Journal of Information Systems Engineering and Management
2025, 10(17s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Enhancing Software Testing with Genetic Algorithm and Binary Search: Integrating Error Classification and Debugging Through Clustering

[1]Aysh Alhroob, [1] Wael Alzyadat, [1] Ameen Shaheen, [2] Heba Nafez Jalal

[1]Department of Software Engineering, Al-Zaytoonah University of Jordan, Amman, Jordan
[2] Ministry of Local Administration, Amman, Jordan
**Corresponding Author: Aysh@zuj.edu.jo*

ARTICLE INFO

Received: 09 Dec 2024

Revised: 30 Jan 2025

Accepted: 10 Feb 2025

ABSTRACT

This paper improves the Binary Search Genetic Algorithm (BSGA) [1] for software testing by combining error classification and debugging through k-means clustering. The extended BSGA approach optimizes test-case generation, it improves method coverage, and makes their quality imperfect streamlines debugging by categorizing them into groups on the basis. This allows for more effective fault localization, identifies the most likely causes of failure, and provides actionable debugging insights. Experimental results show that the proposed approach increases the method coverage by 94% comparing with 88% by BSGA, reduces the number of test cases by 13%, reduces the debugging time by 44%, and in addition, the error classification and the integration of fault location methods reduces testing costs and makes faults a faster system. The approach shows significant improvements in testing efficiency, provides a comprehensive solution that integrates test case generation, bug classification, and bug fixing, and ultimately increases the effectiveness of the software testing process.

Keywords: Error Classification, K-Means Clustering, Software Debugging Technique, Test Case Optimization.

INTRODUCTION

Software testing is an important a part of software program engineering aimed at making sure the reliability, functionality, and average nice of software programs. The demand for powerful testing out has grown exponentially because of the increasing complexity of software program systems, which includes the challenges posed with the aid of huge-scale data and sophisticated architectures. Traditional testing techniques, together with guide testing out and exhaustive testing out, are frequently consuming time, prone to error, and inefficient while carried out to large software systems, making them impractical for cutting-edge applications. As a result, automated test case generation and optimization have end up critical for improving the performance of software testing methods.

Genetic Algorithms (GAs) have gained substantial interest because of their capability to discover large enter areas and optimize solutions. The Binary Search Genetic Algorithm (BSGA) is one such approach that refines the input domain data for test cases through iteratively narrowing down the search space via binary search. This permits for the generation of super test cases with optimized path coverage, reducing the wide variety of test cases required to test a program. However, at the same time as BSGA can enhance take a look at case technology, it lacks included mechanisms for errors type and debugging, which might be essential for improving the debugging manner and decreasing time spent on fault localization.

To overcome those obstacles, this paper proposes an enhancement to the BSGA by incorporating clustering strategies for error type and fault localization. By making use of K-means clustering to classify errors based on their features and execution styles, the approach can become aware of common errors types, prioritize test cases, and optimize debugging techniques. This improved method not simplest improves take a look at case efficiency but additionally streamlines the debugging process by providing extra correct insights into the resources of failure, ultimately decreasing debugging time and improving average software quality.

RELATED WORKS

Hybrid algorithms, combining GAs with other techniques, have been widely used for optimizing test case generation. These methods seek to balance path coverage and testing cost. In [1], the authors have combined GA with mutation and data flow analysis to enhance test case effectiveness. The BSGA integrates binary search to refine input domains, optimizing test case selection for Big Data applications. However, existing approaches, including BSGA, often lack integrated error classification and debugging, which are critical for improving the overall efficiency of software testing [1].

A. Error Classification

Classifying errors based on their type and severity is essential to improve error prevention. Machine learning (ML) techniques have been extensively used in this field. For example, Xie et al. in [2]used decision trees to classify faults, classify faults and help optimize debugging processes. In a similar mood, Huang et al. in [3]used neural networks to classify faults, showing how deep learning can enable fault classification based on training data. More recently, in [4], a Deep Neural Network (DNN) model was developed to predict the type of software error based on program execution traces. Their method achieved an impressive accuracy of 90% in classifying errors into predefined groups, demonstrating the potential of DNNs to improve debugging efficiency. Another notable contribution is in [5], who combined transfer learning with error classification. This approach used knowledge from one software project to classify errors in another, providing innovative solutions for error classification in projects with limited historical data. This study shows how machine learning techniques can be applied improved has been used to improve error classification.

B. Clustering Techniques in Software Testing

Clustering is any other area wherein significant development has been made, specifically in grouping related cases and errors. The use of clustering in software trying out changed into first explored in [6], who established its applicability for grouping comparable errors and optimizing software program execution. In [7], the authors implemented clustering for test case prioritization, grouping similar test cases collectively to maximize coverage while minimizing take a look at execution time. More recently, the authors in [8] delivered a hybrid version that mixed k-method clustering with an evolutionary set of rules for test case prioritization. Their version stepped forward test efficiency with the aid of lowering the quantity of test cases required even as preserving excessive paths coverage.

In the context of error class, Zhang et al. In [9] carried out unsupervised clustering methods, inclusive of Self-Organizing Maps (SOM), to institution errors data. These techniques offer flexibility in figuring out formerly unknown errors patterns without requiring categorized facts. The clustering process is primarily based on error characteristics which includes execution paths, enter values, and system state, and it considerably reduces the attempt required for debugging by way of automatically figuring out similar errors types.

C. Hybrid Algorithms for Test Case Generation and debugging

The integration of Machin Learning (ML) and hybrid algorithms for test case technology and error debugging has received great interest in current years. Wang et al. In [10] proposed a hybrid technique that combined Genetic Algorithms with deep learning knowledge of techniques for generating and choosing test cases. Their model applied neural networks for predicting the maximum important take a look at test cases to run, which were then optimized using GAs. This technique advanced path insurance and decreased the number of test cases by way of focusing on those that have been most probably to reveal faults.

Another in [11] mixed GAs with clustering techniques for producing take a look at test cases primarily based on errors types. By clustering comparable errors and the use of GAs to adapt test cases based totally on those clusters, the approach become capable of optimize the era of test cases in a manner that immediately addressed the maximum common errors, enhancing both the performance and effectiveness of the trying out procedure. Their outcomes showed a 15% discount in testing time compared to traditional techniques, with stepped forward errors detection costings.

Furthermore, Liu et al. In [12] proposed a hybrid testing approach combining GAs, clustering, and fault localization. Their works focused on decreasing debugging time by way of applying clustering algorithms to organization comparable errors, and then the usage of fault localization to awareness on the most in all likelihood faulty code

segments. This method progressed debugging efficiency via 38% compared to conventional techniques, demonstrating the effectiveness of clustering and fault localization in aggregate with GA.

D. Integration of Clustering and Error Classification with Hybrid Algorithms

While error clustering techniques have validated useful independently, their integration into hybrid algorithms like BSGA has been limited. Few research have explored this blended technique, but some latest improvements are beginning to bridge this gap. For example, in [13], Kumar et al. proposed an included approach for test case era and errors class the use of a mixture of GAs and k-Means clustering. Their method used clustering errors into classes, which had been then used to optimize the test technology process. This integration reduced the variety of test cases via 12% while improving test coverage, suggesting that errors type and clustering can substantially the performance of hybrid algorithms.

In [14], a novel approach become proposed combining Genetic Algorithm (GA) optimization with the k-Means clustering set of rules to decorate its performance on excessive-dimensional datasets. The integration of GA improves initial centroid choice and clustering configuration, addressing k-Means' sensitivity to preliminary conditions and its tendency to converge to nearby optima. Experimental outcomes on artificial and real-world datasets exhibit improved accuracy, robustness, and computational performance as compared to conventional clustering methods, making the hybrid set of rules a precious device for statistics mining.

PROPOSED APPROACH

This section presents the extended BSGA approach with integrated error classification and debugging components, focusing on how clustering enhances test case generation and debugging as shown in Fig.1. The BSGA approach optimizes test case selection by combining Binary Search and Genetic Algorithms:

1. Input Domain Reduction: Binary Search narrows down potential input domains.
2. Test Case Optimization: GAs generate test cases to maximize path coverage.
3. Path Coverage Evaluation**:** The effectiveness of each test case is evaluated based on its ability to cover program paths.

E. Error Classification Using Clustering

Using clustering to group similar problems found at some point during testing is part of the approach. It starts with feature extraction, in which error characteristics, execution records, and aid metrics are drawn from failed test cases. These extracted features serve as consider for the clustering procedure, in which an approach is carried out to organize errors into clusters based on their similarities. This clustering enables identifying routine mistake patterns, enabling a based evaluation of the problems. To enhance debugging, spectrum-based totally fault localization is hired, narrowing the focal point to the probable sources of errors and streamlining the research system.

Once errors are clustered, the approach offers debugging suggestions through its dedicated mechanism, as shown in Fig. 1, first, fault localization is leveraged to investigate execution paths, pinpointing the most likely defective components. This is complemented with the aid of debugging suggestions, which provide actionable insights, such as reviewing unique features or examining input conditions that is probably contributing to the errors. These recommendations intention to reduce the effort and time required for resolving the recognized issues.

Finally, the approach contains a feedback loop that refines test case generation based totally on insights received from errors classification and debugging. Test cases focused on excessive-severity errors are prioritized, ensuring an effective testing out manner. This iterative refinement no longer best improves the worthy of the take a look at group however also allows discover hidden or complex bugs, leading to more sturdy software program structures.

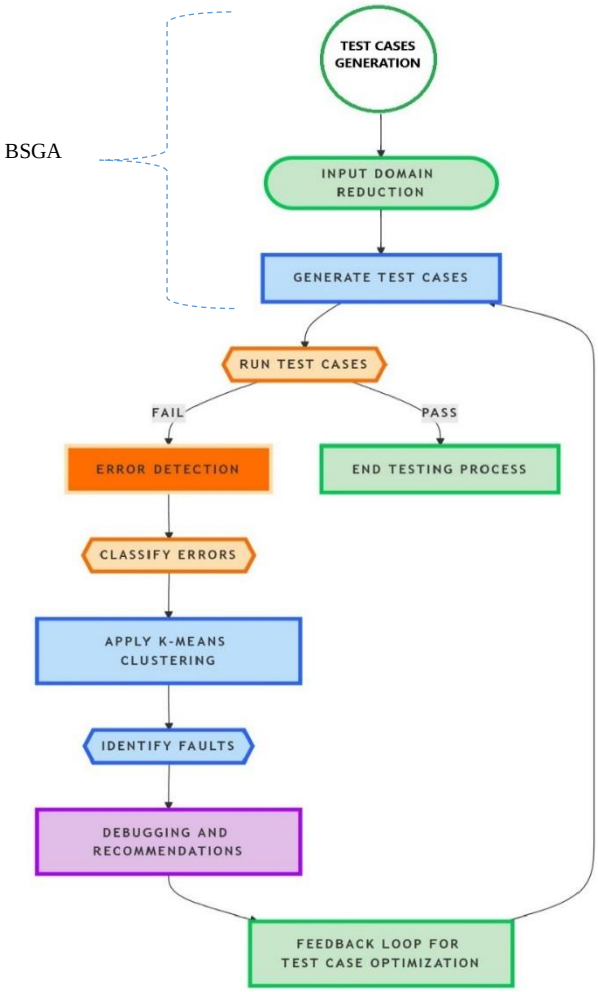

Figure 1. Proposed Approach

1. Input Domain Reduction Using Binary Search

The first step of the proposed approach involves reducing the input domain using Binary Search, systematically narrowing down the potential input values for testing. The input domain, represented by *(D = {x_1, x_2, ..., x_n}),* is divided into two halves recursively Equation.1 and1.1:

$$D\_1 = D[: n/2],\ D\_2 = D[n/2:] \quad (1)$$

The selection of the domain continues based on a fitness function (F(*x*)) that evaluates the effectiveness of each test case in exposing faults:

$$D\_\{selected\} = arg\ max\_\{D\_1, D\_2\}\ F(D\_i) \quad (1.1)$$

This process is repeated until the input domain reaches a pre-defined threshold size or meets the stopping condition, ensuring that the focus remains on the most impactful test cases.

2.Test Case Optimization Using GA

Once the input domain is narrowed, Genetic Algorithms are employed to optimize test case generation. This stage comprises the following steps:

Population Initialization: The population of potential test cases is initialized, represented *as (P_0 = {t_1, t_2, ..., t_m}).* Each test case represents a chromosome containing parameters from the reduced input domain.

Fitness Function Evaluation: Each test case is evaluated based on a fitness function in Equation.2.

$F(t) = sum_{i=1}^{k} (c_i(t) / C_total)$ (2)

Where, $c_i(t)$ represents the path coverage by test case t, and C_total is the total possible path coverage. This evaluation ensures that only the most effective test cases are selected for subsequent generations.

Selection and Crossover: Individuals with the highest fitness scores are selected for crossover to produce offspring. Given parent chromosomes t_a and t_b:

$t_new = t_a[:p] + t_b[p:]$ (3)

Where, p represents a randomly chosen crossover point.

Mutation: To maintain diversity, mutation is applied to randomly selected genes within chromosomes with a probability P_m: $t_mut = t_new + Delta$, where $Delta$ denotes the mutation applied to a gene.

The selection, crossover, and mutation processes are repeated until a specified stopping criterion is met, such as convergence in fitness values or the maximum number of generations.

3. Path Coverage Evaluation

Path coverage evaluation is crucial for assessing the effectiveness of generated test cases [15]. The objective is to maximize path coverage while reducing redundancy:

$C_eff = sum_{i=1}^{N} Cov(t_i)/C_total.$ (4)

Where, $Cov(t_i)$ denotes the path coverage achieved by test case t_i, and C_total represents the maximum possible path coverage.

4. Error Classification Using Clustering

The next step involves error classification through clustering techniques (see Fig.2), specifically k-means clustering:

Feature Extraction: Extract features from failed test cases, $E = \{e_1, e_2, ..., e_r\}$, which include attributes like error type, execution trace, and resource metrics.

- Apply k-Means Clustering: K-means clustering is used to group errors into clusters based on their similarity, minimizing the intra-cluster distances: minimize

$sum_{i=1}^{r} sum_{j=1}^{K} ||e_i - mu_j||^2.$ (5)

Where, mu_j represents the centroid of cluster j.

Error Grouping: Each error is assigned to the nearest cluster C_j: $C_j = arg\ min_{j} ||e_i - mu_j||$, where $||e_i - mu_j||$ is the Euclidean distance between an error and the cluster centroid. This clustering helps identify recurring error patterns, making the debugging process more structured.

Input: *Failed Test Cases E = {e_1, e_2... e_r}, Number of Clusters K*
Function *KMeansClustering (E, K):*
Initialize centroids randomly: mu_j for j = 1 to K
Repeat:
Assign each error e_i to the nearest centroid mu_j:
C_j = {e_i: ||e_i - mu_j|| is minimum for all j}
Update centroids:
mu_j = mean (C_j)
Until convergence
Return Clusters C
Output: *Error Clusters C = {C_1, C_2... C_K}*

Figure 2. Error Classification Using Clustering

5. Fault Localization and Debugging Recommendations

Once errors are classified, fault localization is conducted to identify probable faulty components (see Fig.3:

Spectrum-Based Fault Localization: Fault localization utilizes spectrum analysis of execution paths to determine the

most probable sources of faults. The suspiciousness score $S(f\_i)$ for a fault f_i is calculated as:

$$S(f\_i) = (a\_ef / (a\_ef + a\_nf)) / (a\_ep / (a\_ep + a\_np)), \quad (6)$$

Where, a_ef, a_nf, a_ep, and a_np represent the number of times fault f_i is executed during failing and non-failing test cases as shown in Equation.6.

Debugging Recommendations: Based on the clustering and fault localization results, actionable debugging recommendations are provided, which include reviewing specific faulty components and prioritizing high-severity errors. Failed test cases are analyzed and classified into clusters using k-Means clustering. Error features like type, execution trace, and system metrics are extracted and grouped based on similarity. Clustering helps uncover recurring patterns, allowing testers to identify and prioritize issues systematically. This structured approach makes the debugging process faster and more efficient.

```
Input: Error Clusters C, Execution Data
Function FaultLocalization(C, ExecutionData):
 For each cluster C_j in C:
   Analyze Execution Paths to identify faulty components:
       S (f_i) = (a_ef / (a_ef + a_nf)) / (a_ep / (a_ep + a_np))
   Add Debugging Recommendations based on high S (f_i)
scores:
       Recommendations = "Review component X, analyze input
Y" Return Fault Locations, Recommendations
Output: Fault Locations, Debugging Recommendations
Function
```

Figure 3. Fault Localization and Debugging Recommendations

6. Feedback Loop for Test Case Optimization

As shown in Fig. 4, a feedback loop is introduced to refine test case generation based on insights from error classification and debugging:

Prioritization of High-Severity Errors: Test cases targeting high-severity errors are prioritized to improve the efficiency of future testing. The updated fitness function (see Equation.7) incorporates the severity of errors:

$$F'(t) = F(t) + w\_e * H(e). \quad (7)$$

Where, w_e is the weight for high-severity errors and $H(e)$ represents the error severity metric.

Iterative Improvement: The entire process is iterated to continuously improve path coverage, reduce undetected faults, and enhance the robustness of the software system.

```
Input: Updated Test Cases T, Error Data E, Weight w_e,
Error Severity H (e)
Function FeedbackLoop(T, E, w_e, H):
  For each test case t in T:
   Update Fitness Function:
     F'(t) = F (t) + w_e * H (e) for errors in E
   Re-optimize test cases using GeneticAlgorithmOptimize()
  Return Refined Test Cases
Output: Refined Test Cases
```

Figure 4. Feedback Loop for Test Case Optimization

F. Example: Software Bug in Web Application

1) Initial Test Case Generation with BSGA:

Input Domain Reduction: Start with a large variety of input values for testing a login characteristic, e.g., user credentials (username, password). The binary seek will help slender down the possible mixtures.

Genetic Algorithms: Evolve take a look at test cases through combining distinctive user credentials to cover all paths (successful login, invalid credentials, and so on.).

2) Error Detection:

Run the generated test cases, and suppose Test Case1 with username "admin" and password "wrong bypass" fails because of an authentication error.

3) Error Classification the use of K-approach Clustering:

Extract errors features: for failed test cases, document errors messages ("Invalid password"), execution paths, and other applicable information (e.g., HTTP reputation code).

Apply ok-means clustering to institution errors. For example, errors may cluster into "Authentication Errors," "Database Connection Failures," etc.

4) Fault Localization:

Using spectrum-primarily based fault localization, perceive that the hassle might be within the authentication code (e.g., AuthService.Java).

5) Debugging and guidelines:

Based on the clustering and fault localization effects, the approach shows reviewing the authentication module and examining area test cases wherein invalid credentials trigger errors.

6) Feedback Loop for Test Case Optimization:

With the brand new insights, replace the take a look at cases. Prioritize scenarios with invalid credentials for further testing out, specializing in diverse username and password combos.

EXPERIMENTAL RESULTS

The experimental assessment of the extended BSGA approach become carried out the usage of Defects4J [16], an extensively identified benchmark dataset for software program testing and debugging. This dataset consists of real-international Java programs with reproducible bugs, permitting a comprehensive evaluation of the proposed approach's effectiveness in test case technology, path coverage, errors grouping, and debugging efficiency.

In comparison to the same old Genetic Algorithm (GA), which required 370 test cases to gain 80% path coverage, the unique BSGA reduced the wide variety of test cases to 250 at the same time as improving path coverage to 88%, as proven in Table 1. The enhanced BSGA similarly more desirable overall performance, attaining 94% path coverage with 150 test cases, reflecting a 59.5% reduction in take a look at cases as compared to the usual GA at the same time as substantially enhancing coverage.

The errors classification competencies of the extended approach were evaluated the use of each F1-score and accuracy metrics. As shown in Table 1, the approach performed an F1-score of 0.85, demonstrating a strong balance among precision and don't forget in grouping errors. Additionally, the accuracy of the errors class reached 0.90, indicating that 90% of the classified errors had been correctly detected. These effects underscore the effectiveness of the clustering set of rules in grouping similar errors extracted from the Defects4J dataset. In phrases of debugging efficiency, tremendous improvements had been determined, with the common debugging time per fault reduced appreciably.

In debugging time per fault, reduced from 50 minutes in the standard GA to 40 minutes in the original BSGA and further to 25 minutes in the extended BSGA. These improvements are attributed to the integration of fault localization techniques and clustering-based recommendations.

Table 1. Experiment for Enhanced BSGA Approach

Metric		**Standard GA**	**BSGA**	**Enhanced BSGA**
Test Cases Generated		370	250	150
Path Coverage (%)		80%	88%	94%
Debugging Time per Fault (min)		50	40	25
F1-Score		0.75	0.80	0.85

Accuracy		0.85	0.87	0.90
Testing Cost Reduction (%)		Baseline	32.4%	59.5%

Using Defects4J allowed the approach to be evaluated in real-world scenarios, demonstrating its ability to optimize test cases, increase path coverage, and streamline debugging processes. Furthermore, the approach significantly reduced testing costs, achieving a 59.5% cost reduction compared to the standard GA by minimizing the required number of test cases and optimizing computational resources. Overall, the extended BSGA approach proved to be an efficient and cost-effective solution for software testing and debugging.

CONCLUSION AND FUTURE WORK

This paper presented an enhanced BSGA approach that integrates mistakes classification and debugging using clustering strategies. Experimental outcomes reveal that the extended approach improves path coverage, reduces debugging time, and lowers testing costs. Future work will discover the scalability of the enhanced BSGA approach while implemented to large and extra complex software program structures, assessing its performance in diverse environments. Integration with Continuous Integration/ continuous delivery (CI/CD) pipelines is also deliberate, making an allowance for real-time, automated error classification and debugging. Furthermore, purpose enhancements could involve incorporating advanced strategies, including symbolic execution or reinforcement gaining knowledge of, to enhance fault localization and make errors classification even more accurate. Lastly, the impact of the approach in real-world packages and its capacity for version across unique domains can be investigated.

Acknowledgements

We extend our gratitude to Al-Zaytoonah University of Jordan for their financial support in publishing this research and for providing the necessary resources to conduct this study.

REFERENCES

[1] Alhroob, W. Alzyadat, A. T. Imam and G. M. Jaradat, "The Genetic Algorithm and Binary Search Technique in the Program Path Coverage for Improving Software Testing Using Big Data," Intelligent Automation and Soft Computing, vol. 26, no. 4, p. 725–733, 2020.
[2] X. Xie, H. Zhang and Y. Hu, "A Decision Tree Approach to Error Classification in Software Testing," Software Testing, Verification & Reliability, vol. 24, no. 5, p. 356–375, 2014.
[3] Y. Huang, Y. Zhang and X. Zhi, "Error Classification in Software Systems Using Deep Neural Networks," Journal of Software: Evolution and Proces, vol. 29, no. 2, p. 1–15, 2017.
[4] J. Lyu, H. Liu and P. Zhang, "Deep Learning-Based Error Classification for Software Systems," in in Proceedings of the International Conference on Software Engineering, 2020.
[5] X. Zhang, S. Wang and L. Li, "Leveraging Transfer Learning for Error Classification in Software Projects," IEEE Transactions on Software Engineering, vol. 48, no. 4, p. 1563–1576, 2022.
[6] M. Chen, S. Mao and Y. Liu, "Big Data: A Survey," Mobile Networks and Applications, vol. 19, no. 2, pp. 171-209,, 2014.
[7] Amiri, R. S. Alagheband and H. Aziz, "Clustering-Based Test Case Prioritization Using K-Means," Journal of Software Engineering and Applications, vol. 19, no. 1, pp. 401-416, 2018.
[8] N. Kumar, P. Bhatnagar and M. A. Bansal, "Hybrid Evolutionary Algorithm with K-Means Clustering for Test Case Prioritization," IEEE Access, vol. 9, pp. 14078-14088, 2021.
[9] L. Zhang, C. Li and M. Xie, "Error Classification via Self-Organizing Maps (SOM) in Software Testing," Journal of Software Engineering, vol. 19, no. 3, pp. 215-227, 2020.
[10] J. Wang, R. L. Zhang and T. Xu, "Combining Genetic Algorithm and Deep Learning for Test Case Generation," in in Proceedings of the IEEE International Conference on Software Testing, 2021.
[11] Y. Jiang, Z. Liu and L. Wang, "Test Case Generation Using Genetic Algorithms and Clustering for Error Patterns," ournal of Software: Testing, Verification & Reliability, vol. 32, no. 4, pp. 1-20, 2022.
[12] Y. Liu, X. Li and W. Zhang, "A Hybrid Approach Combining Genetic Algorithms, Clustering, and Fault Localization for Software Testing," Software Testing & Verification, vol. 53, pp. 1-14, 2023.
[13] V. Kumar, A. Patel, Gupta and Ramesh, "Integration of Genetic Algorithms and K-Means Clustering for Software Test Case Generation and Error Classification," Software Quality Journal, vol. 28, no. 1, pp. 131-14, 2020.

[14] R. Raman, V. Kumar, B. G. Pillai, D. Rabadiya, S. Patre and R. Meenakshi, "The Impact of Enhancing the k-Means Algorithm through Genetic Algorithm Optimization on High Dimensional Data Clustering Outcomes," in International Conference on Knowledge Engineering and Communication Systems (ICKECS), 2024.

[15] Y. Al-Kasabera, W. Alzyadat, A. Alhroob, S. A. Showarah and A. Thunibat, "An Automated Approach to Validate Requirements Specification," COMPUSOFT, An international journal of advanced computer technology, vol. 9, no. 2, pp. 3578-3585, 2020.

[16] R. Just, "The Defects4J dataset version 2.0.0," 1 Dec 2024. [Online]. Available: https://github.com/rjust/defects4j.