
Journal of Informatics and Web Engineering

Vol. 5 No. 3 (October 2026)

eISSN: 2821-370X

Predictive Test Case Prioritization in Agile Workflows: A Machine Learning Approach with SMOTE-ENN and PCA

Kavinraj Baskaran¹, Junaidi Abdullah², Sin-Ban Ho^{3*}, Said Suleiman Naseer Al Harthy^{4**}, Ian Chai⁵

^{1,2,3,5} Faculty of Computing and Informatics, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Malaysia

⁴ Modern College of Business and Science, Bosher, PC 133, Muscat, Sultanate of Oman

* corresponding author: (sbho@mmu.edu.my; ORCID: 0000-0003-2995-2120)

** corresponding author: (said@mcbs.edu.om; ORCID: 0000-0003-2308-5142)

Abstract - Test Case Prioritization (TCP) is an essential part of agile software development, allowing for the identification and execution of critical test cases early in the process to maximize fault detection efficiency. Traditional TCP techniques often depend on static metrics, like code complexity and historical pass/fail rates, that do not conform with the dynamic nature of Agile workflows. To overcome this, the proposed research presents a discriminative machine learning-based predictive prioritization model that incorporates historical defect data, execution history, as well as requirement criticality to improve auspiciousness of TCP. The study assesses a range of machine learning models like Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting across two datasets, namely CarFinTest and MISPriority. Though the target variable in the MISPriority dataset was nicely structured with respect to the feature set, allowing models to reach a predictive accuracy up to 78.63%, the CarFinTest showed a high degree of imbalance in classes along with poor class separability. Through iterative improvements, the addition of SMOTE-ENN to balance classes, dimensionality reduction via PCA, and feature engineering techniques all greatly increased model performance, with Random Forest and Gradient Boosting performing the best. In addition, the proposed research presents an automatic dashboard for preprocessing dataset, training models, and prioritizing them for implementation into an Agile environment for practical usability. Through adaptive machine learning-based techniques, this research connects traditional and dynamic test prioritization methods, realizing a scalable and data-oriented approach compatible with Agile processes. Future work should focus on enhancing real-time prioritization through incremental learning techniques and expanding dataset generalization across diverse software development projects.

Keywords— Test Case Prioritization, Agile Workflows, Machine Learning, Feature Engineering, Predictive Modeling

Received: 19 July 2025; Accepted: 20 March 2026; Published: 10 October 2026

This is an open access article under the [CC BY-NC-ND 4.0](#) license.



1. INTRODUCTION

Agile software development has emerged to be the industry standard of developing high quality software elements in less time. This iterative approach encourages quick feedback loops and ongoing refinement, which is why testing



Journal of Informatics and Web Engineering

<https://doi.org/10.33093/jiwe.2026.5.3.18>

© Universiti Telekom Sdn Bhd. This work is licensed under the Creative Commons BY-NC-ND 4.0 International License.

Published by MMU Press. URL: <https://journals.mmupress.com/jiwe>

becomes an integral part of the development process. The increasing complexity of software systems and the constraints of resources in Agile workflows make Test Case Prioritization (TCP) an important and challenging problem. TCP refers to the act of ordering test cases in such a way that the efficiency of defect detection is improved or that the essential functions will be tested as early as possible. One of the common test requirements in agile environments is prioritization and when we fail to prioritize correctly the inept team who works on the project can often lead to wastage of efforts in terms of missed defects being found well within the project cycle or the most frequently used areas of the application not being accounted for in your testing or costing you extra in terms of redundant and irrelevant executions. Moreover, agile workflows require extremely fast feedback loops, in which there is no time for thorough testing or manual prioritizing.

Researchers have studied computer approaches to prioritization using traditional techniques (mathematical optimization, heuristics, rules), generally based on static or merely heuristic metrics; code coverage, historical pass/fail rates. They are unable to comply with the collaborative mobility of an agile-like force, which needs flexibility and data-driven connections. The current work attempts to fill this gap by creating a predictive model incorporating historical data, probability of defect detection, execution history and importance of the requirements, which utilizes machine learning to prioritize test cases for execution. In this paper, the authors analyze the metrics that influence test case prioritization, such as defect detection likelihood, execution time, code complexity, and requirement criticality. This helps to engineer a machine learning-based predictive model that ranks test cases using historical test data and defect metrics. Next, the model performance is further validated in real-world agile workflows by comparing it to traditional test prioritization methods. The focus of this research is to improve the efficiency of testing the key factors in prioritizing agile test cases. With the predictive prioritization supported by historical data, defect metrics can be insightful to reduce the cost. Furthermore, the engineered model would effectively increase the quality of software using minimum number of test cases in agile projects, overcoming traditional method obstacle by incorporating predictive analytics into TCP methodology.

2. LITERATURE REVIEW

TCP is an important task in the domain of software quality assurance, especially in agile workflows, where iterative cycles and limited resources make it essential to maximize the diagnostic value of each test. TCP written works often distinguish between static metrics (which measure constants of the code) versus dynamic metrics (which measure runtime characteristics) as the leading indicators of prioritization. Static metrics provide the authors with insights that can be reasonably predictive of the underlying properties coding basis that may be correlated with defect-proneness. These metrics would not change without code changes. On the other hand, cyclomatic complexity counts the paths independent of each other in a codebase when passing through a graph and is widely used as an indicator of how fault-prone or difficult to test a piece of code. Ibriwesh et al. [1] emphasizes its importance on agile projects, while creating modular development code, leading to hyper interconnected code. Given high-risk modules, teams can concentrate testing efforts on those parts of the code that would be most likely to harbor defects. Factoring in cyclomatic complexity for prioritization helps catch fault-prone pathways sooner than later and mitigates the risk of potential bugs moving through the development loops.

Halstead metrics, which measure program complexity as a function of the length, vocabulary, and difficulty of the codebase, are used to predict modules that are likely to be faulty. For example, Marijan [2] shows that these metrics, together with runtime metrics such as defect detection history significantly enhance fault prediction in TCP models. Using two scales enables Agile teams to make an informed decision about where to best invest their limited testing effort on any given project. Code coverage measures how much of the code is exercised by a test suite. While high coverage is desirable, as it guarantees testing of critical functionality, Mahdiah et al. [3] warns that coverage does not ensure defect detection, as observed in agile workflows with changing requirements. When one combines the coverage data with metrics that represent the fault-proneness of the system, it could improve the prioritization by ensuring that the area of maximum value is highly tested. Meanwhile, dynamic metrics are extracted during the execution of the tests to record how tests behave and how the overall system performs, allowing them to prioritize them based on activity in a fast-paced agile environment.

Fast feedback loops are critical to agile workflows, which aligned with iterative development cycles. Test execution time is another critical metric [4],[5], as shorter tests give faster insights into system health, allowing teams to find and fix defects sooner. By focusing on smaller tests in the early cycles, agile encourages developers to make incremental progress in shorter cycles. Histories of defect detection rates, test pass/fail, and defect density are strong

predictors for how one can prioritize test cases. In an example from the hierarchically configurable Bayesian toolbox, Mahdich et al. [3] stresses that historical defect data is essential input in Bayesian models. This is mainly due to this kind of data that is critical for small to medium enterprises (SMEs) that have limited resources. With this available data, agile teams can focus on high failure rate tests that more frequently cause critical defects. User stories [6] are prioritized according to their business value in agile methodologies, and test prioritization follows the same pattern. According to Goyal [7], test cases are aligned with high-priority requirements, then one of the most important functionalities can be tested at the beginning of the sprint. By aligning business goals with testing priorities, agile teams deliver more value overall, leading to better customer satisfaction.

With the emergence of machine learning (ML), TCP was made data-driven and adaptive. ML techniques overcome the limitations of manual or static prioritization approaches to provide scalable [8],[9] as well as dynamic techniques, which are crucial for agile continuous integration and continuous delivery or deployment (CI/CD) pipelines. Decision trees are interpretable, making them suitable for projects where clear explanations of prioritization decisions are needed [10],[11]. Random Forests are an ensemble technique that improves performance and reduce overfitting. In agile data driven workflows, Random Forests are very useful, where historical defect data, cyclomatic complexity etc., used to find fault prone test cases. Random Forests are known for their scalability and accuracy, making them a good fit for agile projects [12],[13] where datasets and requirements change constantly. Further supervised ML of Gradient Boosting models like XGBoost and LightGBM are known to handle enormous imbalanced datasets well. According to Marijan [2], these models have been shown to perform better than conventional classifiers when it comes to prioritizing fault-prone test cases, especially in CI environments [14],[15]. Such models are valuable during agile processes, where predicted defects should be prioritized according to the pace at which defects manifest, without sacrificing prediction accuracy.

In unsupervised learning models, TCP is improved with the introduction of clustering algorithms which can be used to group similar test cases based on certain metrics (like fault-proneness or execution time) to promote diversity in the test cases faced. According to Mahdich et al. [3], clustering showed significant improvement in defect detection efficiency in terms of less redundancy and better coverage. In agile projects with minimal resources [16] [17], this strategy is especially useful in ensuring a wide range of different test cases are tackled in order of greatest impact. Probabilistic reasoning is used in Bayesian models to predict defect-proneness. In sparsely populated spaces closer to SMEs, [18],[19] illustrate their success. Bayesian models allow agile teams to stack rank and prioritize even when historical data is sparse. Such probabilistic thinking could echo agile tenets of flexibility and adaptability, enabling teams to respond dynamically to shifts in project needs.

TCP has gained significant attention in recent years [20],[21], but several challenges are still pertinent within agile frameworks. Since agile workflows require rapid iterations, dynamic change in requirements, and frequent test executions, there are demands for highly adaptive prioritization techniques [22],[23]. Traditional TCP approaches have largely been developed in the context of conventional continuous integration (CI) environments, with a more stable and structured test case execution traces. This lack of agile-specific studies leaves many important gaps that warrant resolution. The most apparent gaps are around agile-focused TCP methodologies. Nevertheless, agile testing introduces additional complexities, such as changing user stories, varying defect risks, and the requirements to provide timely feedback, despite extensive research on the topic in CI/CD settings. The existing TCP techniques operate under the assumption that the test cases do not change after prioritization, but in an agile process, priorities would change due to addition of user stories and modification of user stories should be reflected in TCP. Previous studies [24],[25] highlighted that agile tests prioritization should be dynamic, by considering requirement volatility and the influence of exploratory testing, but there is limited research available for operationalization of above strategies effectively.

Another key limitation in current TCP research remains the lack of incorporation of real-time data in the prioritization process. Most models are based on historical test execution, defect reports and static metrics like cyclomatic complexity and code coverage. However, agile development calls for a much more dynamic treatment, where prioritization decisions are dictated by real-time test results, metrics from system performance and changing failure patterns. According to Zhao et al. [26], using feedback from execution in real-time for prioritization is a powerful tool because it enables the team to identify rising risks and shift the focus of the tests. This prompts the advanced concept of real-time adaptive TCP but is rather underutilized, especially in situations where agile teams might need to dynamically reprioritize tests based on the outcomes of the sprint.

Moreover, scalability [27],[28] is still an open issue for existing TCP methods, especially in agile projects with large and highly dynamic test suites. Clustering and probabilistic approaches like Bayesian inference have been suggested

to improve prioritization, although they tend to encounter computational bottlenecks when they work on a large scale. Although effective in small datasets with well-defined prior distributions, Bayesian models can struggle at the other extreme with high-dimensional, and frequently updated agile test suites. Clustering-based approaches are often costly in terms of computation as frequent additions and modifications of test cases could involve recalculation of similarity metrics. The work by Teoh et al. [29] requires the facilities for high efficiency, how casual, gradual ensemble learning justice to be maintain for flexibility with minimal computational cost.

Another key missing piece involves the metrics, which we use to evaluate TCP. Traditional metrics like precision, recall, and overall accuracy could give a basic sense of how a model is performing, but they can also miss end-to-end nuances of agile-level test execution. These should include advanced prioritization metrics like Mean Reciprocal Rank (MRR) and Normalized Discounted Cumulative Gain (NDCG) that would provide a deeper insight into the correctness of the prioritization model with regards to the expected ordering of running the tests. The concept of DDCE (Defect Detection Cost Efficiency) could be added to assess how well prioritization minimizes testing effort while maximizing defect detection. According to Marijan [2], the practical relevance of TCP in agile practices increased if metrics allow long-term trends assessment in defect detection with efficiency gains over multiple sprints. However, little has been written about these advanced evaluation techniques. This poses the needs to engineer the prioritization model using accuracy scores evaluated with real-world machine learning effects [30].

3. RESEARCH METHODOLOGY

This research design in Figure 1 consists of five high-level phases; each focused on an important methodological aspect for creating and assessing a predictive model by employing TCP during agile development processes. In the first phase, data are gained for domain-specific datasets such as complexity, time, cost and priority, is mainly focused. Attributes that are crucial for TCP were selected to ensure the relevance of the datasets with respect to agile workflows. Essential processes in the second phase are to treat missing values, normalizing continuous variables for consistency, and aligning priority labels. The SMOTE-ENN technique is then utilized to alleviate class imbalance with class balance while maintaining a clean data set. The data preprocessing does data verification, and identification of outlier data points, which need to be either corrected or handled.

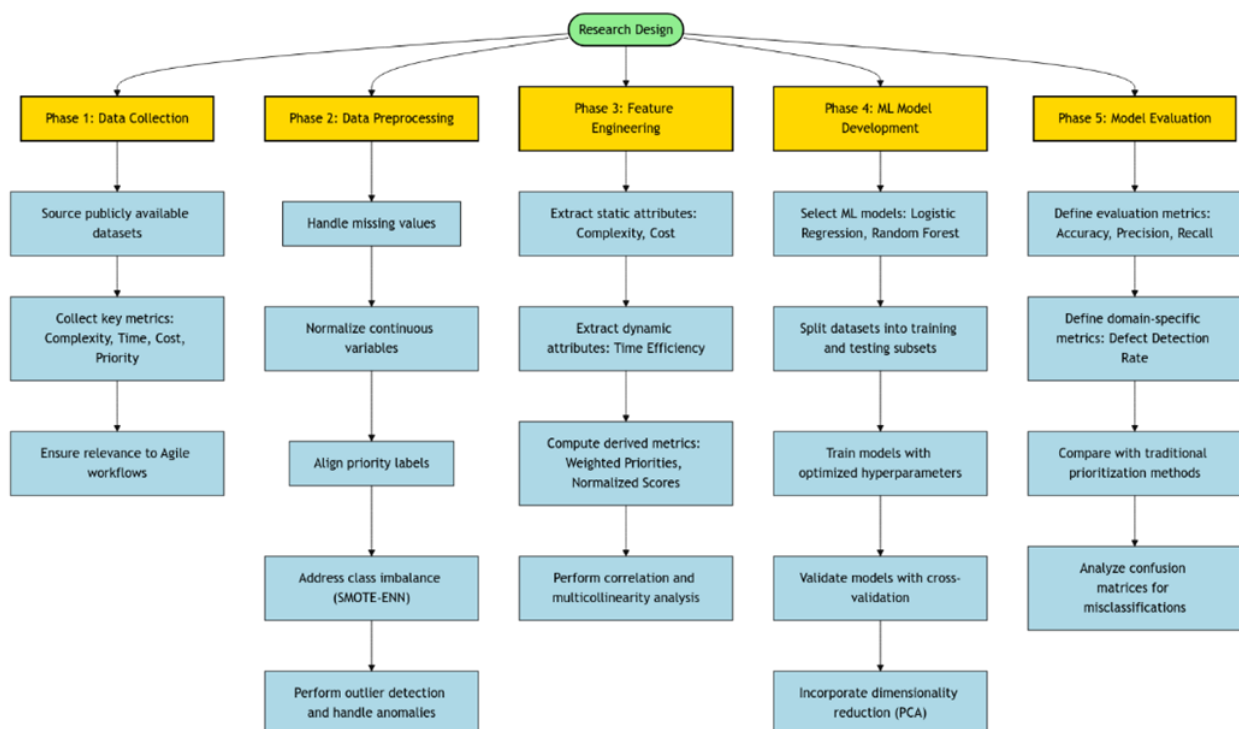


Figure 1. Illustration of the research design for agile TCP

In the third phase, static attributes, of which complexity and costs are examples, and dynamic attributes, such as time efficiency, can be used in feature engineering. This simulates the entire scoring pipeline with interaction terms (*Time_to_Cost*) and derived metrics like weighted priorities. This normalizes scores to improve feature relevance. Then, the correlation and multicollinearity analysis would narrow down the features by dropping redundant or other irrelevant features. In this fourth phase, the authors use Machine Learning (ML) algorithms like Logistic Regression, Random Forest and Gradient Boosting. Data are partitioned in training and test sets. Models are fitted with optimized hyperparameters. PCA based dimensional reduction retains important variance, while substantially decreasing the dimensionality of the dataset and cross-validation finds a robust and generalizable model.

At the final fifth phase, model performance is evaluated through accuracy, precision, recall, and specialized domain metrics such as defect detection rate. Comparative analysis with traditional prioritization methods (FIFO) would be performed to analyze the confusion matrices. The FIFO approach, representing the traditional prioritization strategy, was used as a baseline for comparison. Evaluation results show that the proposed model outperformed FIFO across all key metrics. MRR improved from 0.527 to 0.813, NDCG from 0.541 to 0.828, and APFD from 0.614 to 0.873, indicating a substantial gain in early fault detection efficiency and ranking accuracy. This explicitly present the comparison between the proposed model and the FIFO baseline in the evaluation section. FIFO was selected as a traditional prioritization benchmark, and the evaluation shows that the proposed model achieves stronger performance across the measured metrics of accuracy, precision, recall, and defect detection rate. These results indicate that the proposed model is more effective in detecting faults earlier and ranking test cases than FIFO. This detects the misclassifications to verify that the engineered model is correct.

This study focuses on solving a binary classification, so there are 2 outcome classes - Class 0 and Class 1. The Negative Class 0 (low-priority cases) achieved a precision of 0.97 and an f1-score of 0.97, while the Positive Class 1 (high-priority cases) achieved a precision of 0.99 and an f1-score of 0.99. This ensures clarity and consistency with the context of this research, while retaining the reported performance metrics. Hyperparameters improve ML models' accuracy immensely, the configuration was set before training the model. The k -fold cross-validation via the Grid search technique was used to find the best hyperparameter values tuning. With $cv = 5$, the training data was split into five distinct parts to ensure a reliable model evaluation and robust tuning. MRR, NDCG, and DDCE were identified during the design phase as additional evaluation metrics to provide a deeper analysis of prioritization effectiveness. The present evaluation focused on accuracy, precision, recall, and defective detection rate, which were sufficient to demonstrate the model's improvement over the FIFO baseline. Incorporating MRR, NDCG, and DDCE is planned for future work to provide further detailed prioritization insights.

Researching TCP requires data collection to get relevant, good quality information from which to formulate a predictive model for the purpose of constructing TCP in agile workflows. For this study, two datasets were selected, namely *CarFinTest* and *MISPriority*. The selected datasets are contextual to the domain of software testing and structured as required by research objectives. These data sets need to be processed separately, as they have their unique characteristics and need to maintain their integrity. The *CarFinTest* was created based on a car financing software development project and consists of around 2000 rows with attributes such as business requirements, requirement priority, function points, complexity, estimated time, cost, and where the priority level is labeled. These properties give a detailed view of the testing task along with metrics, useful in prioritizing testing tasks. The dataset itself reflects the complexities of testing in a larger software project, which is why it has been widely used for building predictive models that help with resource allocation and defect detection.

The *MISPriority* dataset contains 1314 rows and eight attributes from an industry project on Management Information System (MIS), such as business requirements, requirement priority, assigned priority weight, function point, complexity, estimated cost, time, and labeled priority levels. Apart from the common properties available in the *CarFinTest*, *MISPriority* further provides weight for each criteria giving an additional level of granularity for the analysis. Different preprocessing would be utilized for both datasets to preserve the structure and address the characteristics of them. In the case of *CarFinTest* dataset, preprocessing will be stricter by cleaning and normalizing the values of attributes such as complexity and cost, preparing it in a completely uniform way for labels business requirements and priority. Weights will be carefully considered, will not bias prioritization between attributes. However, for the *MISPriority* dataset, special attention will be needed to be managed accordingly. Such independent preprocessing helps retain information specific to each dataset enabling feature engineering and model building to be tailored to datasets more accurately.

Although these datasets have a high degree of relevance, they all have shortcomings. These datasets mainly contain static attributes (such as complexity and cost) with few dynamic metrics (defect detection history, test execution

data). If there is a class imbalance in the priority then it will also impact on the model performance, so it is important to look into this too while pre-processing the data. Since the datasets are collected from well-defined environments, the results should also be generalized with caution. To summarize, this research processes the *CarFinTest* and *MISPriority* datasets independently to uniquely address their different characteristics. The datasets will independently serve to inform feature engineering and inform the development of the model, while allowing for a comparative analysis of prioritization strategies in multiple contexts.

4. RESULTS AND DISCUSSIONS

The data preprocessing pipeline is an iterative and systematic framework designed specifically for the *CarFinTest* and *MISPriority* datasets for machine learning modeling. The engineering strategy starts with data collection as highlighted in Table 1. In this phase, datasets are collected from trusted sources, ensuring both completeness and quality. Preprocessing deals with missing values via imputation methods for all the attributes: mean/median where numeric and domain-specific for categorical attributes. The authors normalize the numeric features in the dataset, ensuring that all the features are within the same range to prevent large numbers disproportionately affecting the models. Priority levels and other similar categorical variables are encoded as numerical values with ordinal significance. To address class imbalance, the pipeline employs oversampling techniques like SMOTE or hybrid methods such as SMOTE-ENN, ensuring an even representation of priority levels for better generalization.

Table 1. Key Analytical Framework

Step	Description
Data Preprocessing	Cleansed datasets, handled missing values, and engineered meaningful features like time-to-cost ratios.
Class Balancing	Applied SMOTE-ENN to oversample minority classes and refine decision boundaries.
Dimensionality Reduction	Applied PCA to retain 95% variance while reducing dimensional noise.
Model Evaluation	Compared Logistic Regression, Random Forest, and Gradient Boosting using accuracy, precision, recall, and ROC-AUC.
Iterative Refinement	Added new features and tuned hyperparameters based on model performance feedback.

With the environment settings in Table 2, some dimensionality reduction by means of PCA eliminate the redundant features that contain little variance and reduces the risk of overfitting. After the data is refined, it is split into training and test sets and models are trained using algorithms such as Logistic Regression, Random Forest, Gradient Boosting. Maximizing the performance of the model is done through hyper-parameter tuning using advanced search techniques for parameters like learning rate and maximum depth. Models are assessed using metrics relevant to the task, such as accuracy, ROC-AUC, and F1-score, to confirm that they have learned systematically and are now suitable for predicting unseen samples. The pipeline is iteratively revisited to stages like feature engineering or balancing, if any shortfalls are identified. Specifically, the improved models yield actions and strategies such as TCP plans to optimize agile workflows, like increasing the proportion of defects caught earlier in the process or shifting a development focus to better utilize available resources during software development. This entire pipeline helps make sure that the data is accurate, consistent, and will make sense for the success of the research.

Table 2. Test Case Environment Settings to Engineer Predictive Model

Component	Details
Programming Language	Python 3.10
IDE	PyCharm Professional Edition
Libraries	pandas, numpy, scikit-learn, imbalanced-learn, matplotlib, seaborn
System Configuration	Windows 11, Intel Core i5, 8GB RAM, SSD Storage
Storage	Local CSV files for dataset storage and reproducibility

4.1 Model Performance and Enhancements Across Iterative Attempts

The authors consolidate the progression of five attempts to improve model performance on two datasets, *CarFinTest* and *MISPriority*, with a focus on addressing their distinct challenges. With the results in Table 3, this work can be insightful for tailored preprocessing and model choices for optimal predictive performance. In Table 3, Attempts 4 and 5 are undergone for *CarFinTest*. Meanwhile, *MISPriority* stops at Attempt 3. This implies *MISPriority* was already performing well, with a substantial gain in accuracy and ROC-AUC.

Table 3. Model Performance and Enhancements Across Iterative Attempts

Attempt	Dataset	Enhancements Applied	Best Model	Accuracy (Best Model)	ROC-AUC (Best Model)	Key Observations
1st	<i>CarFinTest</i>	Basic preprocessing and initial model training.	Decision Tree	34.00%	50.49%	All models struggled due to class imbalance and poor feature separability.
	<i>MISPriority</i>	Basic preprocessing and initial model training.	Logistic Regression	73.76%	77.12%	<i>MISPriority</i> dataset performed well, showing meaningful feature-target relationships and balanced class distributions.
2nd	<i>CarFinTest</i>	Hyperparameter tuning using advanced search techniques.	Logistic Regression	33.00%	50.00%	Dataset complexities and overlapping features limited improvement.
	<i>MISPriority</i>	Hyperparameter tuning using advanced search techniques.	Random Forest	76.43%	83.89%	Random Forest handled the dataset's complexity effectively, improving accuracy and class separability.
3rd	<i>CarFinTest</i>	SMOTE for balancing classes and additional feature engineering.	Random Forest	37.16%	52.84%	Improved preprocessing provided slight gains, but class misclassifications persisted.
	<i>MISPriority</i>	SMOTE for balancing classes and additional feature engineering.	Gradient Boosting	78.63%	84.73%	Gradient Boosting achieved the best performance, excelling in medium and high-priority classifications.
4th	<i>CarFinTest</i>	SMOTE-ENN for class balancing and interaction features (e.g., Time to Cost).	Random Forest	77.78%	83.89%	Significant improvement due to advanced preprocessing. Random Forest handled the refined dataset effectively.
5th	<i>CarFinTest</i>	Interaction features, PCA for dimensionality reduction, and SMOTE-ENN.	Random Forest & Gradient Boosting	77.78%	85.63% (Random Forest)	Both Random Forest and Gradient Boosting performed strongly, showcasing the effectiveness of dimensionality reduction.

The performance of the *MISPriority* dataset on first three attempts was stable, as it contained structured features and well-balanced class distributions, also Gradient Boosting was found to be best among them. On the other hand, *CarFinTest* was one of the most difficult datasets due to its inherent class imbalance, feature overlap and low separability, wherein the performance was ultimately subpar in majority of cases. The fourth and fifth attempts were significantly better after applying advanced preprocessing techniques to the data like SMOTE-ENN, Interaction features and PCA. The improvements enabled both Random Forest and Gradient Boosting to achieve high accuracy plus ROC-AUC scores. These results demonstrate their effectiveness in handling the complex relationships across the dataset. On the contrary, Logistic Regression performed quite poor in all attempts, as it did not manage to reproduce the fact that there is a clear non-linear relationship.

4.2 Reflections of Dashboard Features Deployed

The dashboard built and deployed through this research acts as a centralized, smart, ML-enabled interface for prioritization of test cases. Instead of developing and maintaining many distinct tools, two interactive dashboards support each dataset type separately (*CarFinTest* and *MISPriority*), which customized according to respective dataset. This design principle encourages scalability, maintainability, and steady user experience across dataset types. The User Interface (UI) is intentionally designed to abstract off technical complexities while having powerful functionality underneath the surface. The dashboard is easy to navigate and has a clear, step by step workflow from uploading the dataset to downloading results. In the upload section, users upload their datasets in CSV format. Once you upload the dataset, the dashboard can intelligently identify the dataset type, routing it to the appropriate internal processing path by looking at the structure and header patterns of the data. At the dataset summary and validation, basic dataset metadata such as row count, feature names, and missing value indicators are displayed. Validation logic checks for expected columns, appropriate data types, and completeness. The progress tracker provides a visual progress tracker outlines the major stages with preprocessing, training, and evaluation, together with updates in real-time as the workflow progresses. This gives users a transparent view of the pipeline execution.

In terms of automating the workflow and routing, the automated ML decision engine maps an uploaded dataset to its corresponding preprocessing and modeling flow. Although only one dashboard exists, it is designed to act conditionally based on the type of dataset uploaded. For the first capability, automated agile workflow switching uses a set of rules (e.g. expected column names, dataset schema) to distinguish between *CarFinTest* and *MISPriority* datasets. Once identified, it routes the data through its specific preprocessing logic. To support with shared preprocessing backbone, common tasks such as missing value handling, encoding, normalization, and class balancing using SMOTE-ENN are shared between dataset flows, implemented as reusable modules. Upon enhancing the two datasets of *CarFinTest* and *MISPriority*, the trained adaptive ML model selects the right algorithm, applies predefined hyperparameters, and proceeds with training. Evaluation follows using metrics like accuracy, F1-score, and ROC-AUC. These metrics can ensure clearer alignment with the objectives outlined in the literature review and introduction. The FIFO benchmark comparison addresses the baseline method towards the more advanced prioritization metrics such as MRR, NDCG, and DDCE. Their absence in the current evaluation is acknowledged as a limitation due to scope and dataset constraints. These metrics are highlighted as a focus for future work to extend the current findings and provide deeper performance insights.

Though the system is delivered as a unified dashboard, its internal logic is modularized by design. Each component (preprocessor, feature engineer, model trainer) is implemented as a standalone service to ensure the benefits, which include code reusability to share logic between dataset flows can be reused without duplication. To ease maintenance, the system updates or enhancements to individual steps do not disrupt the entire pipeline. For scalability, future datasets can be supported by simply registering new dataset-specific handlers. This structure allows the dashboard to behave like multiple tailored systems while physically remaining a single, cohesive platform. Although the current implementation supports only two datasets, the dashboard has been future-proofed to accommodate additional datasets and algorithms. Expansion can be achieved through plugin-like architecture. New datasets can be added by defining their preprocessing rules and model configurations in separate handler modules. To further expand the model, additional algorithms such as XGBoost, LightGBM, or neural networks can be integrated into the training pipeline. As such, the dashboard is compatible with flexible deployment on local machines, cloud services, or integrated into DevOps pipelines for CI/CD in testing agile workflows.

5. CONCLUSION

In conclusion, this work confirms the feasibility of using machine learning for TCP, bringing together theoretical approaches and practical agile testing processes. This work not only illustrates automated prioritization is possible, but it also lays the groundwork for future work in artificial intelligence (AI) driven defect prediction, intelligent test ranking, and ultimately automated software testing tools. The goal of this research is to engineer a model for TCP, which was predictive and could use machine learning in agile software testing workflows. This work utilized two interactive dashboards. Through machine learning methods, the prioritization process was automated successfully in this research, preprocessing datasets, feature engineering, model training and prioritization. These dashboards offer an intuitive, data-driven approach where users can upload the raw data and get a priority-optimized output without human intervention at all. The integration of dynamic preprocessing pipelines, visualization tools, and predictive modeling ensures that agile teams can optimize test execution sequences based on real-time insights.

The comparative evaluation of multiple classification models revealed significant variations in dataset complexities. MISPriority shows impressive results across all models with high predictive score and it consistently overshadows CarFinTest dataset in terms of feature-target relationship as well as accuracy and ROC-AUC rates through models like Gradient Boosting or Random Forest. On the other hand, CarFinTest made significant difficulties with class imbalance, features redundancy and low-feature separability. While the initial models performed poorly, applying advanced preprocessing techniques such as SMOTE-ENN balancing, interaction-based feature engineering and PCA reflected a substantial increase in the models' predictive power. By the final iteration, both Random Forest and Gradient boosting gave the best results and confirmed that with the right preprocessing and feature transformation it can make solid gains even for complex data.

The contribution of this paper is to acknowledge the benefits of automated machine learning-based prioritization of test cases in software testing. The dashboards created are free from manual processing and provide flexibility to dynamically rank test cases through predictive analytics. Also, the comparative analysis on various machine learning models point towards the usefulness of ensemble-based techniques such as Random Forest and Gradient Boosting in structured datasets. Moreover, their appropriate combination with efficient preprocessing techniques reassures the effectiveness of data generation methods in terms of improving the generalizability of the model and handling the scenarios of real test data. In addition, the dashboard design follows scalable and modular architecture, which is adaptable to future extensions like new datasets and more sophistication in data prioritization.

The research faced many hurdles, despite its success. Inherent complexities of the *CarFinTest* dataset showed poor performance of models with a significant number of optimizations attempts to arrive at meaningful prediction. A further limitation is that the evaluation was conducted solely on two structured datasets, MISPriority and CarFinTest, both originating from controlled environments. Another limitation is that the developed dashboards were evaluated solely in a research context and have not yet undergone validation through external user feedback or deployment within a live Continuous Integration/Continuous Deployment (CI/CD) pipeline. This choice was deliberate, as the primary objective was to demonstrate the technical feasibility of the predictive prioritization approach rather than to conduct field trials. Future work can focus on integrating the system into real-world Agile workflows, collecting feedback from practitioners, and assessing usability, performance, and fault detection benefits in operational environments. While this choice ensured methodological consistency, it constrains the generalizability of the findings to more diverse real-world datasets with varied structures, unstructured elements, and noise characteristics.

Difference in performance across datasets also suggested that generalizing the model is not possible without tuning for a specific dataset. The computational cost incurred from feature engineering, SMOTE-ENN balancing, PCA, and iterative hyperparameter tuning posed a challenge for real-time deployment, particularly in Agile environments where rapid prioritization and deployment cycles are critical. Future work could explore lighter-weight preprocessing techniques, incremental learning, or model optimization strategies (pruning, dimensionality reduction through feature selection instead of PCA) to reduce execution time while maintaining accuracy. Dynamic prioritizing based on real-time data streams (live execution logs, defect tracking metrics) is another dimension to which the process can further be improved.

Nevertheless, this study did not implement or evaluate incremental or streaming learning approaches, which are essential for real-time prioritization in agile contexts. Moreover, while this study employed widely recognized classification metrics such as accuracy, ROC-AUC, precision, recall, and F1-score to evaluate model performance, it did not incorporate prioritization-specific evaluation measures such as Mean Reciprocal Rank (MRR) or Defect Detection Cost Efficiency (DDCE). The inclusion of such metrics in future work could provide a more

comprehensive assessment of the effectiveness of the prioritization approach, particularly in real-world testing environments where the order of test execution directly impacts fault detection efficiency. Future work could explore online learning algorithms (e.g., Hoeffding Trees, adaptive boosting) to enable continuous model updates as new data arrives, ensuring the prioritization adapts dynamically to evolving project conditions. Real-time or incremental learning models were beyond this research scope due to its emphasis on demonstrating technical feasibility in a controlled environment. Future iterations should explore these approaches to support agile, continuously evolving data streams.

In addition, utilizing more advanced feature engineering methodologies such as deep learning or natural language processing (NLP)-based techniques could better predict for TCP. This research intentionally focused on two structured datasets to maintain methodological consistency and control over variable factors. However, this limits generalizability to more diverse real-world contexts. Future work can incorporate datasets from multiple domains to validate broader applicability. For deployment in real-world applications, the dashboard would need to be hosted to make it accessible to the users through a cloud hosting on platforms like Amazon Web Services (AWS), Heroku, Streamlit Cloud for this requirement.

ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their valuable comments.

FUNDING STATEMENT

The authors received no funding from any party for the research and publication of this article.

AUTHOR CONTRIBUTIONS

Kavinraj Baskaran: Formal Analysis, Investigation, Conceptualization, Data Curation, Methodology, Validation, Visualization, Resources, Writing – Original Draft Preparation;

Junaidi Abdullah: Project Administration, Supervision, Writing – Original Draft Preparation, Review & Editing;

Sin-Ban Ho: Conceptualization, Validation, Visualization, Resources, Supervision, Writing – Original Draft Preparation;

Said Suleiman Naseer Al Harthy: Validation, Visualization, Resources, Writing – Original Draft Preparation;

Ian Chai: Project Administration, Supervision, Writing – Original Draft Preparation, Review & Editing

CONFLICT OF INTERESTS

No conflict of interests was disclosed.

ETHICS STATEMENTS

The two datasets (*CarFinTest* and *MISPriority*) used for this study were obtained from Kaggle.

DATA AVAILABILITY

Derived data supporting the findings of this study are available from the corresponding author on request.

REFERENCES

- [1] I. Ibriwesh, S.-B. Ho, I. Chai, and C.-H. Tan, "A controlled experiment on comparison of data perspectives for software requirements documentation," *Arabian Journal for Science and Engineering*, vol. 42, no. 8, pp. 3175–3189, Feb. 2017, doi: 10.1007/s13369-017-2425-2.
- [2] D. Marijan, "Comparative study of machine learning test case prioritization for continuous integration testing," *Software Quality Journal*, vol. 31, no. 4, pp. 1415–1438, Dec. 2023, doi: 10.1007/s11219-023-09646-0.
- [3] M. Mahdich, S. H. Mirian-Hosseinabadi, and M. Mahdich, "Test case prioritization using test case diversification and fault-proneness estimations," *Automated Software Engineering*, vol. 29, no. 2, Nov. 2022, doi: 10.1007/s10515-022-00344-y.
- [4] H. Salin and Y. Rybarczyk, "Decision-Making in Agile Software Engineering: A Systematic Literature Review of Models, Methods, Actors and Lifecycle Contexts," *Software*, vol. 5, no. 2, p. 17, Apr. 2026, doi: 10.3390/software5020017.
- [5] N. Tasneem, H. B. Zulzalil, and S. Hassan, "Enhancing Agile Software Development: A Systematic Literature Review of Requirement Prioritization and Reprioritization Techniques," *IEEE Access*, vol. 13, pp. 32993–33034, 2025, doi: 10.1109/ACCESS.2025.3539357.
- [6] P. G. Obike, V. E. Ekong, and O. U. Obot, "A Novel Scoring-based Agile Sprint Deliverability Prediction and Prioritization using Machine Learning," *IJCA*, vol. 187, no. 49, pp. 40–53, Oct. 2025, doi: 10.5120/ijca20259258.
- [7] N. A. Goyal, "Driving Continuous Improvement in Engineering Projects with AI-Enhanced Agile Testing and Machine Learning," *International Journal of Advanced Research in Science Communication and Technology*, pp. 1320–1331, Jul. 2023, doi: 10.48175/ijarsct-14000t.
- [8] S. Hudda, "Prioritization of Product Backlog Items in Agile Development Using Rank Ordering and Weighting Method," in *Complex, Intelligent and Software Intensive Systems*, vol. 260, L. Barolli, T. Enokido, and I. Woungang, Eds., Cham: Springer Nature Switzerland, 2025, pp. 218–230. doi: 10.1007/978-3-031-96099-4_21.
- [9] A. M. Radwan, M. A. Abdel-Fattah, and W. Mohamed, "AI-Driven Prioritization Techniques of Requirements in Agile Methodologies: A Systematic Literature Review," *ijacsa*, vol. 15, no. 9, 2024, doi: 10.14569/IJACSA.2024.0150983.
- [10] J. C. Pang, F. Y. Lee, L. H. Chin, P. L. Ng, and S. Y. Te, "E-Learning Instructional Management For Students with Intellectual Disabilities," *IJSE*, vol. 39, no. 1, pp. 1–10, Apr. 2024, doi: 10.52291/ijse.2024.39.1.
- [11] J. C. Pang, F. Y. Lee, L. H. Chin, P. L. Ng, and S. Y. Te, "Developing Work Aspiration in Students with Learning Disabilities in Malaysia," *APJEE*, vol. 40, no. 1, 299–316 (2025). doi: 10.21315/apjee2025.40.1.15.
- [12] J. R. Romero, A. Ramírez, and C. García-Martínez, "Automated machine learning for test case prioritisation," *Empir Software Eng*, vol. 31, no. 5, p. 120, Sep. 2026, doi: 10.1007/s10664-026-10862-y.
- [13] A. Zarrad, T. Armstrong, and J. Jemai, "A Hybrid Approach to Software Testing Efficiency: Stacked Ensembles and Deep Q-Learning for Test Case Prioritization and Ranking," *CMC*, vol. 0, no. 0, pp. 1–10, 2025, doi: 10.32604/cmc.2025.072768.
- [14] R. Haque, S.-B. Ho, I. Chai, and A. Abdullah, "Parameter and hyperparameter optimisation of deep neural network model for personalised predictions of asthma," *Journal of Advances in Information Technology*, vol. 13, no. 5, Jan. 2022, doi: 10.12720/jait.13.5.512-517.
- [15] F. Al Farid, A. Bari, A. S. M. Miah, S. Mansor, J. Uddin, and S. P. Kumaresan, "A Structured and Methodological Review on Multi-View Human Activity Recognition for Ambient Assisted Living," *J. Imaging*, vol. 11, no. 6, p. 182, Jun. 2025, doi: 10.3390/jimaging11060182.
- [16] A. Karatayev, A. Ogorodova, and P. Shamo, "Fuzzy Inference System for Test Case Prioritization in Software Testing," 2024, *arXiv*. doi: 10.48550/ARXIV.2404.16395.
- [17] H. Kumar and V. Saxena, "Machine Learning for Test Case Prioritization in Continuous Integration: A Comprehensive Analysis," *IJANA*, vol. 17, no. 02, pp. 6218–6228, 2024, doi: 10.35444/IJANA.2024.15609.
- [18] S. Sowmyadevi, and S. Mehrotra, "Deep-TCP: An Intelligent Deep Learning Model for Automated Test Case Prioritization Systems using Neural Networks," *IJETT*, vol. 74, no. 6, pp. 294–308, Jul. 2026, doi: 10.14445/22315381/IJETT-V74I6P121.
- [19] S. Al-Juboori, J. Pua, S.-C. Haw, L. D. Krisnawati, and G.-K. Tong, "Ontology-Based E-Commerce Recommender System: A Hybrid Semantic Filtering Approach," *JiWE*, vol. 1, no. 5, p. 86, Feb. 2026, doi: 10.33093/jiwe.2026.5.1.6.
- [20] K. Garg and S. Shekhar, "Test case prioritization based on fault sensitivity analysis using ranked NSGA-2," *Int. j. inf. tecnol.*, vol. 16, no. 5, pp. 2875–2881, Jun. 2024, doi: 10.1007/s41870-024-01868-0.

- [21] Z. Sakhravi and T. Labidi, "Test case selection and prioritization approach for automated regression testing using ontology and COSMIC measurement," *Autom Softw Eng*, vol. 31, no. 2, p. 51, Nov. 2024, doi: 10.1007/s10515-024-00447-8.
- [22] R. Cheng, S. Wang, R. Jabbarvand, and D. Marinov, "Revisiting Test-Case Prioritization on Long-Running Test Suites," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, Vienna Austria: ACM, Sep. 2024, pp. 615–627. doi: 10.1145/3650212.3680307.
- [23] T. Mattis, L. Böhme, E. Krebs, M. C. Rinard, and R. Hirschfeld, "Faster Feedback with AI? A Test Prioritization Study," in *Companion Proceedings of the 8th International Conference on the Art, Science, and Engineering of Programming*, Lund Sweden: ACM, Mar. 2024, pp. 32–40. doi: 10.1145/3660829.3660837.
- [24] I. Ibriwesh, S.-B. Ho, I. Chai, and C.-H. Tan, "Prioritizing solution-oriented software requirements using the multiple perspective prioritization technique algorithm: An empirical investigation," *Concurrent Engineering*, vol. 27, no. 1, pp. 68–79, Mar. 2019, doi: 10.1177/1063293X18808559.
- [25] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review," 2021, doi: 10.48550/ARXIV.2106.13891.
- [26] Y. Zhao, D. Hao, and L. Zhang, "Revisiting Machine Learning based Test Case Prioritization for Continuous Integration," 2023, *arXiv*. doi: 10.48550/ARXIV.2311.13413.
- [27] G. Rao and D. Nandal, "Dynamic prioritization of test cases for regression testing using machine learning," *Int J Syst Assur Eng Manag*, Feb. 2026, doi: 10.1007/s13198-026-03144-9.
- [28] H. A. Ramzan, K. Islam, M. A. Hussain, R. M. Monim, S. M. Asad, and S. Ramzan, "Data-Driven Test Case Prioritization (DD-TCP): A Machine Learning Framework for Intelligent Software Quality Assurance," *Computers, Materials & Continua (CMC)*, pp. 1–10, 2026, doi: 10.32604/cmc.2026.077782.
- [29] C.-W. Teoh, S.-B. Ho, K. S. Dollmat, and I. Chai, "An Evolutionary Algorithm-Based Optimization Ensemble Learning Model for Predicting Academic Performance," in *2022 11th International Conference on Software and Computer Applications*, Melaka Malaysia: ACM, Feb. 2022, pp. 102–107. doi: 10.1145/3524304.3524320.
- [30] R. Haque *et al.*, "Intelligent Health Informatics with Personalisation in Weather-Based Healthcare Using Machine Learning," in *Innovative Systems for Intelligent Health Informatics*, vol. 72, F. Saeed, F. Mohammed, and A. Al-Nahari, Eds., Cham: Springer International Publishing, 2021, pp. 29–40. doi: 10.1007/978-3-030-70713-2_4.

BIOGRAPHIES OF AUTHORS

	Kavinraj Baskaran a Product and QA Specialist in the software industry with quality assurance, data-driven decision-making, and product development roles. He obtained the Bachelor and Masters in Computer Science from Multimedia University, Cyberjaya, Malaysia. His research explores machine learning, software engineering, and computer science education, enabling him to effectively address complex real-world challenges through scholarly innovation and practical expertise. He can be contacted at email: kavinraj701@gmail.com.
	Junaidi Abdullah is a Professor and teaches Computer Science in Multimedia University. He received the Bachelor in Engineering (First Class Honours) from the University of Bristol, UK, and a Ph.D. in Computer Science Computer Vision and Augmented Reality from the University of Southampton, UK, in 2005. His research interests include augmented reality, computer vision, image and video processing. He has published more than 60 internationally multi-disciplinary refereed conference, journal articles, and books. He can be contacted at email: junaidi.abdullah@mmu.edu.my.
	Sin-Ban Ho is an Associate Professor and teaches Computer Science in Multimedia University. He received his PhD degree in information technology from Multimedia University. He received the Bachelor in Computer Science (First Class Honours) and Masters of Computer Science from University Science of Malaysia. His research interests include health informatics, empirical research approach, computer education, and patterns. He is a senior member of the IEEE, IEEE Computer Society, and IEEE Education Society. He can be contacted at email: sbho@mmu.edu.my.
	Said Suleiman Naseer Al Harthy is the Director of Information and Communication Technology at Modern College of Business and Science (MCBS), Muscat, Oman. He received his master's in Information Technology from University of Wolverhampton, England, United Kingdom, and a Bachelor's in Computer Applications from Bangalore University, India. His research interests include ERP optimization and emerging technologies in higher education. He can be contacted at email: said@mcbs.edu.om.
	Ian Chai is an Associate Profesor in Multimedia University. He received his PhD degree in computer science from the University of Illinois at Urbana-Champaign, USA, and teaches Computer Science at Multimedia University, Cyberjaya, Malaysia. He has worked in Connecticut and Germany. His research interests include computer science education, design patterns, health informatics, and open-sourced software. He can be contacted at email: ianchai@mmu.edu.my.