Performance Evaluation of Clustering Algorithms for Enhancing Test Case Prioritization in Regression Testing

S.Prabath Puviskar
Department of Computing & Information Systems
Sabaragamuwa University of Sri Lanka
Sri Lanka
prabath096@gmail.com

W.V.S.K. Wasalthilaka
Department of Software Engineering
Sabaragamuwa University of Sri Lanka
Sri Lanka
subodhi@foc.sab.ac.lk

Prof. B.T.G.S Kumara
Department of Computing & Information Systems
Sabaragamuwa University of Sri Lanka
Sri Lanka
kumara@appsc.sab.ac.lk

***Abstract*—Regression testing plays a crucial role in maintain software quality as applications evolving and getting more complex. Selection of a regression testing technique significantly influences overall software quality. In initial stages, this involves picking relevant test cases and remove unnecessary redundancies during minimization phase. Finally, prioritization of test cases phases most important ones is executed first which continue process iteratively or whole testing carried out which effect time and cost intensively. Test Case Prioritization method effective way to address the issue. In this study, we proposed approach to enhance Test Case Prioritization by integrating fault based methods and time of execution analysis. The study also evaluates the efficiency in proposed methodology through, employing APFD metric. The study aims to improve the precision and effectiveness of test case prioritization methods, advancing overall quality and reliability in dynamic software development environments. Notably, AHC outperforms in comparisons, showcasing its efficacy in improving outcomes.**

Keywords—Test Case Prioritization, K-Mean, Expectation Maximization, Agglomerative Hierarchical Clustering, Spectral Clustering

I. INTRODUCTION

Software testing [1] is an essential process of the Software Development Life Cycle (SDLC) that ensures the quality, reliability, and functionality of the software product. Software testing verifies where the software product goes with software requirements with defects free, manually or automation testing manner. Testing an essential part of the SDLC process that could identify defects early in a process, which would improve satisfaction and trust in the software system. Testing the software system could detect vulnerabilities previously which would save in time and cost factors. Especially testing software systems could determine whether the software systems could break or exceed the threshold in different parameters [2]. As the development progress and debugging undertaken the newly errors are emergence is common occurrences. To address this issue Regression Testing becomes crucial phase in testing process.

Regression testing [3] is a type of functional testing that is performed to ensure that a modification to code-base doesn't impact the product's current functioning. Regression testing makes sure product functions properly with the new functionality, or modifications to current features [4]. The primary goal of regression testing is to uncover regressions or unexpected side effects that may arise from the introduction of new codes, bug patches, or system upgrades, thereby contributing to dependability and stability of application systems throughout their development phases. A variety of factors including the level of complexity of the software system and accessibility of assets, regression testing can be conducted manually or through automated testing methods. Automated regression testing is frequently used because it enables more rapid and effective execution of test cases, particularly when significant amount of test cases must be executed. Nevertheless, re-executing test cases in regression testing can be tedious, time-consuming, and resource rigorous. Manually executing all the test cases could be hard, and address the time and cost factors. Also, could be difficult to determine critical test cases should be executed first [5]. Additionally, to initiate smooth testing phase in regression testing the environment should be stable which able to meet the deadlines timely. For an effective testing process, regression testing can be performed in various ways.

A. Retest All
B. Selection Test
C. Test Case Prioritization (TCP)

In the context of this study, our primary significance is directed towards TCP technique. This specific focus is driven by, when comparing to other testing technique, is characterized by a lower resource intensity. TCP is a technique that used in software testing to identify the order in which test cases should be executed primarily. TCP is a possible approach for optimizing the order in which test cases are conducted, to enhance fault detection rates and for resource allocation effectiveness. It allows for better detection of flaws and fast feedback to the development teams. TCP provides an ideal method for executing test cases, enabling more effective and targeted testing activities. This can result in less testing duration and better resource utilization. Primarily various TCP methods on behalf of regression testing such as coverage, requirement, risk, search, fault, history-based, and other kinds of methods can be addressed [6].

However, the effective implementation of TCP can be further elevate integration with clustering techniques, which significantly enhance the stems from the need to manage the complexity of modern software systems. Clustering algorithms [7] are a sort of technique utilized in data mining and machine learning to group similar equivalent data points. As software applications grow in scale and intricacy, clustering provides systematic approach to organize test cases and reducing the overall number of comparisons required. This method allows for more streamlined

2024 4th International Conference on Advanced Research in Computing (ICARC) | 979-8-3503-8486-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICARC61713.2024.10499739

979-8-3503-8486-4/24/$31.00 ©2024 IEEE

prioritization process, reduce the resource utilization nature of regression testing. Researchers have widely acknowledged clustering as valuable technique to optimize test case prioritization, also to enhance fault detection rates and improve overall testing efficiency. For this study we have used K-Means, Agglomerative Hierarchical Clustering (AHC), Expectation–Maximization (EM), and Spectral clustering algorithms for evaluations.

II. RELATED WORKS

In the last few years, number of studies have been conducted in test case prioritization using various clustering algorithms including K-Means, Fuzzy K-medoid, Fuzzy C-means and etc.

N.Gokilavani et al. [8] (2020) investigated on Fuzzy K-medoid clustering algorithm in combination with the Enhanced Adaptive Random Sequence (EARS) algorithm. Proposed method builds on an improved cluster-based fuzzy algorithm, emphasizing the understanding of K-medoid. In the study discovered notable limitation in the methodology's lack of validation through real-world scenarios. The proposed technique demonstrated effectiveness particularly in prioritizing test cases based on fault detection and utilizing with the EARS algorithm for organized and listed sequences. Thus, absence of practical validation raised concerns about the broader applicability and effectiveness of the proposed technique in real-world software testing contexts. The study proposed future works to extend in increased performance metrics measurement for high dimensional inputs.

Paruchuri Ramya and her team [9] (2018) utilizes with k-means clustering algorithm and prioritization test cases based on code complexity. The study highlighted that incorporating requirements in the testing phase, revealing substantial advantages in fault detection. The study emphasized the absence of source code information diminishes the effectiveness of the technique. The study was limited with the absence of validation, challenging that to gauge the practicality and effectiveness of the proposed methodology in real-world scenarios. Additionally, research unable to comparatively analysis with existing test case prioritization techniques, produce uncertainties about the method's relative effectiveness and practical advantages. Despite these promising outcomes, the identified limitations underscore the need for future research to validate the proposed technique in diverse scenarios and compare its effectiveness with existing methodologies.

Geetanjali Chaurasia *et al.* [10] (2015) investigates on Fuzzy C-means clustering algorithm with Intra-Cluster test case prioritization technique. The study introduced novel clustering-based prioritization approach, resulting performance comparison to basic coverage-based techniques in JUnit testing. Notably, the proposed method considered execution time as crucial attribute, which leading to an enhanced Average Percentage of Fault Detected (APFD) value. However, limitations in the study surface as it lacks of information about the data size utilized. Additionally, the absence of real-world case studies raising concerns about the scalability and real-world relevance. The future work proposed to extend the technique beyond Java applications, incorporating languages like C and C++. Furthermore, exploring prioritization with languages such as TestNG and Test Specification Language (TSL), and considered more sophisticated features for clustering, are identified as crucial for further study's.

M. J. Arafeen and H. Do, [11] (2013) investigated on K-Means clustering algorithm with Intra-Cluster prioritization technique, that mainly focused on source code information for test case prioritization. The study concluded that requirements-based clustering approach, combined with traditional code analysis, would enhance the effectiveness of test case prioritization techniques, out coming various results based on the cluster sizes. Notably, the study acknowledged key limitations that influencing the outcomes based on the selection of code metric and cluster numbers. This sensitivity to parameter choices implies that the method's effectiveness may vary depending on these decisions.

Sarika Chaudhary *et al.* [12] (2020) adopts multiple clustering algorithms, including K-Means, Density Based Spatial Clustering of Applications with Noise (DBSCAN), Density-Based K-mean Clustering (DBK mean), and Agglomerative Hierarchical Clustering (AHC), utilizing code coverage information for test case prioritization technique. The study evaluations show that DBK-means demonstrated superior performance across all scenarios when compared to other methods. Study's limitations included that small program sample size and a focus only on hard clustering methods, as indicated by the conclusion. These limitations raised concerns about the generalizability of the findings to real-world scenarios and the potential oversight of other clustering techniques, particularly fuzzy clustering. The study proposed that incorporating fuzzy clustering in future research could contribute to enhanced classification accuracy and runtime efficiency.

In above mentioned research studies, various clustering algorithms have been employed for test case prioritization and several prioritization techniques methods have been associated with specific clustering approaches, offering diverse strategies for optimizing testing processes. But there were crucial issues in, such as the absence of real-world scenario validation, limited comparative analysis, and uncertainties related to data size and parameters. Therefore we are proposing an approach to address the need for comprehensive evaluation of clustering-based prioritization methods, providing valuable insights into their applicability and performance in the context of software testing.

III. METHODOLOGY

To perform the study data undergoes into several clustering algorithms and prioritize the test cases as shown in the Fig. 1. Finally, the novel prioritized technique effectiveness measured by APFD metric. There are five major phases in the methodology such as Data Collection, Data Preprocessing, Applying Clustering Algorithms, Implementation of Environment and Workflow Overview.

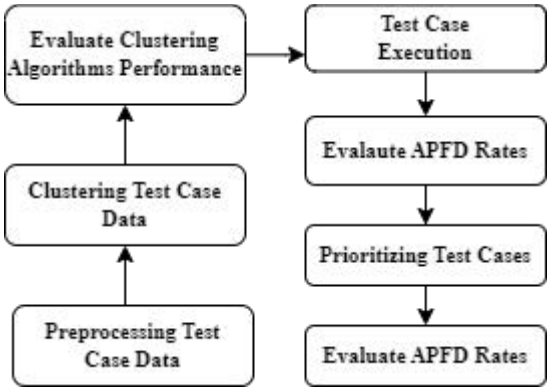

Fig. 1. Proposed Research Methodology

A. Data Collection

Data collected from trustworthy company project along 700 test cases containing important header row attributes like ID, Scenario, Type of Testing, Test Case, Test Steps, Expected Results, Actual Results, Pass/Fail and Execution Time (ET) columns. These datasets stand as valuable resource for in-depth analysis, offers insights to test case characteristics, execution timelines, and other essential vital information for efficient quality assurance and testing procedures.

B. Data Prepossessing

In preparation the collected test case data for analysis, a systematic approach to data preprocessing is important for ensuring data accuracy and reliability in subsequent evaluations. The following steps outline an effective data preprocessing workflow:

1) **Handling missing Values**: Identify and address any missing values in the test case data. Finding missing values may involve depending on the nature of missing data, considering the removal of corresponding rows or columns.

2) **Handling Duplicates**: Identify and remove any duplicate rows to maintain data integrity and prevent redundancy in subsequent analyses.

3) **Handling word cases and special characters:** Convert all text data, such as test case scenarios, test case to lowercase to ensure uniformity and avoid redundancy in the analysis. Additionally, special characters and symbols that may not contribute meaningfully to the analysis are removed.

4) **Stop Words**: Identify common stop words frequent words often carry minimal semantic meaning (e.g., "and," "the," "is") and removed with scikit-learn library. This may reduce noise in the datasets and focuses on more meaningful terms.

5) **Lemmatization:** Applied lemmatization to reduce words to their base or root form. This helps word standardize and unify variation, improving the accuracy of subsequent analyses by considering words in their root state.

6) **Handling Outliers:** Handles outliers by identifying and employing appropriate techniques, such as statistical methods or machine learning algorithms. It ensures the robustness of the datasets against outliers that important for maintaining the integrity of subsequent analyses.

C. Applying Clustering Algorithms

The following clustering algorithms were used to prioritization of test cases.

1) ***K-Means:*** K-Means is a partition-based clustering algorithm partitions datasets into a predetermined number of clusters (k) based on similarity of the data points. The data points iteratively assign to nearest cluster centroid and recalculates the centroids until convergence. We utilized the algorithm to group test cases with comparable characteristics, leveraging the inherent structure within the dataset for effective test case prioritization. The clustering results were significant in enhancing our understanding of the relationships among test cases and guiding the prioritization process.

2) ***Agglomerative Hierarchical Clustering (AHC):*** AHC is a hierarchical clustering algorithm that begins with each data points as separate cluster and iteratively merge clusters based on their similarity which process continues until all data points belong to single cluster. AHC provides a hierarchy of clusters, allowing to choose the desired level of granularity. We utilized the 'ward' linkage criterion to measure the distance between clusters. AHC allowed to capture hierarchical relationships among test cases, providing a structured view of the data. The resulting clusters were then analyzed for their impact on prioritization techniques, and their implications for clustering efficiency.

3) ***Expectation-Maximization (EM):*** EM is a probabilistic clustering algorithm under model based clustering category commonly used for data with mixed distributions or when dealing with missing values. EM follows iterative process of assigning probabilities to data points belonging to different clusters, updating cluster parameters, and repeating until convergence. EM is well-suited for modeling complex data structures, such as Gaussian Mixture Models (GMMs) [13], and often used in machine learning contexts. In this study, EM was instrumental in determining underlying patterns and distributions within the dataset. By assigning probabilities to test cases belonging to different clusters, EM contributed to a probabilistic view of the data and enriching the prioritization process.

4) ***Spectral Clustering:*** Spectral clustering is a graph-based clustering algorithm that views data points as nodes in a graph and clusters them based on the graph's

eigenvalues and eigenvectors. It utilizes spectral properties to identify linked components in the data points. Spectral clustering was utilized to group test cases based on the spectral decomposition of the affinity matrix. We employed the nearest neighbors' affinity measurements to construct the adjacency matrix, capturing the relationships between test cases. It allowed to uncover intricate patterns and structures in the data, especially useful when dealing with non-linear separations.

D. *Implementation Environment*

For an efficient and accessible implementation process we selected Google Colab as the primary environment for coding and executing algorithms. Integrated, seamlessly with Python and the platform provided versatile environment for the computational requirements.This section explores the details of the implementation process, highlighting the essential features that rendered in Google Colab, paired with Python's scikit-learn libraries.

1) ***Duplicates:*** Handling duplicate entries is primal step in enhancing the quality the dataset., we leverage libraries with Pandas in Python. Pandas provides robust functionality for identifying and removing duplicate rows, streamlining the process and maintaining data integrity.

2) ***Word Cases and Special Characters:*** Uniformity in word phrases and the elimination of special characters contribute to cleaner and more standardized textual data and eliminate using string manipulation functions in Python, along with regular expressions for specific characters' removal.

3) ***Stop Words****:* Removal of common stop words much important on more meaningful terms in the dataset. Libraries like NLTK (Natural Language Toolkit) offers extensive stop word lists, facilitating their efficient removal from the text collection during the preprocessing.

4) ***Lemmatization:*** Lemmatization, a process of reducing words to their base or root form, which implemented with Python's NLTK libraries that are vital in providing lemmatization capabilities, ensuring that words are transformed to their base forms for better analysis.

5) ***Handling Outliers:*** Handling outliers is vital for maintaining integrity of the dataset. Scikit-learn in Python offer powerful algorithms like Isolation Forest and Local Outlier Factor (LOF) for detecting and handling outliers through machine learning methods, which provide reliable datasets.

6) ***Clustering:*** Clustering is a significant step in our analysis, grouping similar data points together. We employed algorithms with K-Means, AHC, EM, and Spectral Clustering, that have distinct advantages in uncovering patterns within the datasets.

7) ***Calculate PCA Score:*** Principal Component Analysis (PCA) [14] quantifies the variance in datasets by transforming into set of linearly uncorrelated variables called principal components. The PCA score is calculated by multiplying the standardized data matrix by the matrix of principal components which formulas denotes in (1), (2), (3), (4), (5) respectively.

In this study, we utilized Pass/Fail values and Execution Time (ET) values. The dataset undergo with Z-score normalization for uniform scaling, followed by the extraction of principal components through eigen analysis. The top two components were selected to transform the standardized data, resulting in the final PCA values. The derived composite PCA scores played a pivotal role in guiding subsequent analyses, elevating comprehension, and facilitating effective modeling in alignment with the study's objectives.

a) Standardize the Data (Z-score normalization)

$$X_{Standardized} = \frac{X - \mu}{\sigma} \quad (1)$$

X is the original data matrix, μ is the mean vector, and σ is the standard deviation vector.

b) Compute the Covariance Matrix:

$$\begin{bmatrix} Var(x_1) & \cdots & Cov(x_n, x_1) \\ \vdots & \ddots & \vdots \\ Cov(x_n, x_1) & \cdots & Var(x_n) \end{bmatrix} \quad (2)$$

c) Calculate Eigenvalues and Eigenvectors

$$C\upsilon = \lambda\upsilon \quad (3)$$

C is the covariance matrix, υ is the eigenvector, and λ is the eigenvalue.

d) Sort Eigenvalues and Select Top k Components.

$$P = [\upsilon_1 \upsilon_2 \ldots \upsilon_k] \quad (4)$$

e) Transform the Data

$$Z = X_{standardized} P$$
$$PC_i = Z_i = X_{standardized} \cdot \upsilon_i \quad (5)$$

8) ***Calculate the APFD rate:*** APFD evaluates the effectiveness of test case prioritization method which

is assess by summing the positional differences between the actual faulty positions and the positions at which the faults are detected and normalized by the total number of faults and test cases. The APFD rates [15] evaluates using by (6) respectively.

$$APFD = 1 - \left(\frac{T_{F1} + T_{F2} + \ldots + T_{Fm}}{n \times m}\right) + \frac{1}{2n} \quad (6)$$

$T_{F1}, T_{F2}, \ldots, T_{Fm}$ represent the positions at which the faults are detected., n is the total number of faults, and m is the total number of test cases. The higher APFD rate, the more effective the prioritization method.

E. *Workflow Overview*

This section provides an insight into the holistic workflow governing the study, encompassing key phases from data preprocessing to the evaluation of prioritization technique.

1) ***Prepossessing the data:*** Before delving into analysis, the data undergoes numerous preprocessing, handling issues such as duplicates, word cases, special characters, eliminating stop words, lemmatization and outliers. This initial stage sets the basis for subsequent robust analysis.

2) ***Clustering the test case:*** Employing with diverse clustering algorithms, including K-Means, AHC, EM), and Spectral, this step groups test cases based on similarities, revealing underlying patterns in the dataset.

3) ***Evaluate Clustering Algorithms Performance:*** Following the clustering phase, we carefully assess the performance of each algorithm, revealing meaningful clusters. The evaluation employed with following metrics, offering a comprehensive analysis of the clustering effectiveness.

   a) Silhouette Score: Silhouette score is a clustering performance metric that assesses how well defined and distinct the clusters are in a given configuration which higher scores representing well-defined clustering. It quantifies the cohesion in clusters and separate between the clusters.

   b) Calinski-Harabasz Index: Calinski-Harabasz index evaluates the ratio of between-cluster variance to within-cluster variance, determining configurations with high inter-cluster differences and low intra-cluster variations. Higher index signifies well-defined and distinct clusters.

4) ***Test Execution:*** The clustered test cases undergo execution phase, involves running test cases on the testing environmental system to identify potential faults and assess the system's behavior.

5) ***Evaluate Pre-APFD rates:*** Prior to prioritization, the APFD measurements are computed, offering a baseline understanding of the fault detection rates at the stage.

6) ***Prioritize test case:*** Leveraging prioritization techniques, test cases are reordered based on predefined criteria with PCA method, optimizing the order of execution to enhance fault detection efficiencies.

7) ***Evaluate Post-APFD rates:*** APFD rates are re-calculated, allowing comparative analysis against with pre-prioritization phase. This provides insights into the effectiveness of the applied prioritization methodology.

IV. Results and Discussion

For this study, selection employed with three clusters was made to ensure more balanced representation, as two clusters,demonstrating higher scores and indices, led imbalances in certain algorithms after clustering. This decision reinforced by visual representation in Fig. 2, illustrating silhouette score and Calinski-Harabasz index across varying cluster numbers,utilizing with Google Colab for implementation.

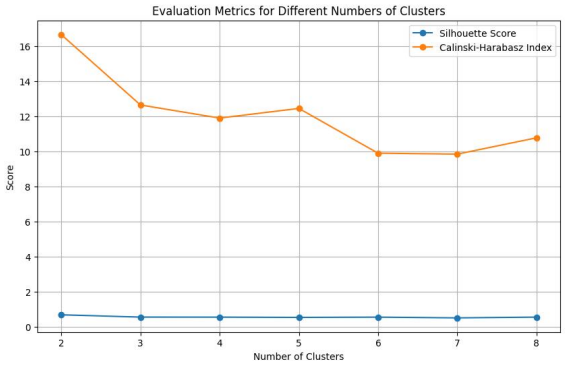

Fig. 2. Evaluation Metrics for Different Number of Clusters

A. *Metrics for Assessing Clustering Algorithm Performance*

When evaluating the effectiveness of clustering algorithms, two primary categories of metrics are considered: runtime performance and clustering quality.

1) ***Runtime Performance:*** Runtime performance of a clustering algorithm is a vital metric that measures the time taken for the algorithm to complete the clustering process. This metric indicative of the computational efficiency and scalability of the algorithms. Minimal runtime generally desirable, signifying faster processing and improved algorithmic efficiency. TABLE 1 represents run time execution according to the clustering algorithms. Notably, K-Means exhibited the shortest execution time, showcasing its computational efficiency. AHC closely followed and emphasizing its competitive performance. EM and Spectral clustering demonstrated slightly higher execution times.

TABLE 1: EXECUTION TIME OF CLUSTERING ALGORITHM

Clustering Algorithm	Execution Time(seconds)
AHC Execution Time	0.0317
EM Execution Time	0.2727
Spectral Execution Time	0.2749
K-Means	0.0384

2) ***Clustering Performance Metrics***

TABLE 2 represents clustering performance metrics including silhouette score and calinski-harabasz index. The evaluation of clustering algorithms reveals distinctive performances across silhouette score and calinski-harabasz index metrics. K-Means and AHC exhibit higher Silhouette Scores, indicating well-defined clusters, with AHC outperform in this aspect. Calinski-Harabasz scores, highlight AHC as the algorithm providing enhanced cluster separation and compactness.

TABLE 2: CLUSTERING PERFORMANCE METRICS

Clustering Algorithm	Silhouette Score	Calinski-Harabasz Index
K-Means	0.570219441	12.66243236
AHC	0.582916787	15.32831024
EM	0.51469946	6.742542429
Spectral	0.518157405	9.668457543

B. Assessing Prioritization Technique Performance

The evaluation of prioritization technique performance is measured by APFD rates. The focus is on TABLE 3 and TABLE 4, illustrating the pre and post-application rates of the APFD. These tables act as quantitative standards, representing the prioritization technique's influence on improving fault detection efficiency by comparing outcomes before and after its implementation.

TABLE 3: APFD BEFORE PRIORITIZATION

Clustering Algorithms	Cluster 1	Cluster 2	Cluster 3
K-Means	21.63	24.33	94.49
AHC	58.97	41.59	40.10
EM	20.83	75.18	35.68
Spectral	47.34	46.26	46.83

TABLE 4: APFD AFTER PRIORITIZATION

Clustering Algorithms	Cluster 1	Cluster 2	Cluster 3
K-Means	21.74	26.25	95.78
AHC	62.50	43.15	42.42
EM	24.09	75.25	35.74
Spectral	48.21	48.97	46.97

V. CONCLUSION

The performance evaluation of clustering algorithms and the subsequent impact on APFD rates provides valuable insights. The study utilized with four major clustering algorithms which AHC outperform well than other. Also the experimentation utilized a medium sized test case dataset, chosen to closely simulate real-life application scenarios, providing a representative basis for evaluating the effectiveness of clustering algorithms in practical software testing contexts. Results suggest that AHC could enhance fault detection capability of prioritization techniques, which contributing to more effective and efficient software testing processes. For future works we can extend exploring integration with machine learning models for comparison and addressing scalability challenges in larger-scale systems. This enhance the reliability and adaptability of the proposed clustering-based test case prioritization technique, emphasizing its effectiveness and suitability for various testing scenarios.

REFERENCES

[1] A. Lawanna, “The Theory of Software Testing,” 2012. [Online]. Available: https://www.researchgate.net/publication/236031163

[2] K. Yasar, “software testing,” WhatIs.com, Aug. 12, 2022. https://www.techtarget.com/whatis/definition/software-testing.

[3] A. Ngah, M. Munro, and M. Abdallah, “An Overview of Regression Testing”.

[4] S. Yoo and M. Harman, “Regression testing minimization, selection and prioritization: a survey,” Softw. Test. Verif. Reliab., vol. 22, no. 2, pp. 67–120, 2012.

[5] “A Detailed Analysis on Advantages, Disadvantages, Challenges and Risks of Regression Testing,” testsigma.com. https://testsigma.com/regression-testing/advantages-of-regression-testing

[6] P. Ranjan Srivastava, “Test case prioritization.” [Online]. Available: www.jatit.org

[7] T. Muthusamy and Seetharaman. K, “Effectiveness of Test Case Prioritization Techniques Based on Regression Testing,” International Journal of Software Engineering & Applications, vol. 5, no. 6, pp. 113–123, Nov. 2014, doi: 10.5121/ijsea.2014.5608.

[8] N. Gokilavani and B. Bharathi, "An Enhanced Adaptive Random Sequence (EARS) Based Test Case Prioritization Using K-Mediods Based Fuzzy Clustering," 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), Tirunelveli, India, 2020, pp. 567-572, doi: 10.1109/ICOEI48184.2020.9142966.

[9] P. Ramya, V. Sindhura and P. Vidya Sagar, "Clustering Based Prioritization of Test Cases," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, India, 2018, pp. 1181-1185, doi: 10.1109/ICICCT.2018.8473253.

[10] G. Chaurasia, S. Agarwal and S. S. Gautam, "Clustering based novel test case prioritization technique," 2015 IEEE Students Conference on Engineering and Systems (SCES), Allahabad, India, 2015, pp. 1-5, doi: 10.1109/SCES.2015.7506447.

[11] M. J. Arafeen and H. Do, "Test Case Prioritization Using Requirements-Based Clustering," 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Luxembourg, Luxembourg, 2013, pp. 312-321, doi: 10.1109/ICST.2013.12.

[12] S. Chaudhary and A. Jatain, "Performance Evaluation of Clustering Techniques in Test Case Prioritization," 2020 International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 2020, pp. 699-703, doi: 10.1109/ComPE49325.2020.9200083.

[13] V. DiFrancesco, “Gaussian Mixture models for clustering - towards data science,” Medium, Jan. 01, 2022. [Online]. Available: https://towardsdatascience.com/gaussian-mixture-models-for-clustering-3f62d0da675

[14] J. Shlens, “A Tutorial on Principal Component Analysis.” Accessed: Dec. 05, 2023. [Online]. Available: https://arxiv.org/pdf/1404.1100

[15] V. D. K. Rajendran and R. Pradeepa, “Effectiveness of Test Case Prioritization using APFD Metric: Survey,” ResearchGate, Feb. 2013, [Online]. Available: https://www.researchgate.net/publication/268034515_Effectiveness_of_Test_Case_Prioritization_using_APFD_Metric_Survey