1 \\ i=j}}^{N} y_{ij} = \; G_c \; for \; j = 1, \; 2, \; \ldots, \; N, \tag{6}$$

$$\rho y_{jj} + \sum_{i=1}^{N} y_{ij} \geq 0 \; for \; j = 1, \; 2, \; \ldots, \; N \; and \; i > j, \tag{7}$$

$$and \; y_{ij} = 0 \; or \; 1 \; for \; i, \; j = 1, \; 2, \; 3, \; \ldots, \; N. \tag{8}$$

Using similarity values obtained from 4(d) the optimization problem is derived as per Equations (4) through (8) and using MATLAB® software, the optimal grouping between the test cases is obtained. Optimization toolbox is used that adopts linear programming solver with a dual-simplex algorithm. Further, the parameters are set according to the application, and they are discussed below.

After finding the 0–1 adjacency matrix F2, there is the need to find grouping between the faultiness. The following procedure is used to find the grouping of the similarity values between the faulty functions, which are similar to step (4) with minor modifications.

Construct a lower triangular similarity matrix for the given faulty functions from the 0–1 adjacency matrix F2 from:

$\text{FAULT}_{\text{sim}} = \left\{ S_{ij}; \; i > j \; and \; \forall i, \; j = 1, \; 2, \; 3, \; \ldots, \; M \right\}$, where M = total number of faultiness identified by all the test cases and:

$$S_{ij} = \left(\frac{a\left(\frac{a}{ne^{\left(\frac{n-a}{n+a} \right)}} + d \right)}{\left[a\left(\frac{a}{ne^{\left(\frac{n-a}{n+a} \right)}} \right) + \; b + c + ad \right]} \right). \tag{9}$$

Here, a is the number of faulty functions exists on both the faulty functions FAULT_i and FAULT_j; b is the number of faults exists only on FAULT_i c is the number of faults exists only on FAULT_j and d is the number of faults does not exist on faulty functions FAULT_i and FAULT_j.

When faulty function uncovered by test cases, repetition of faulty functions may occur; if such a scenario exists, then any of the faulty function is removed before applying grouping techniques. Suppose $S_{ij} = 1$, then FAULT_i and FAULT_j have the same faulty functions, and either FAULT_i or FAULT_j is removed, and these two test cases are tagged into one, namely FAULT_ij. This will reduce the time complexity of the problem.

For finding the optimal grouping of the test cases, the following 0–1 optimization problem has been constructed:

The optimal clustering problem for grouping test cases, which maximizes the sum of similarities S_{ij} and it is derived as:

$$Max \; Z = \; \sum_{i=1}^{M} \sum_{j=1}^{M} S_{ij} x_{ij} \; for \; i > j, \tag{10}$$

where $x_{ij} = \begin{cases} 1; \; both \; the \; faulty \; items \; are \; mutually \; inclusive \\ 0; \; both \; the \; faulty \; items \; are \; mutually \; exclusive \end{cases}$ and M = number of faulty functions identified by the test cases.

Subject to the constraints:

$$\sum_{j=1}^{M} x_{ij} = 1 \; for \; i = 1, \; 2, \; \ldots, \; M \; and \; i \geq j, \tag{11}$$

$$\sum_{j=1}^{M} x_{jj} = \; G_c \; for \; j = 1, \; 2, \; \ldots, \; M, \tag{12}$$

$$\rho x_{jj} + \sum_{i=1}^{M} x_{ij} \geq 0 \; for \; j = 1, \; 2, \; \ldots, \; M \; and \; i > j, \tag{13}$$

$$and \; x_{ij} = 0 \; or \; 1 \; for \; i, \; j = 1, \; 2, \; 3, \; \ldots, \; M. \tag{14}$$

Using similarity values, the optimization problem is derived as per Equations (10) through (14) and solved by using MATLAB® software (MathWorks, Natick, MA, USA) with linear programming solver and dual-simplex algorithm. As mentioned earlier, parameter values are application-specific, and they are discussed below. Finally, the optimal grouping between the faultiness is obtained

Using the optimal grouping between the test cases and faultiness, the incidence matrix F_2 is rearranged as per the grouping got from steps (4) and (5) and then finally, optimal clustering is obtained.

For ease of understanding Algorithm 1 is presented, which clearly describes the working of fuzzy-similarity test case prioritization model.

Algorithm 1: Pseudo code for fuzzy-similarity test case prioritization model (FSTPM)

Input: Given F = [L_{ij}], for i = 1, 2, … , n and j = 1, 2, … , m are the linguistic relationship matrix between the set of n-test cases and m-faulty items.

Output: F1 = [a_{ij}], for i = 1, 2, … , n and j = 1, 2, … , m, with clustering partition between the set of test cases and faultiness.

1. Using Gaussian membership function, convert the given F-linguistic relationship matrix into F1 = [b_{ij}] for i = 1, 2, … , n and j = 1, 2, … , m as fuzzy matrix

2. To compute fuzzy 0–1 matrix F2 = [a_{ij}] for i = 1, 2, … , n and j = 1, 2, … , m from F_1 by using the relation

$$a_{ij} \leftarrow \begin{cases} 0 \; if \; b_{ij} \leq \vartheta \\ 1 \; if \; b_{ij} > \vartheta \end{cases} \quad \text{where } \vartheta \leftarrow min\left\{\frac{1}{\rho}, \; \frac{G_c}{N}\right\}$$

3. for i = 1 to n do

{

for j = 1 to n do

{

i. to compute a, b, c and d values between the i^{th} and j^{th} row

ii. to compute the similarity between the test cases by using the relation $S_{ij} \leftarrow \left(\frac{a\left(\frac{a}{ne^{\left(\frac{n-a}{n+a}\right)}} + d\right)}{\left[a\left(\frac{a}{ne^{\left(\frac{n-a}{n+a}\right)}}\right) + \; b+c+ad\right]} \right)$

}

}

4. using the similarity index S_{ij} as obtained from step-3, to solve the following 0–1 programming:

$$Max \; Z = \sum_{i=1}^{N} \sum_{j=1}^{N} S_{ij} y_{ij} \; for \; i > j$$

$$\text{where } y_{ij} = \begin{cases} 1; \; both \; the \; test \; cases \; TC_i \; and \; TC_j are \; mutually \; inclusive \\ 0; \; both \; the \; test \; cases \; TC_i \; and \; TC_j are \; mutually \; exclusive \end{cases}$$

subject to the constraints

$$\sum_{i=1}^{N} y_{ij} = 1 \; for \; j = 1, \; 2, \; \ldots, \; N \; \; and \; \; i \geq j$$

$$\sum_{\substack{j=1 \\ i=j}}^{N} y_{ij} = \; G_c \; for \; j = 1, \; 2, \; \ldots, \; N$$

$$\rho y_{jj} + \sum_{i=1}^{N} y_{ij} \geq 0 \; for \; j = 1, \; 2, \; \ldots, \; N \; and \; \; i > j \; and \; y_{ij} = 0 \; or \; 1 \; for \; i, \; j = 1, \; 2, \; 3, \; \ldots, \; N$$

5. Apply steps 3 and 4 for faultiness and based on the $y_{ij,}$ values, group/cluster the faultiness

6. Based on the group of test cases and the faultiness, rearrange the rows and columns of F_2, then we got the clustering. This will be the given input of an inter and intra ranking of clusters

3.2. Dominancy Test Based Clustering for Test Case Prioritization (DTTCP)

The limitation of FSTPM is the random assumption of many terms, namely (i). The number of clustering, (ii). The maximum number of test cases accommodated in a group, (iii) Fixation of the threshold value, and so on. Again, if the number of groups varies, then the clustering accuracy also varies. In the above example, if the number of groups is fixed as 3, then the clustering accuracy is above 60%, whereas for the same problem if we fix the number of groups as 2, then the clustering accuracy exceeds 85%. The limitation of the FSTPM is that they cannot handle clusters of varying numbers between test cases and faculty functions. To overcome these limitations, a new clustering method based on the dominancy test for test case prioritization (DTTCP) has been proposed.

The advantages of the DTTCP are: (1) based on the nature of the problem, the number of clusters can be calculated systematically and (2) if the number of grouping between the test cases and faulty items mismatch, then the resultant matrix is used to find the clustering, which requires very less computational time when compared to its counterpart. Once the dominancy test is applied, the size of the problem will reduce drastically. The following is the procedure for DTTCP:

Start with the matrix F2 derived from Section 3.1. T1, T2 . . . , Tn is the set of test cases (given in rows) and F1, F2, . . . , Fm is the set of faulty items (given in columns).

Compare the dominance between the rows. If the row vector Ta contains the row vector Tb then row Tb is removed and re-designate row Ta by Tab. If Ta ⊇ Tb ⊇ Tc then remove rows Tb and Tc and re-designate row Ta by Tabc. Similarly, if Ta ⊇ Tb . . . ⊇ Tk then removes rows Tb, Tc, . . . Tk and re-designate row Ta by Tab . . . k.

For the ease of understanding, Algorithm 2 is presented, which describes the working of DTTCP.

Algorithm 2: Pseudo code for dominancy test for test case prioritization (DTTCP)

Input: Given F = [L_{ij}], for i = 1, 2, . . . , n and j = 1, 2, . . . , m are the linguistic relationship matrix between the set of n-test cases and m-faulty items.
Output: F2 = [a_{ij}], for i = 1, 2, . . . , n and j = 1, 2, . . . , m, with clustering partition between the set of test cases and faultiness (or) Recommended to move to FSTPM with reduced F2 fuzzy 0–1 matrix

1. Using Gaussian membership function, convert the given F-linguistic relationship matrix into F1 = [b_{ij}] for i = 1, 2, . . . , n and j = 1, 2, . . . , m as fuzzy matrix
2. To compute fuzzy 0–1 matrix F2 = [a_{ij}] for i = 1, 2, . . . , n and j = 1, 2, . . . , m from F_1 by using the relation
$$a_{ij} \leftarrow \begin{cases} 0 \; if \; b_{ij} \leq \vartheta \\ 1 \; if \; b_{ij} > \vartheta \end{cases} \quad \text{where } \vartheta \leftarrow min\left\{\frac{1}{\rho}, \frac{G_r}{N}\right\}$$
3. To apply the following steps to find the cluster or the reduced F2 matrix:
   a. If the row vector Ta contains (⊇) the row vector Tb, then row Tb is removed and re-designate row Ta by Tab.
   b. If no more rows are contains with other rows, then stop the process
   c. Apply steps 3a and 3b for columns also, and do the same process
4. If the size of the matrix is a square matrix then stop the process. The resulting matrix is partitioned accordingly. Otherwise, the size of the matrix of F2 is drastically reduced and then performs steps 3 to 6 as FSTPM.

In the previous example, $TC_1 \supseteq TC_8$ and $TC_6 \supseteq TC_5$, then remove rows TC_5 and TC_8, then the resultant F2 matrix is modified as follows:

$$F2 = Testcases\begin{Bmatrix} & \begin{matrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9\end{matrix} \\ \begin{matrix}18\\2\\3\\4\\65\\7\\9\end{matrix} & \begin{bmatrix}1&0&0&1&1&1&1&0&0\\0&1&1&1&1&1&0&0&0\\0&1&1&1&0&0&1&1&0\\0&1&1&0&0&0&1&1&1\\1&0&0&0&1&1&1&0&1\\1&1&1&1&0&0&0&1&0\\1&1&1&1&0&1&0&0&0\end{bmatrix}\end{Bmatrix}.$$

Similarly, perform step-2 for columns also. Here, $F_2 \supseteq F_3 \supseteq F_8$, then remove columns three & eight and re-designate column F_2 by F_{238}. Likewise, $F_6 \supseteq F_5$, then remove columns five and re-designate column F_6 by F_{65}. Again, $F_7 \supseteq F_9$, then remove column 9 and re-designate column F_7 by F_{79}. Hence, the resultant F_2 is given by:

$$F2 = Testcases\begin{Bmatrix} & \begin{matrix}1 & 238 & 4 & 65 & 79\end{matrix} \\ \begin{matrix}18\\2\\3\\4\\65\\7\\9\end{matrix} & \begin{bmatrix}1&0&0&1&1\\0&1&1&1&1\\0&1&1&1&0\\0&1&1&0&0\\1&0&0&0&1\\1&1&1&1&0\\1&1&1&1&0\end{bmatrix}\end{Bmatrix}.$$

Apply steps 2 and 3 until the number of groups of rows and columns is equal in number.

From step 2 we get, $F_{18} \supseteq F_{65}$, then remove row 65 and re-designate column F_{18} by F_{1865}. Again $TC_3 \supseteq TC_4$ and $TC_9 \supseteq TC_7$, then remove rows TC_4 and TC_7, then the resultant F2 matrix is modified as follows:

$$F2 = Testcases\begin{Bmatrix} & \begin{matrix}1 & 238 & 4 & 65 & 79\end{matrix} \\ \begin{matrix}1865\\2\\34\\97\end{matrix} & \begin{bmatrix}1&0&0&1&1\\0&1&1&1&1\\0&1&1&1&0\\0&1&1&0&0\end{bmatrix}\end{Bmatrix}.$$

Similarly, it is applied to columns and rows, and finally, the matrix F2 is reduced as:

$$F2 = Testcases\begin{Bmatrix} & \begin{matrix}651 & 238 & 784\end{matrix} \\ \begin{matrix}1865\\972\\34\end{matrix} & \begin{bmatrix}1&0&0\\0&1&1\\0&1&1\end{bmatrix}\end{Bmatrix}.$$

Expanding the above matrix, the required clustering is obtained, which is the output of the final clustering for dominancy test. The APFD measure of the above two methods, along with non-prioritization, is given in the following Figure 2a–c.

Clearly, from the above Figure 2a–c, it is concluded that cluster based on the similarity coefficient performs better than the other methods.

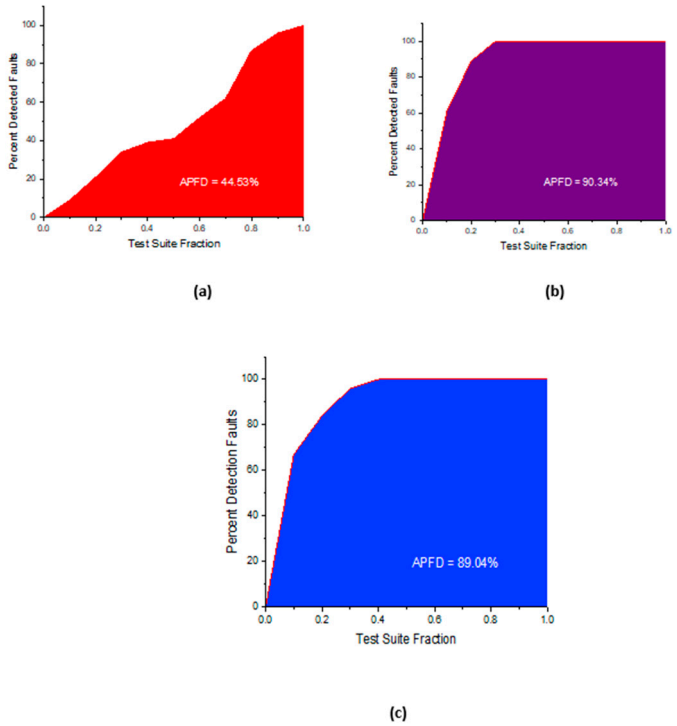

Figure 2. Area plot: (**a**) Average Percentage Fault Detection (APFD) measure for non-prioritization; (**b**) APFD measure for cluster-based similarity coefficient prioritization; and (**c**) APFD measure for cluster-based on prioritization.

3.3. Discussion

As per the client's requirement, not all the requirements are equally important, and some requirements are used frequently when the software is deployed. Hence, those frequently addressed requirements are more important during the clustering process. For this purpose, we have taken input as linguistic, and it provides the degree of relationship between the test cases and the faulty functions. The uniqueness of the proposed clustering methodology is that the identical properties among the test-cases and the faulty functions are identified separately and then combined. The proposed methodology performs better than the other clustering algorithms suggested in the recent state-of-the-art techniques by Gokce et al. [38], Mohammed et al. [44], Chaurasia et al. [45], and Mohammed and Do [46]. Grouping accuracy is based on: (i) the number of clusters to select and (ii) the number of test cases accommodated in each cluster.

Three different measures are used for measuring the performance of the proposed method. The first one is an accuracy, which is based on how many 1's are accommodated in the clusters. For optimal ρ, G_c, and ϑ, the proposed method achieved nearly 71% accuracy, whereas the other state-of-the-art techniques achieved a maximum of 62% with regards to the final clustering. In the above example, in cluster-1, the detected faults are f_2, f_3, and f_4, and the undetected fault is f_5. Similarly, in cluster-2, detected faults are f_7 and f_9; and in this case, no fault is being undetected. In cluster-3, detected faults are f_1 and f_6, and fault f_8 is undetected. The advantage of this clustering methodology is that applying grouping technology for test cases separately and faults separately, so that the number of undetected faults is very less, compared to the other clustering algorithms.

Depending on the clusters set, the clustering accuracy varies. In the same example, if the number of cluster set is assumed to be two, then the clustering accuracy will be 87.5%. Hence, the assumption of cluster sets is critical while applying clustering techniques for prioritization.

The performance analysis, before and after clustering is ascertained through Average Percentage Fault Detection (APFD) measure. APFD achieves a better result of 51.34% through the clustering method than other methods (without clustering) with an APFD score of 48.76%. Hence, the clustering

method of prioritization gives a relatively better result. The following session discusses the variously estimated clustering, and the ranking among the clustering, which is calculated through the Weighted Arithmetic Sum and Product Assessment (WASPAS) method.

3.4. An Inter and Intra Ranking of Clusters

Consider the procedure for ranking the clusters and then selecting desirable order in which test cases must be utilized for better performance.

Step 1: Obtain k clusters each having an evaluation matrix of order m by n where m denotes the number of test cases, and n denotes the number of criteria.

Step 2: The values in these matrices are linguistic. These are converted into fuzzy values by using Gaussian membership function.

Step 3: Initially, the dominant cluster is estimated by using the weighted arithmetic method given in Equation (15). The weight of each test case is considered to be equal, and this helps the procedure to pay equal attention to each test case.

$$\zeta_i = \sum_{j=1}^{n} w_j \mu_{ij}, \tag{15}$$

where n is the number of criteria, w_j is the weight of the criteria with $0 \leq w_j \leq 1$, and $\sum_j w_j = 1$ and μ_{ij} is the fuzzy value.

Step 4: Using step 3, the weighted arithmetic value of each test case pertaining to a particular cluster is obtained. The average is calculated for each cluster, and these values are normalized to obtain the weight of the cluster. They are given by Equations (16) and (17):

$$c_p = \frac{\sum_{i=1}^{m} \zeta_i}{m}, \tag{16}$$

$$\varphi_p = \frac{c_p}{\sum_{p=1}^{k} c_p}, \tag{17}$$

where p is the number of clusters taken for the study.

Step 5: From step 4, the weight of each cluster is obtained, and using these values dominant cluster can be determined.

Step 6: The matrix of order m by n is chosen from the dominant cluster, and the test cases are ranked using the WASPAS method. The formulations are given by Equations (18)–(20):

$$Q_1 = \sum_{j=1}^{n} w_j \mu_{ij}, \tag{18}$$

$$Q_2 = \prod_{j=1}^{n} \left(\mu_{ij}\right)^{w_j}, \tag{19}$$

$$Q_3 = \lambda Q_1 + (1-\lambda) Q_2, \tag{20}$$

where Q_1 is the weighted sum method (WSM), Q_2 is the weighted product method (WPM), Q_3 is the final rank value w_j is the weight of the j^{th} criterion with $0 \leq w_j \leq 1$ and $\sum_j w_j = 1$ and λ is the strategy value with $0 \leq \lambda \leq 1$.

Step 7: The Q_3 value is obtained for each test case and the test case, which has the highest value is a highly preferred test case and so on.

4. Numerical Example

4.1. Illustration of Clustering of Test Cases

The proposed algorithm is illustrated with nine test cases, which uncovers nine faults and the scenario of uncovering is as shown below:

TC_1 = { f1, f4, f5, f6, f7}, TC_2 = { f2, f3, f4, f5, f6}, TC_3 = {f1, f2, f3, f4, f7, f8}, TC_4 = { f2, f3, f7, f8, f9},
TC_5 = { f1, f4, f6, f7, f9}, TC_6 = { f1, f3, f5, f6, f7, f9 }, TC_7 = { f1, f2, f3, f4, f8}, TC_8 = { f1, f6, f8},
TC_9 = { f1, f2, f3, f4, f6}.

From the available information, the fuzzy-linguistic matrix F is constructed as shown below:

$$F = Testcases \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \overset{\overset{Faulty\ Functions}{\overset{\overset{1}{\downarrow}\ \overset{2}{\downarrow}\ \overset{3}{\downarrow}\ \overset{4}{\downarrow}\ \overset{5}{\downarrow}\ \overset{6}{\downarrow}\ \overset{7}{\downarrow}\ \overset{8}{\downarrow}\ \overset{9}{\downarrow}}{\begin{bmatrix} VH & -- & -- & MM & VH & MH & MM & -- & -- \\ -- & HH & MM & HH & VH & MM & -- & -- & -- \\ LL & HH & VH & MH & -- & -- & MM & VH & -- \\ -- & MM & MM & -- & -- & -- & MH & HH & VH \\ MH & -- & -- & LL & -- & VH & MM & -- & HH \\ MH & -- & LH & -- & HH & VH & HH & -- & VH \\ MM & VH & MH & MH & -- & -- & -- & HH & -- \\ HH & -- & -- & -- & -- & VH & -- & LL & -- \\ MM & HH & VH & HH & -- & MM & -- & -- & -- \end{bmatrix}}}{}.$$

By applying appropriate value to their concerned Gaussian Membership Function, the following fuzzy-matrix F1 is derived. The input membership function for the given problem is defined in Figure 3.

$$F1 = Testcases \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \overset{\overset{Faulty\ Functions}{\overset{\overset{1}{\downarrow}\ \overset{2}{\downarrow}\ \overset{3}{\downarrow}\ \overset{4}{\downarrow}\ \overset{5}{\downarrow}\ \overset{6}{\downarrow}\ \overset{7}{\downarrow}\ \overset{8}{\downarrow}\ \overset{9}{\downarrow}}{\begin{bmatrix} 0.89 & 0.00 & 0.00 & 0.34 & 0.92 & 0.71 & 0.34 & 0.00 & 0.00 \\ 0.00 & 0.81 & 0.89 & 0.74 & 0.88 & 0.36 & 0.00 & 0.00 & 0.00 \\ 0.12 & 0.77 & 0.89 & 0.67 & 0.00 & 0.00 & 0.34 & 0.92 & 0.00 \\ 0.00 & 0.44 & 0.53 & 0.00 & 0.00 & 0.00 & 0.75 & 0.73 & 0.85 \\ 0.70 & 0.00 & 0.00 & 0.13 & 0.00 & 0.88 & 0.53 & 0.00 & 0.69 \\ 0.56 & 0.00 & 0.12 & 0.00 & 0.74 & 0.85 & 0.72 & 0.00 & 0.89 \\ 0.34 & 0.96 & 0.66 & 0.74 & 0.00 & 0.00 & 0.00 & 0.68 & 0.00 \\ 0.76 & 0.00 & 0.00 & 0.00 & 0.00 & 0.89 & 0.00 & 0.14 & 0.00 \\ 0.36 & 0.67 & 0.95 & 0.78 & 0.00 & 0.35 & 0.00 & 0.00 & 0.00 \end{bmatrix}}}{}$$

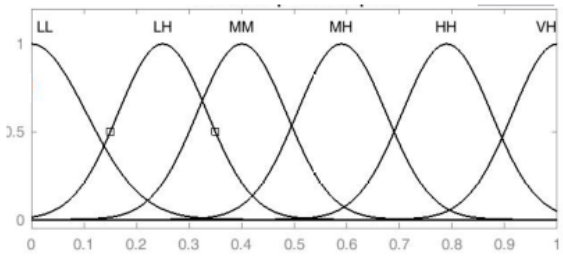

Figure 3. Input Gaussian Membership function.

In the sample problem, the number of test-cases (N = 9) and the number of faults identified by all the test-cases (M = 9) are equal. Let us assume that the number of groups G_c = 3. Therefore, $\vartheta = min\left\{\frac{1}{\rho}, \frac{G_c}{N}\right\} = min\left\{\frac{1}{4}, \frac{3}{9}\right\} = 0.25$. Hence, modify the fuzzy matrix F1 into 0–1 incidence matrix, namely F2 using Equation (2) as:

$$Faulty\ Functions$$

$$\mathrm{F2} = Testcases \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \begin{matrix} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{matrix}.$$

From F2, construct a lower or upper triangular similarity index matrix for test cases $TC_{sim} = \left\{S_{ij};\ i > j\ and\ \forall i,\ j = 1,\ 2,\ 3,\ \ldots,\ 9\right\}$ using Equation (3) as:

$$Test\ Cases\ vs\ Test\ Cases$$

$$S_{ij} = Testcases \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \begin{matrix} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{matrix} \\ \begin{bmatrix} 0.000 & - & - & - & - & - & - & - & - \\ 0.609 & 0.000 & - & - & - & - & - & - & - \\ 0.130 & 0.458 & 0.000 & - & - & - & - & - & - \\ 0.000 & 0.254 & 0.945 & 0.000 & - & - & - & - & - \\ 0.751 & 0.130 & 0.250 & 0.445 & 0.000 & - & - & - & - \\ 0.857 & 0.255 & 0.130 & 0.255 & 0.945 & 0.000 & - & - & - \\ 0.255 & 0.609 & 0.762 & 0.609 & 0.130 & 0.000 & 0.000 & - & - \\ 0.754 & 0.419 & 0.092 & 0.033 & 0.858 & 0.754 & 0.419 & 0.000 & - \\ 0.609 & 0.863 & 0.271 & 0.255 & 0.459 & 0.255 & 0.863 & 0.754 & 0.000 \end{bmatrix} \end{matrix}.$$

Substitute $TC_{sim}(S_{ij})$ value in the optimization function given in (4) and expand the constraints given in Equations (5)–(8). Then by using MATLAB® software (optimization toolbox, with solver as linear programming) solve the 0–1 programming, to obtain the following result. (Since G_c is 3, by applying Equation (2), the maximum number of test cases accommodated in a single group will be $\rho = 4$ and the remaining indices not mentioned below (varying between i = 1–9 and j = 1–9) are zero. Hence, relating the above, three clusters among test cases have been obtained, namely Cluster-1 = {1, 5, 6, 8}, Cluster-2 = {2, 9, 7} and Cluster-3 = {3, 4}.

$$y66 = y77 = y44 = y16 = y29 = y34 = y56 = y85 = y97 = 1$$

In a similar mechanism, grouping technology concepts are applied to faulty functions using step (5) in the solution procedure from F2, construct a lower or upper triangular similarity index matrix for faulty functions $FAULT_{sim} = \left\{S_{ij};\ i > j\ and\ \forall i,\ j = 1,\ 2,\ 3,\ \ldots,\ 9\right\}$ using Equation (3) as:

$$Faulty\ Functions\ vs\ Faulty\ Functions$$

$$S_{ij} = Testcases \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \begin{matrix} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{matrix} \\ \begin{bmatrix} 0.000 & - & - & - & - & - & - & - & - \\ 0.000 & 0.000 & - & - & - & - & - & - & - \\ 0.000 & 1.00 & 0.000 & - & - & - & - & - & - \\ 0.609 & 0.762 & 0.762 & 0.000 & - & - & - & - & - \\ 0.459 & 0.271 & 0.271 & 0.754 & 0.000 & - & - & - & - \\ 0.837 & 0.000 & 0.000 & 0.609 & 0.762 & 0.000 & - & - & - \\ 0.379 & 0.255 & 0.255 & 0.130 & 0.623 & 0.379 & 0.000 & - & - \\ 0.130 & 0.869 & 0.869 & 0.419 & 0.092 & 0.000 & 0.623 & 0.000 & - \\ 0.459 & 0.271 & 0.271 & 0.033 & 0.566 & 0.459 & 0.869 & 0.566 & 0.000 \end{bmatrix} \end{matrix}.$$

Substitute $FAULT_{sim}$ (S_{ij}) value in the optimization function given in (9) and expanding the constraints given in Equations (10)–(13), and then using MATLAB® software to solve the 0–1 programming, the following results were obtained (Assumed G_c is 3, then using (2), the maximum number of test cases accommodated in a single group will be $\vartheta = 4$).

$$x66 = x22 = x77 = x15 = x34 = x42 = x56 = x87 = x98 = 1$$

The indices not mentioned in Equation (15) (varying for i = 1–9, and j = 1–9) are zero. Hence, relating the above, three clusters have been obtained for faulty functions, namely Cluster-1 = {1, 5, 6}, Cluster-2 = {2, 3, 4} and Cluster-3 = {7, 8, 9}. Here also, if we set G_c as 2, then test case numbered 8 is either added to the first cluster or the second cluster. Rearrange the matrix F2 as per the equality constraints given above, the resultant matrix is a clustering matrix and is given in (16).

$$FinalClustering = Testcases \begin{matrix} \\ 1 \\ 5 \\ 6 \\ 8 \\ 2 \\ 9 \\ 7 \\ 3 \\ 4 \end{matrix} \begin{matrix} \overset{Faulty\ Functions}{\overbrace{}} \\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9 \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

4.2. An illustrative Example for Inter and Intra Test Case Prioritization

Form matrices of order $m \times n$ where m represents the test cases and n represents the faults. Based on the clustering process, these matrices are formed of order 4×3, 3×3, and 2×2, respectively.

$$\text{Matrix1} = \begin{bmatrix} 0.89 & 0.92 & 0.71 \\ 0.7 & 0 & 0.88 \\ 0.56 & 0.74 & 0.85 \\ 0.76 & 0 & 0.89 \end{bmatrix} \text{Matrix 2} = \begin{bmatrix} 0.81 & 0.89 & 0.74 \\ 0.67 & 0.95 & 0.78 \\ 0.96 & 0.66 & 0.74 \end{bmatrix} \text{Matrix 3} = \begin{bmatrix} 0.34 & 0.92 & 0 \\ 0.75 & 0.73 & 0.83 \end{bmatrix}$$

Calculate the WASPAS parameters using Equations (20)–(22) and they are shown in Table 1.

Table 1. Rank value analysis for cluster-1.

Alternative	Sum	Product	Rank
A1	0.84	0.834	0.83
A2	0.526	0	0.26
A3	0.716	0.706	0.711
A4	0.55	0	0.275

From Table 1, the intra ranking is given by $TC_1 > TC_5 > TC_6 > TC_8$ for cluster-1, $TC_2 > TC_9 > TC_7$ for cluster 2 and $TC_3 > TC_4$ for cluster 3, respectively. The inter ranking is given by C3 > C1 > C2 with values 3.59, 2.63, and 2.4, respectively. A similar idea is adopted for Tables 2 and 3 as well.

Table 2. Value analysis for cluster-2.

Alternative	Sum	Product	Rank
A1	0.813	0.81	0.81
A2	0.8	0.79	0.7959
A3	0.786	0.77	0.781

Table 3. Rank value analysis for cluster-3.

Alternative	Sum	Product	Rank
A1	0.42	0	0.21
A2	0.776	0.7749	0.7758

5. A Real Case Study Analysis

The proposed technique is tested with the programs developed by researchers and is extracted from the local software company, which is collaborated with some group of students in the university, and it is available in (https://sir.unl.edu). The programs chosen for testing purpose are illustrated with descriptions in Table 4. The chosen programs belong to the category of concurrency. These programs inherently have the quality of multithreaded activities requiring arbitration.

Table 4. Software projects tested.

Objects	Language	Req.	Versions	Size (KLoC)	Classes	# Faulty Versions	Fault Types	Bug Type	Bug Description
Pool 1	Java	19	2	3.62	28	36	Real	Race	Thread execution order and execution speed problem
Pool 2	Java	59	2	18.66	88	114	Real	Deadlock	Resource allocation contention problem
Pool 3	Java	123	3	38.09	124	136	Real	Deadlock	Resource allocation contention problem
Pool 4	Java	84	4	20.43	94	112	Real	Deadlock	Resource allocation contention problem

In Table 4, each of the objects is associated with the programming languages which are used to create an object, the number of lines of code in the program, number of classes used in an object, number of faulty versions, the number of versions, number of requirements and its types, and the types of bugs based on the given requirements. The test case studied in this work is a functional test case associated with requirements. The performance analysis of the proposed methods with the following ten popular prioritization techniques is given below.

Total statement coverage prioritization (P1): Test cases are ordered with respect to the number of statements covered by them. The greater the number of statements covered, the higher the prioritization. When more than one test case covers the same number of statements, then they are assigned priority randomly, being next to each other.

Additional statement coverage prioritization (P2): Test cases are ordered based on their higher number of statements coverage. Then, this ordering is adjusted based on which test case covers the uncovered statements.

Total branch coverage prioritization (P3): It is very similar to the total statement coverage prioritization, except that the coverage is for branches, rather than for the statements.

Additional branch coverage prioritization (P4): It is very similar to the additional statement coverage prioritization, except that the coverage is for branches, rather than for the statements.

Total function coverage prioritization (P5): It is very similar to the total statement coverage prioritization, except that the coverage is for functions, rather than for the statements.

Additional function coverage prioritization (P6): It is very similar to the additional statement coverage prioritization, except that the coverage is for functions, rather than for the statements.

Total fault exposing potential prioritization (P7): This technique considers the identification of a fault in two dimensions. The first dimension is that the test covers a faulty statement. Another dimension is that the fault of a statement failing a test case means that the test case does not expose the fault. Based on both dimensions, a test case will be awarded value. This awarded value is computed by considering three factors. 1) Statement execution probability. 2) Probability in a change of state (computed through statement mutation) when mutants are exposed and or are not exposed 3) The subsequent state changes due to the preceding factor. For a test case, this awarded value is found for each of the

statements covered by it. Moreover, the summation of these values will be the Total Fault Exposing Potential value of that test case. The test cases of the suite are reordered according to the descending order of total Fault Exposing Potential (FEP) value, yielding prioritization.

Total FEP-function level prioritization (P8): This technique is analogous to total FEP statement-level prioritization, and a function is considered for coverage in lieu of statements.

Additional fault exposing potential prioritization (P9): It is an extension of Total fault exposing potential prioritization. The Fault Exposing Potential (FEP) value of a test case with respect to a statement is adjusted, by lowering the award values of all other test cases covering the same statement. The rationale behind this is the correctness of the statement is very first ascertained by the test case covering it first. This value is estimated by means of a factor called "confidence", ranging between 0 and 1. When a test case exercises a statement and not finding any fault, then the confidence value of that statement increases. The confidence of a statement is influenced by the FEP of the test case too. Additional confidence of a statement is obtained by finding the difference between prior confidence value and posterior confidence value of that statement. If a test case "TCi" covers 'm' statements, then the additional confidence values of all 'm' statements covered by TCi is summed up to figure out Additional fault exposing the potential of the test case TCi. Finally, the test cases are arranged in descending order of Additional fault exposing potential values.

Additional FEP-function level prioritization (P10): This technique is analogous to additional FEP statement-level prioritization. And, a function is considered for coverage in lieu of statements.

From the literature, it is highly evident that the performances of the prioritization techniques are improving the performance of software testing in terms of Average Percentage of Faults Detected (APFD) measure and Code Complexity Measure (CCM) measures. Hence, these two metrics are used in software testing to validate the test case prioritization. The details of the performance metrics are given below.

Various runs are made for the programs mentioned in Table 4 as per coverage techniques stated above, and their efficiency is analyzed in terms of Average Percentage of Faults Detected (APFD) measure. The APFD measure is given in Equation (21).

$$\text{APFD} = 1 - \frac{\text{FCP}_\text{I} + \text{FCP}_2 + \ldots + \text{FCP}_\text{tf}}{\text{ttc} \times \text{tf}} + \frac{1}{2 \times \text{ttc}} \tag{21}$$

In Equation (21), 'FCP' signifies the fault covered in its very first position among the test cases, 'ttc' refers a total number of test cases considered, and 'tf' refers a total number of faults. This is one of the performance measurement metrics, and during the execution of the test suite, the APFD measure is obtained, and this value lies typically between 0 and 100. A higher APFD measure signifies higher prioritization and a lesser APFD value implies lesser prioritization. Depending on LoC, the number of clusters is taken. In this study, the performance is tested with three different cluster sizes, namely 3, 6, and 9. Then the performance of the test case prioritization is estimated with different cluster sizes and is given in Table 2.

The second performance measure, the Code Complexity Measure (CCM) depends on three pieces of information, lines of code (LOC), Nested block depth (NBD), and McCabe cyclomatic complexity (MCC). The LOC is the total number of lines in a class, NBD is the number of nested statements used in a class and MCC is the number of linearly independent paths to reach the goal of a method. CMM is obtained from the relation:

$$CMM = \frac{1}{3}\left(\frac{LOC}{max(LOC)} + \frac{NBD}{max(NBD)} + \frac{MCC}{max(MCC)}\right). \tag{22}$$

Figure 4a–d summarizes the performance analysis of the proposed methods and others with respect to the percentage of APFD scores after having applied prioritization techniques for the programs extracted from sources given above with respect to the generated random test cases. Here, N1 and N2 represents the two proposed methods namely clustering based prioritization with similarity coefficient and clustering based prioritization with dominancy. The performance analyses of the proposed methods with 3, 6, and 9 clusters are tested and Table 2 shows the percentage of APFD score below.

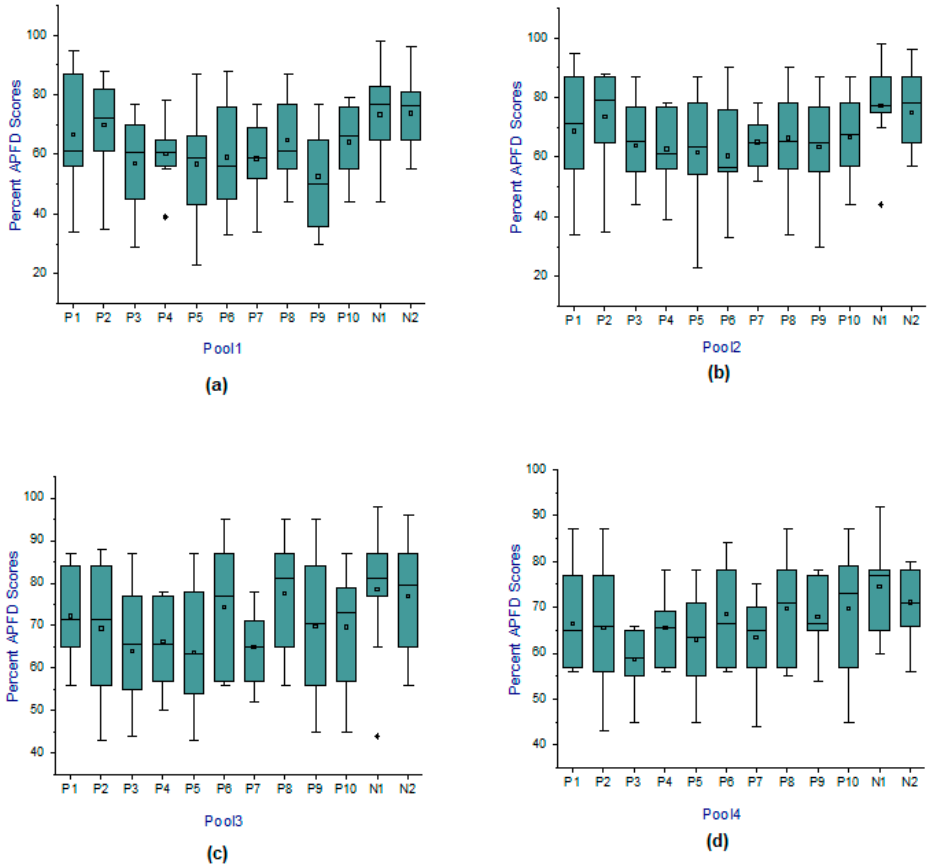

Figure 4. Box plot: (**a**) Performance analysis of the proposed methods and others—Pool 1; (**b**) Performance analysis of the proposed methods and others—Pool 2; (**c**) Performance analysis of the proposed methods and others—Pool 3; (**d**) Performance analysis of the proposed methods and others—Pool 4.

Table 5 shows that the optimal groupings for Pool 1 to Pool 4 are 3, 6, 6, and 6, respectively. From the box plots, Figure 4a–d indicates that the proposed method performs better than the other state-of-the-art techniques available in the literature. By applying various rounds of prioritization techniques, the standard deviation of N1 and N2 are better than many of the other techniques and comparing mean and median values with other methods are always better. In Pool 3, one data of N1 is moving to the outlier and this may easily be corrected by using any of the removals of outlier methods.

Table 5. APFD score values in different number of clusters.

Proposed Methods	Pool 1 No. of Clusters			Pool 2 No. of Clusters			Pool 3 No. of Clusters			Pool 4 No. of Clusters		
	3	6	9	3	6	9	3	6	9	3	6	9
N1	73.21	72.56	69.23	73.48	78.36	75.86	74.16	77.12	73.57	73.31	74.67	71.78
N2	71.97	71.64	70.05	74.51	76.43	74.09	74.82	76.07	75.66	66.85	70.96	68.32

6. Conclusions

This paper proposes and demonstrates two fuzzy based clustering techniques, FSTPM and DTCTP, which are based on the newly developed similarity coefficient and the dominancy test, respectively. These proposed methods are implemented, tested, and analyzed using an empirical study that accesses requirement based fuzzy clustering techniques in test case prioritization. The results show that the requirements-based-fuzzy-clustering-approach which incorporates traditional code analysis information can improve the effectiveness of test case prioritization techniques, but the results

vary based on the cluster size. Finally, it is concluded that, by grouping test cases associated with a similarity or dominancy related set of requirements, regression testing processes can be managed more effectively.

Author Contributions: A.D.S., R.K. and V.S. prepared the initial plan of the research, collected data, pre-processed the data, and prepared a prototype of the research model. K.S.R. and S.K. fine-tuned the idea and gave valuable suggestion throughout the research work. A.D.S. and R.K. implemented the fine-tuned idea and prepared the manuscript. J.J. and P.Z. provided good insights on research paper writing and verified out result.

Funding: Authors are extremely thankful to the funding agencies for the financial aid from University Grants Commission (UGC), India, (F./2015-17/RGNF-2015-17-TAM-83), Department of Science and Technology (DST), India, (SR/FST/ETI-349/2013), Council of Scientific & Industrial Research (CSIR), India (09/1095/(0026)18-EMR-I), and partially within the framework of the program of the Minister of Science and Higher Education under the name "Regional Excellence Initiative" in the years 2019-2022, project number 001/RID/2018/19, the amount of financing PLN 10,684,000.00.

Acknowledgments: The authors thank the editor and the anonymous reviewers for their valuable comments, which promoted the quality of the paper to a better extent.

Conflicts of Interest: Authors share no conflict of interest.

Appendix A

The symbols used in the equations along with their meaning are provided in Table A1.

Table A1. Symbols along with explanations.

Symbol	Description
F	Fuzzy linguistic matrix
F1	Fuzzy matrix of F
F2	Fuzzy 0–1 matrix
Lij	Linguistic relationship between the test case i and faulty item j
LL	Low
LH	Low-high
MM	Medium
MH	Medium-high
HH	High
TC_i	Test case i
f_j	Faulty item j
a_{ij}	1 if the faultiness j is in test case i; otherwise 0
a	Faulty item is occurring in both the test cases i and j, while finding the similarity coefficient between i^{th} and j^{th} test cases
b	Faulty item is occurring in the test case i but not j
c	Faulty item is occurring in the test case j but not i
d	Faulty item is not occurring in both of the test cases i and j
N	Total number of test cases
M	Total number of faultiness available in the test cases
G_c	Number of permissible groups
ρ	Maximum number of permissible test cases
υ	Threshold value
S_{ij}	Similarity between test cases i and j / faultiness j and j
y_{ij}	Decision variables used during test cases grouping
x_{ij}	Decision variables used during faultiness grouping
T_a	Test case a
T_i	i^{th} faults
W_i	Weight criteria i
μ_{ij}	Fuzzy value between i^{th} and j^{th} criterion
ζ_i	Weighted arithmetic value
ς_p	Average weighted arithmetic
φ_p	Normalized weighted arithmetic
Q_1	Weighted sum method
Q_2	Weighted product method
Q_3	Final rank
λ	Strategy value

References

1. Wang, R.; Jiang, S.; Chen, D.; Zhang, Y. Empirical Study of the effects of different similarity measures on test case prioritization. *Math. Probl. Eng.* **2016**. [CrossRef]
2. Srikanth, H.; Banerjee, S. Improving test efficiency through system test prioritization. *J. Syst. Softw.* **2012**, *85*, 1176–1187. [CrossRef]
3. Yu, Y.T.; Lau, M.F. Fault-based test suite prioritization for specification-based testing. *Inf. Softw. Technol.* **2012**, *54*, 179–202. [CrossRef]
4. Rothermel, G.; Untch, R.H.; Chu, C.; Harrold, M.J. Prioritizing Test Cases for Regression Testing. *IEEE Trans. Softw. Eng.* **2001**, *27*. [CrossRef]
5. Huang, Y.-C.; Peng, K.-L.; Huang, C.-Y. A history-based cost-cognizant test case prioritization technique in regression testing. *J. Syst. Softw.* **2012**, *85*, 626–637. [CrossRef]
6. Chittimalli, P.K.; Harrold, M.J. Recomputing coverage information to assist regression testing. *IEEE Trans. Softw. Eng.* **2009**, *35*, 452–469. [CrossRef]
7. Yoo, S.; Harman, M. Regression testing minimisation, selection and prioritisation: A survey. *Test Verif. Reliab.* **2007**, *22*, 1–7.
8. Jeffrey, D.; Gupta, N. Improving fault detection capability by selectively retaining test cases during test suite reduction. *IEEE Trans. Softw. Eng.* **2007**, *33*, 108–123. [CrossRef]
9. Elbaum, S.; Kallakuri, P.; Malishevsky, A.G.; Rothermel, G.; Kanduri, S. Understanding the effects of changes on the cost-effectiveness of regression testingtechniques. *J. Softw. Test. Verif. Reliab.* **2003**, *12*, 65–83. [CrossRef]
10. Rothermel, G.; Untch, R.H.; Chu, C.C.; Harrold, M.J. Test case prioritization: An empirical study. In Proceedings of the IEEE International Conference on Software Maintenance, (ICSM 99), Oxford, UK, 30 August–3 September 1999; pp. 179–188.
11. Elbaum, S.; Malishevsky, A.G.; Rothermel, G. Test case prioritization: A family of empirical studies. *IEEE Trans. Softw. Eng.* **2002**, *28*, 159–182. [CrossRef]
12. Khalilian, A.; Azgomi, M.A.; Fazlalizadeh, Y. An improved method for test case prioritization by incorporating historical test case data. *Sci. Comput. Program.* **2012**, *78*, 93–116. [CrossRef]
13. Huang, C.-Y.; Chang, J.-R.; Chang, Y.H. Design and analysis of GUI test-case prioritization using weight-based methods. *J. Syst. Softw.* **2010**, *83*, 646–659. [CrossRef]
14. Chen, J.; Zhu, L.; Chen, T.; Towey, D.; Kuo, F.-C.; Huang, R.; Guo, Y. Test Case Prioritization for Object-Oriented Software: An adaptive random sequence approach based on clustering. *J. Syst. Softw.* **2018**, *135*, 107–125. [CrossRef]
15. Muhammad, K.; Isa, M.A.; Jawawi, D.N.A.; Tumeng, R. Test case prioritization approaches in regression testing: A systematic literature review. *Inf. Softw. Technol.* **2018**, *93*, 74–93.
16. Singh, Y. Systematic literature review on regression test prioritization techniques. *Informatica* **2012**, *36*, 379–408.
17. Catal, C.; Mishra, D. Test case prioritization: A systematic mapping study. *Softw. Qual. J.* **2012**, *21*, 445–478. [CrossRef]
18. Kumar, A.; Singh, K. A Literature Survey on Test Case Prioritization. *Compusoft* **2014**, *3*, 793. [CrossRef]
19. Kiran, P.; Chandraprakash, K. A literature survey on TCP-test case prioritization using the RT-regression techniques. *Glob. J. Res. Eng.* **2015**. Available online: https://engineeringresearch.org/index.php/GJRE/article/view/1312 (accessed on 11 October 2019).
20. Jeffrey, D.; Gupta, N. Experiments with test case prioritization using relevant slices. *J. Syst. Softw. Sci. Direct.* **2007**, 196–221. [CrossRef]
21. Lijun, M.; Chan, W.K.; Tse, T.H.; Robert, G. Merkel. XML-manipulating test case prioritization for XML-manipulating services. *J. Syst. Softw.* **2011**, *84*, 603–619. [CrossRef]
22. Krishnamoorthi, R.; Sahaaya Arul Mary, S.A. Factor oriented requirement coverage based system test case prioritization of new and regression test cases. *Inf. Softw. Technol.* **2009**, *51*, 799–808. [CrossRef]
23. Thomas, S.W.; Hemmati, H.; Hassan, A.E.; Blostein, D. Static test case prioritization using topic models. *Empir. Softw. Eng.* **2012**. [CrossRef]
24. Do, H.; Rothermel, G. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Trans. Softw. Eng.* **2006**, *32*. [CrossRef]

25. Zhai, K.; Bo, J.; Chan, W.K. Prioritizing Test Cases for Regression Testing of Location-Based Services: Metrics, Techniques, and Case Study. *IEEE Trans. Serv. Comput.* **2014**, *7*, 54–67. [CrossRef]
26. Haidry, S.; Miller, T. Using Dependency Structures for Prioritization of Functional Test Suites. *IEEE Trans. Softw. Eng.* **2013**, *39*, 258–275. [CrossRef]
27. Srikanth, H.; Hettiarachchi, C.; Do, H. Requirements based test prioritization using risk factors: An Industrial Study. *Inf. Softw. Technol.* **2016**, *69*, 71–83. [CrossRef]
28. Hettiarachchi, C.; Do, H.; Choi, B. Risk-based test case prioritization using a fuzzy expert system. *Inf. Softw. Technol.* **2016**, *69*, 1–15. [CrossRef]
29. Huang, R.; Chen, J.; Towey, D.; Chan, A.T.S.; Lu, Y. Aggregate-strength interaction test suite prioritization. *J. Syst. Softw.* **2015**, *99*, 36–51. [CrossRef]
30. Jiang, B.; Chan, W.K. Input-based adaptive randomized test case prioritization: A local beams approach. *J. Syst. Softw.* **2015**, *105*, 91–106. [CrossRef]
31. Ledru, Y.; Petrenko, A.; Boroday, S.; Mandran, N. Prioritizing test cases with string distances. *Autom. Softw. Eng.* **2012**, *19*, 65–95. [CrossRef]
32. Zhang, C.; Chen, Z.; Zhao, Z.; Yan, S.; Zhang, J.; Xu, B. An improved regression test selection technique by clustering execution profiles. In Proceedings of the 10th International Conference on Quality Software (QSIC'10), Washington, DC, USA, 14–15 July 2010; pp. 171–179. [CrossRef]
33. Jiang, B.; Zhang, Z.; Chan, W.; Tse, T. Adaptive random test case prioritization. In Proceedings of the 24th International Conference on Automated Software Engineering (ASE'09), Auckland, New Zealand, 16–20 November 2009; pp. 233–244. [CrossRef]
34. Breno, M.; Bertolino, A. Scope-aided test prioritization, selection and minimization for software reuse. *J. Syst. Softw.* **2017**, *131*, 528–549.
35. Kim, J.M.; Porter, A. A History-based Test Prioritization Technique for Regression Testing in Resource Constrained Environments. In Proceedings of the 24th International Conference on Software Engineering, Orlando, FL, USA, 19–25 May 2002; pp. 119–129. [CrossRef]
36. Fang, C.; Chen, Z.; Wu, K.; Zhao, Z. Similarity based test case prioritization using ordered sequence of program entities. *Softw. Qual. J.* **2014**, *22*, 335–361. [CrossRef]
37. Noor, T.B.; Hemmati, H. A similarity-based approach for test case prioritization using historical failure data. In Proceedings of the 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), Gaithersbury, MD, USA, 2–5 November 2015.
38. Gokce, N.; Belli, F.; Eminli, M.; Dincer, B.T. Model-based test case prioritization using cluster analysis: A soft-computing approach. *Turk. J. Electr. Eng. Comput. Sci.* **2015**, *23*, 623–640. [CrossRef]
39. Li, Z.; Harman, M.; Hierons, R.M. Search algorithms for regression test case prioritization. *IEEE Trans. Softw. Eng.* **2007**, *33*, 225–237. [CrossRef]
40. Elberzhager, F.; Rosbach, A.; Munch, J.; Eschbach, R. Reducing test effort: A systematic mapping study on existing approaches. *Inf. Softw. Technol.* **2012**, *54*, 1092–1106. [CrossRef]
41. Elbaum, S.; Rothermel, G.; Kanduri, S.; Malishevsky, A.G. Selecting a Cost-Effective Test Case Prioritization Technique. *Softw. Qual. J.* **2004**, *12*, 185–210. [CrossRef]
42. Do, H.; Mirarab, S.; Tahvildari, L.; Rothermel, G. The effects of Time Constraints on Test Case Prioritization: A Series of Controlled Experiments. *IEEE Trans. Softw. Eng.* **2010**, *36*, 593–617. [CrossRef]
43. Shrivathsan, A.D.; Ravichandran, K.S. Meliorate test efficiency: A survey. *World Appl. Sci. J.* **2014**, 133–139. [CrossRef]
44. Mohammed, A.R.; Mohammed, A.H.; Mohammed, S.S. Prioritizing Dissimilar Test Cases in Regression Testing using Historical Failure Data. *Int. J. Comput. Appl.* **2018**, *180*. [CrossRef]
45. Chaurasia, G.; Agarwal, S. A Hybrid Approach of Clustering and Time-aware based Novel Test Case Prioritization Technique. *Int. J. Database Theory Appl.* **2016**, *9*, 23–44. [CrossRef]
46. Mohammed, J.A.; Do, H. Test Case Prioritization using Requirements-Based Clustering. In Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification, and Validation, Luxembourg, 18–22 March 2013; pp. 312–321. [CrossRef]