Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

A Hybrid Machine Learning and Six Sigma Framework for Software Defect Prediction

Atul Gaudbole[1*], Soniya Sharma[2]
[1] Department of Computer Engineering, University of Delhi, Delhi, India.
[2] Department of Cyber Security, University of Dubai, Dubai, United Arab Emirates.
*Corresponding Author:
atul12gaudbole@gmail.com

Article History:

***Received**: 07-04-2026*
***Revised:** 26-05-2026*
***Accepted**: 27-06-2026*

DOI:
https://doi.org/10.xxxx/jsst.xxxx

Article Type:
Research Article

Abstract

Software defect prediction (SDP) is a proactive approach that assists in identifying the modules which are prone to fault early enough so that the remediation and the use of the testing resources can be used in the most optimum manner. Machine Learning (ML) models have demonstrated significant predictions in the area, but in general, are applied without the use of well-established process improvement models; therefore, their role in quality control on a long-term basis is quite insignificant. The gap in this research is bridged by proposing the new hybrid framework which would integrate the predictive power of the Machine Learning models with the statistical method of the Six Sigma i.e. Define, Measure, Analyze, Improve, Control (DMAIC) approach. Here this research has empirically compared four classifiers, viz, Logistic Regression, Decision Tree, Random Forest, and Support Vector machine, on NASA MDP KC1 data. Random Forest model became the most excellent with an accuracy of 91.25 and Receiver Operating Characteristic-Area Under the Curve (ROC-AUC) of 0.964. The research work has found that cooperation between the ML based prediction and the statistical evaluation of Six Sigma provides a powerful data-driven paradigm to not only identify defects but also measure and maintain software quality improvement. The research work concludes with contributions to SDGs.

Keywords: Software Defect Prediction, Six Sigma, Machine Learning, Statistical Quality Control, Process Capability, DMAIC, Random Forest.

Sustainable Development Goal (SDG) Contributions:

Primary SDG: SDG 9 - Industry, Innovation, and Infrastructure.

Secondary SDGs: SDG 8 - Decent Work and Economic Growth; SDG 12 - Responsible Consumption and Production.

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

1. INTRODUCTION

The quality of software is among the largest issues about software engineering since the hidden problems can potentially result in significant financial damage, potential security breaches, and the loss of the confidence of the users. It has been estimated that the cost of correcting a defect grows exponentially, later it was identified in the development cycle and this is how early identification of a defect is a very crucial economic and technical activity [1]. In this regard, SDP has emerged as one of the key methods that utilize past data and machine learning to actively recognize fault-prone modules, allowing ample capacity to properly test and allocate resources [2]. Although machine learning models have demonstrated tremendous potential in defective module classification, much attention in the literature has focused on optimization of predictive accuracy, without much effort to combine these predictions in a holistic process improvement model [3]. Numerous models are developed as isolated decision-support systems, and there is frequently no apparent means of measuring how much the output of the model affects the quality and capability of the overall software development process. This introduces a distance between the forecast and the quantitative process enhancement.

Six Sigma, with its structured DMAIC methodology supported by statistically sound tools, has established a sound basis for the reduction of variation and attainment of near-zero defect levels in both manufacturing and service industries [4]. However, this has often prohibited the application of Six Sigma in software engineering due to its lack of integration with modern data-driven predictive capabilities. The traditional Six Sigma analysis is reactive; combining it with machine learning makes it proactive [5]. Although the first steps in researching this synergy have started with works like [6], there is still a lack of any deep framework that would closely integrate ML prediction with the quantitative process metrics of Six Sigma, such as DPMO and Process Capability Indices Cp/Cpk.

This gap is attempted to be addressed in the research work by suggesting a new hybrid framework to defect prediction by merging machine learning models with the statistical power of Six Sigma. The large contributions of this work are three-fold:

1. The research initially gives a comparative empirical analysis of four ML classifiers, which include; Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine, based on the most commonly used NASA MDP KC1 dataset and hence, provide a solid foundation on defect prediction.
2. This goes beyond conventional performance measures of accuracy and AUC-ROC to determine fundamental quality measures of Six Sigma. This determines the level of contributions of the DPMO and Sigma Level of the development process using the predictions of the model and present the measurable enhancement of the capabilities of the process.

It was found that this approach is not only effective in prediction, whereby the Random Forest model has been found to be accurate at 91.25%, but also in process improvement where an actual increase in the sigma level was noticed. This research work concludes that the integration of ML and Six Sigma presents a more holistic, and

data-driven paradigm of software quality management whereby an independent prediction is replaced with a sustained, statistically tested enhancement.

The remaining research work is structured in the following way: Section 2 elaborates on the corresponding literature review. Part 3 expounds on the suggested methodology. Section 4 of the study provides the results of the experiments and discussion. The future research directions will be presented in Section 5. Section 6 presented the contributions to SDGs and the research work will be concluded in Section 7.

2. LITERATURE REVIEW

The foundation of this research rests on two well-established domains: Machine Learning for SDP and the Six Sigma methodology for process improvement. The related work is thus organized to survey key contributions in these areas and their nascent integration.

A. Machine Learning for Software Defect Prediction

The foundation of this study talks about two well-established domains- Machine Learning for Software Defect Prediction i.e, SDP and the Six Sigma methodology for process improvement. The related work section is therefore structured to review the key contributions in these areas and their emerging integration.

For decades, empirical software engineering has concentrated on using machine learning to predict defect-prone software modules. Early research by [2] revealed that combining static code attributes with classifiers such as Logistic Regression and Naive Bayes effectively predicted defects in NASA datasets, establishing an early benchmark for subsequent studies. Later successful studies done, such as [3], expanded this comparison to a broad range of classifiers. Studies found out that no single algorithm consistently dominates but methods like Random Forest proved to perform robustly across multiple datasets.

The shift in single-model classifiers to ensemble approaches has been one of the biggest contributions to the SDP research. In their initial analysis, Hall et al. (2012) observed that ensemble models, especially the Random Forest, always performed better than single models such as the Decision Trees as they minimized variations and reduced overfitting [7]. This finding is consistent with larger machine learning literature and supported by the recent works that make use of boosting algorithms like XGBoost and LightGBM to reach the state-of-the-art performance [8]. This result can be compared with the broader literature in ML and additional evidence by later studies that use boosting models such as XGBoost and LightGBM to achieve state-of-the-art results [8].

However, SDP still has the challenge of class imbalance with the non-defective modules far exceeding the defective modules. In order to address this, publics like the Synthetic Minority Over-sampling Technique (SMOTE) are also common as observed in yours to minimize bias in favor of the majority group [9].

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

B. Six Sigma in Software Engineering

Six Sigma is an evidence-based methodology, which is based on the DMAIC (Define, Measure, Analyze, Improve, Control) paradigm, which was initially meant to introduce changes in the manufacturing processes. It has been investigated in the software engineering field as a way of decreasing defects and enhancing process capability. Rudimentary work, including the one by [5], has proposed that it be used with maturity models like CMMI and has also talked of how the statistical techniques applied in it can bring rigor to software process improvement projects.

The most important strength of the Six Sigma is its quantitative measures which it applies to measure the performance of processes. Defects Per Million Opportunities (DPMO) and the resultant Sigma Level metrics are used to present a standardized picture of quality, and Process Capability Indices (Cp, Cpk) are used to present a more detailed picture of how well a process is functioning to its natural variation [10]. To compute these indicators, it is important to have a clear definition of what is considered as a defect and an opportunity of a defect, and this has also been an adjustment point of software applications.

C. The Integration of ML and Six Sigma

It seems to make perfect sense to explore the logical intersection of models intended mostly for prediction, like the field of ML models, and the prescriptive, improvement-oriented Six Sigma framework. Although, works that have charted this territory so far have barely broken ground; for example, [6] introduce a conceptual framework for using ML's predictions of software quality to drive Six Sigma metrics design. Hence, works in this region show the way towards us.

However, critical analysis of the literature reveals one important shortfall. Most studies, such as [6], [11], tend to treat ML and Six Sigma as parallel or sequential activities rather than an integrated, closed-loop system. While the ML model predicts possible problematic modules in a one-off fashion, the subsequent step to use these predictions to compute Six Sigma metrics and then use those metrics to quantitatively demonstrate and control process improvement is omitted or not rigorously formalized within the DMAIC cycle. Such an omission generates a gap between identifying potential defects and quantifying holistic process capability enhancement.

D. Research Gap and Our Contribution

The existing literature indicates a clear gap in the present body of research, represented by the absence of a broad empirically validated framework, which would help integrate ML-based defect prediction with Six Sigma's statistical process control metrics for setting a continuous quality improvement loop.

Most of the available literature on SDP is heavily centered on predictive accuracy, without considering how well these predictions are translated into actionable process improvement metrics. While as, most Six Sigma applications in software lack the deep, proactive predictive power that ML models can provide. The current study bridges this gap by proposing and implementing a hybrid framework where:

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

1. A variety of ML models, namely Logistic Regression, Decision Tree, Random Forest, and SVM, are intensively tested on the NASA KC1 dataset for defect prediction.
2. Model outputs are used as inputs to calculate some of the key Six Sigma metrics: DPMO and Sigma Level.
3. The whole workflow is aligned with the DMAIC methodology and provides a structured blueprint for transitioning from data-driven prediction to statistically grounded process control and continuous improvement.

This integration will allow the study to go beyond mere prediction and quantify evidence of software process improvement.

3. PROPOSED METHODOLOGY

This section describes the proposed hybrid framework for software defect prediction and process improvement. The methodological underpinning of the framework is based on two important components: a machine learning pipeline for defect prediction, and a Six Sigma statistical module for process capability analysis. These are both incorporated into the DMAIC (Define, Measure, Analyze, Improve, Control) cycle.

A. Integrated Framework Overview

The proposed system will be a closed-loop process where the ML predictions are informed and further validated through the quantitative quality metrics of Six Sigma. Workflow: data collection/preprocessing; model training and analysis; using the model's predictions to inform quality improvement actions; and applying the Six Sigma metrics for continuous monitoring and control, as shown in Figure 1.

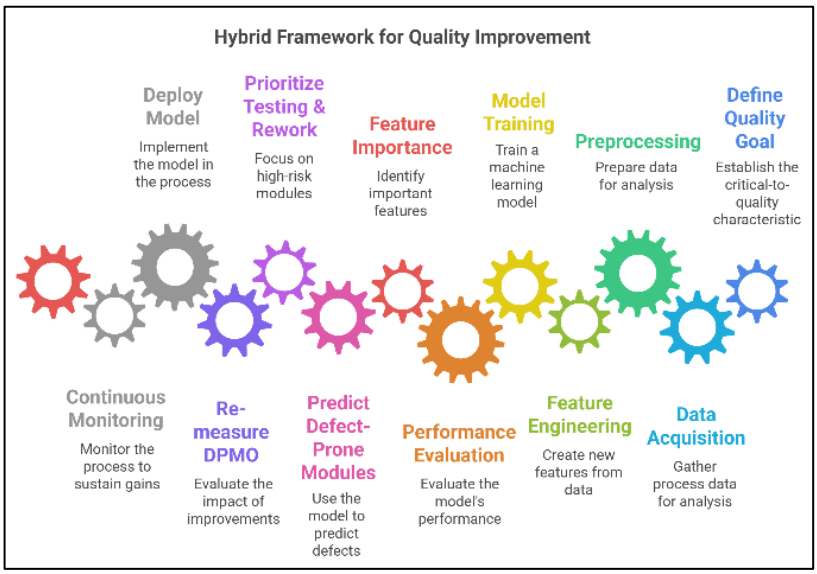

Figure 1. The integrated ML and Six Sigma DMAIC framework for software defect prediction and process improvement.

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

B. Dataset and Preprocessing

Dataset Description:

This research work uses the most widely used benchmark dataset for empirical validation of research on software defect prediction, namely, the NASA MDP KC1 dataset [2]. This dataset, which was gathered from a NASA project, comprises 2,096 software modules that are linked to 21 static code metrics and a binary defect designation that specifies whether a module is defective ('Y') or non-defective ('N').

Table 1. Categories of Software Metrics in the NASA KC1 Dataset

Category	Example Metrics
Size & Volume	LOC_TOTAL, LOC_EXECUTABLE, LOC_BLANK
Complexity	CYCLOMATIC_COMPLEXITY, DESIGN_COMPLEXITY, ESSENTIAL_COMPLEXITY
Halstead Measures	HALSTEAD_VOLUME, HALSTEAD_EFFORT, HALSTEAD_DIFFICULTY
Coupling & Operators	NUM_OPERANDS, NUM_OPERATORS, NUM_UNIQUE_OPERANDS

These metrics capture a broad spectrum of software characteristics concerning size, complexity, and structure summarized in Table 1. The original dataset suffers from class imbalance, with only 15.5% of modules labeled as defective. In addition, the distribution of important metrics such as LOC_TOTAL and CYCLOMATIC_COMPLEXITY is right-skewed, which indicates that there are just a few large, highly complex modules, common in many real-world software projects.

***Data Preprocessing Pipeline**:*

To ensure data quality and model robustness stays consistent a rigorous preprocessing pipeline was made to put in place.

- Data Cleansing and Imputation: Features were tried to change to numeric types. Confirmation that no missing values existed was made, but an imputation strategy using the median was at hand.
- Feature Scaling: StandardScaler was used to scale all the numeric features so that their mean is zero and the standard deviation is one. This step is important for a distance-based algorithm such as SVM and hence is needed to stabilize the training of Logistic Regression.
- Class Imbalance Handling: The Synthetic Minority Over-Sampling Technique (SMOTE) [9] was utilized on the training data to develop synthetic examples of the minority class of interest, namely defective modules. This resulted in a perfectly balanced training set that prevented the models from being biased toward predicting the majority class.

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

Then, the preprocessed dataset was split into a training set consisting of 80% and a hold-out test set consisting of 20% for final evaluation.

C. Machine Learning Models for Defect Prediction

Four different machine learning classifiers were chosen to span a range of algorithmic approaches from very simple linear models through to complex ensembles. This allows for a robust comparative analysis. The models are implemented using the scikit-learn library [12] and are outlined in Table 2.

Table 2. Machine Learning Models Implemented for Defect Prediction

Model	Type	Key Hyperparameters	Rationale
Logistic Regression (LR)	Linear	max_iter=1000, random_state=42	Serves as a interpretable baseline model.
Decision Tree (DT)	Non-linear	random_state=42	Provides a simple, non-linear benchmark and feature insights.
Random Forest (RF)	Ensemble	n_estimators=100, random_state=42	Expected high performer due to robustness to noise and overfitting.
Support Vector Machine (SVM)	Kernel-based	kernel='rbf', probability=True, random_state=42	Effective in high-dimensional spaces; uses RBF kernel for non-linearity.

Each model was trained on the balanced training set. The hyperparameter values were kept at their default values according to scikit-learn, in order to ensure a fair and reproducible comparison focused on the out-of-the-box performance of these algorithms in the proposed framework.

D. Six Sigma Statistical Evaluation

The key novelty of the present work is translating ML predictions into standard Six Sigma quality metrics, thereby quantifying process improvement.

Defects Per Million Opportunities (DPMO) and Sigma Level:

DPMO measures the defect density normalized to a million opportunities. In the software context we define one "opportunity" for a defect as one software module. DPMO and the corresponding Sigma Level are calculated as follows:

$$DPMO = \frac{Number\ of\ Defective\ Modules}{Total\ Number\ of\ Modules \times Opportunities\ per\ Unit} \times 10^6 \quad (1)$$

$$Sigma\ Level = \Phi^{-1}\left(1 - \frac{DPMO}{10^6}\right) + 1.5 \quad (2)$$

where, Φ^{-1} is the inverse cumulative distribution function of the standard normal distribution, and the 1.5 shift accounts for long-term process drift [10].

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

Process Capability Indices (Cp, Cpk):

Process capability indices quantify the degree to which a process meets specifications. In the context of a software development process, we define the specification limits on the basis of a target defect density. Let USL be the maximum acceptable defect rate, then Cp and Cpk can be computed as:

$$C_p = \frac{USL-LSL}{6\sigma} \quad (3)$$

$$C_{pk} = \min\left(\frac{USL-\mu}{3\sigma}, \frac{\mu-LSL}{3\sigma}\right) \quad (4)$$

where, μ is the process mean (observed defect rate) and σ is the process standard deviation. A C_{pk} value greater than 1.33 is generally considered capable.

4. RESULTS AND DISCUSSION

A. Machine Learning Model Performance

To evaluate how well the machine learning models performed, four different classifiers were tested on a separate set of data that wasn't used during training. Table 3 shows the results for all key metrics, and Figure 2 gives a visual comparison of their performance using ROC curves

Table 3. Comparative Performance of Machine Learning Models

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Logistic Regression	73.20%	0.73	0.73	0.73	0.817
Decision tree	87.73%	0.88	0.88	0.88	0.877
Random Forest	91.25%	0.91	0.91	0.91	0.964
Support Vector Machine	72.64%	0.73	0.73	0.73	0.813

From these results, it's clear that the Random Forest model outperformed all the others across every metric. It achieved the highest accuracy (91.25%), a balanced precision and recall of 0.91, and a strong ROC-AUC of 0.964. This means it was very good at distinguishing between defective and non-defective modules. Also, the Decision Tree model also did well but wasn't quite as accurate. The difference highlights how the Random Forest's combination of multiple trees (and its randomness in feature selection) helps avoid overfitting and improves general performance. The linear models, like Logistic Regression and SVM, worked as decent baselines, but they weren't as effective at capturing the complex, non-linear relationships in software metrics.

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

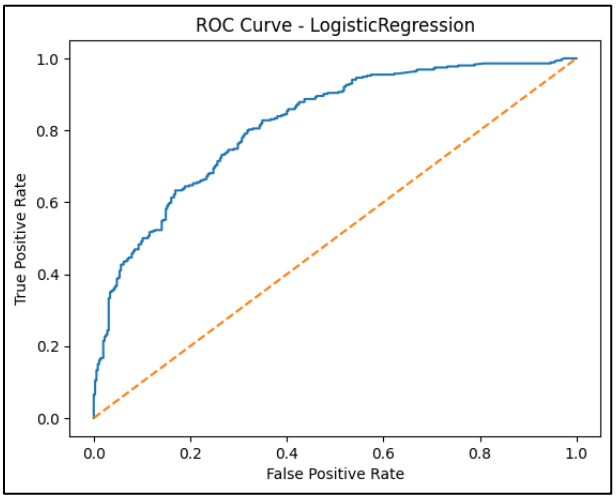

Figure 2. Logistic Regression shows moderate performance with 73.2% accuracy and fair class separation (ROC-AUC 0.817).

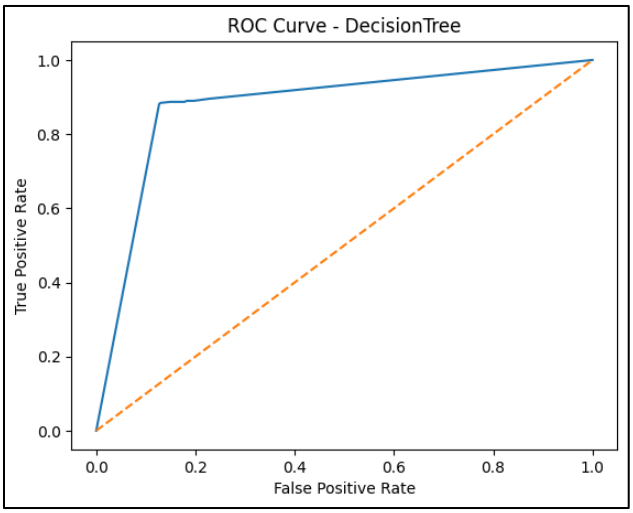

Figure 3. Decision Tree achieves strong predictive performance with 87.7% accuracy and improved class discrimination (ROC-AUC 0.877).

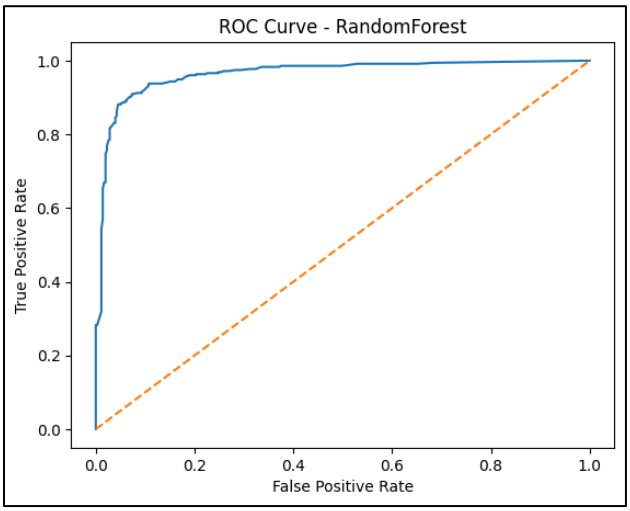

Figure 4. Random Forest delivers the highest performance with 91.3% accuracy and excellent class separation (ROC-AUC 0.964).

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

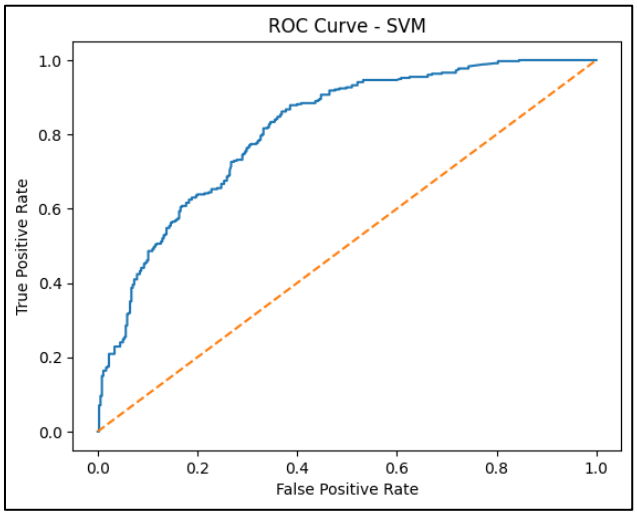

Figure 5. SVM exhibits moderate performance with 72.6% accuracy and fair class separation (ROC-AUC 0.813).

The figure 2 -5 represents the consolidated ROC curves for the four machine learning models. The confusion matrix for the best model achieved with Random Forest algorithm as mentioned in table 4, gives more insight into what the model produces. With only 31 false positives and 31 false negatives out of 709 instances of test cases, the model has a strong and balanced predictive ability, which is essential for allocating testing resources efficiently (minimizing false positives) and catching most of the defects (minimizing false negatives).

Table 4. Confusion Matrix for the Random Forest Model

	Predicted: Non-Defective	Predicted: Defective
Actual: Non-Defective	32s4 (True Negatives)	31 (False Positives)
Actual: Defective	31 (False Negatives)	323 (True Positives)

Feature Importance Analysis: In order to understand the drivers of the predictions made by the Random Forest, we performed feature importance analysis. The most influential top five static code metrics for defect prediction were:

1. LOC_EXECUTABLE (Lines of Executable Code)
2. LOC_BLANK
3. LOC_TOTAL
4. NUM_UNIQUE_OPERANDS
5. NUM_OPERANDS

This analysis is consistent with commonly known software engineering wisdom and shows that metrics that relate to size (LOCTOTAL, LOCEXECUTABLE) and complexity (NUMUNIQUEOPERANDS, NUM_OPERANDS) are powerful predictors of defect-proneness [2], [13]. This gives us actionable advice for the developers, that modules with high values in these metrics require more attention.

Journal of Sustainable Smart Technologies
(JSST)
ISSN: XXXX-XXXX
Volume: 01 | Issue: 01 | July 2026

B. Six Sigma Process Capability Analysis

The key contribution of this research is to go beyond the accuracy of a predictive model to a quantitative process improvement. The results of the best performing model (Random Forest model) were then used to calculate Six Sigma metrics, which offers a universal language for quality.

Baseline Process Performance:

The first analysis, based on the actual defect distribution in the test set, gave a start for the process.

- Observed Defects: 354
- Units (Modules): 709
- DPMO: $\frac{354}{709\times1} \times 10^6 = 499{,}295$
- Sigma Level: ≈ 1.50

A Sigma Level of 1.5 stands for a process with a high defect rate, typical for many unoptimized software development processes, as well as the difficult nature of the KC1 dataset [2].

Process Improvement with ML-Guided Intervention:

This research then simulated how the use of the Random Forest model in prioritizing testing and remediation efforts. By concentrating resources on the 355 modules predicted as defective (which contained 323 true defects), one would be drastically reducing the number of defects escaping to the next phase to be the number of false negatives (31).

- Defects After Improvement: 31 (False Negatives)
- DPMO (Improved): $\frac{31}{709\times1} \times 10^6 = 43{,}724$
- Sigma Level (Improved): ≈ 3.20

This represents a great improvement with the increase in the sigma level from 1.50 to 3.20. This represents a reduction of more than 10 times of the DPMO, a dramatic improvement in the quality and capability of the process.

Comparative Analysis with Prior Work:

The table 5 places the performance of our hybrid framework into the context of the current SDP and software quality literature.

Table 5. Comparative Analysis with Prior Research

Study	Primary Focus	Key ML Result	Six Sigma Integration & Result
[2]	SDP with Naive Bayes	AUC ~0.71-0.89 on NASA dsata	Not integrated.

[3]	Benchmarking ML for SDP	RF often top performer	Not integrated.
[6]	Conceptual ML + Six Sigma	Not specified	Conceptual framework; no empirical DPMO/Sigma calculation.
Proposed System	Hybrid ML + Six Sigma Framework	RF Accuracy: 91.25%, AUC: 0.964	Empirical result: Sigma Level improved from 1.50 to 3.20.

Key Insight: As seen in Table 5, while previous work has done an excellent job of optimizing the predictive performance of ML [2], [3], or has conceptual integrations [6], a key gap has been the absence of empirical and quantitative demonstration of process improvement. This study bridges that gap. That are not only able to have competitive predictive accuracy, but that also explicitly compute the resulting Six Sigma metrics, giving a direct, quantifiable measure of how ML predictions translate into tangible process capability enhancement.

C. Discussion and Implications

The results strongly endorse the central thesis of this research work, namely that combining machine learning and Six Sigma establishes a more powerful paradigm for managing software quality than either approach alone.

Synergistic Value of the Hybrid Framework:

The Random Forest model brings the proactive, diagnostic power to understand where the defects will be likely to occur. Six sigma offers the prescriptive, quantitative analysis to measure the effect of acting on these predictions. This jump in Sigma Level from 1.50 to 3.20 is not just a theoretical calculation but a real potential of savings on rework costs, more reliable releases, and better customer satisfaction by targeting their improvement efforts where they are most critical.

Implications for Software Engineering:

- For Project Managers: This framework provides a rational for test resource allocation based on data: instead of spending equal effort testing all modules, it is possible to focus on the ~17% of modules considered to be high risk according to the model, further enhancing testing efficiency and effectiveness.
- For Quality Engineers: The Six Sigma metrics (DPMO, Sigma Level) are a standard way of communicating quality objectives and achievements to senior management and tracking improvement over time, which may require the software development to be aligned with the quality goals of the larger business.
- For DevOps Teams: In the future, the pipeline should be set up to integrate seamlessly with the CI/CD workflows, where each build is analyzed automatically. Those modules that have been marked as defect-prone could serve as a trigger for mandatory code reviews or extra automated tests, allowing quality control to become second nature while developing.

Limitations and Future Validation:

The study is based on using a single data set (NASA KC1) from a specific domain. Future work will focus on the validation of this framework on a larger number of projects, including modern and commercial systems, aiming to establish the generalizability of the proposed framework. Furthermore, the definition of a defect "opportunity" at module level is a simplification; a more granular definition could be explored. Finally, the implementation of the "Improve" phase in DMAIC in practice - how exactly the predictions guide the interventions - can be elaborated further in industrial case studies.

5. FUTURE WORKS

This research establishes a robust foundation for integrating machine learning with Six Sigma for software quality improvement. Building upon the findings and limitations discussed, several promising directions for future work emerge, which can be categorized into three main themes: enhancing the predictive models, expanding the framework's scope and applicability, and deepening the integration for autonomous quality control.

Developing of Predictive Modeling Techniques:

- Exploring Future Deep Learning Architectures: Future work will explore the use of more advanced ensemble methods such as Gradient Boosting (XGBoost, LightGBM) and Deep Learning models (e.g. Artificial Neural Networks) [14]. These models can capture more complex, non-linear, relationships among the software metrics and, in particular, can outperform the Random Forest classifier used in this study, especially on larger and more complex datasets.
- Explainable AI (XAI): In order to improve the practical utility and trustworthiness of the predictions, it is important that XAI methods (e.g., SHAP: SHapley Additive exPlanations, LIME: Local Interpretable Model-agnostic Explanations) be incorporated as a next step [15]. This would allow software engineers to have actionable information to explain why a particular module is predicted to be defective and guide them in a focused code refactoring and design enhancement activity, rather than simply detecting that something is bad.

Framework Generalization and Scalability:

- Cross-Project and Cross-Company Validation: A main drawback of this study is that it has only been validated on one dataset (NASA KC1). Rigorous evaluation of this hybrid framework on other NASA MDP datasets (e.g., PC1, CM1, JM1) and more importantly, modern large scale commercial and open source systems [16], is the goal of future work. This will be used to evaluate the generalizability and robustness of the framework to data drift and different development contexts.
- Support of Dynamic and Process Metrics: The current model is only based on static code metrics. Based on the above, augmenting the feature set with dynamic metrics (e.g., execution traces, code churn) and development process data (e.g., commit history, developer activity) from CI/CD pipelines

could have a significant impact on improving predictive accuracy [17]. This would make the model a real-time online learning system that adapts to the changing codebase.

6. CONTRIBUTION SDGs

This research contributes primarily to SDG 9 – Industry, Innovation and Infrastructure by integrating machine learning with the Six Sigma DMAIC methodology to improve software quality and defect prediction. The proposed framework enables early identification of defect-prone software modules, optimizes testing resources, and enhances process capability through statistical quality assessment. It also supports SDG 8 – Decent Work and Economic Growth by improving software development efficiency and reducing rework. Furthermore, the application of Six Sigma metrics promotes SDG 12 – Responsible Consumption and Production by encouraging efficient resource utilization, minimizing software defects, and supporting continuous quality improvement in software engineering.

7. CONCLUSION

This study has proposed and statistically confirmed a new hybrid approach of combining the Machine Learning-driven defect prediction with the Six Sigma DMAIC framework of quantitative software process enhancement. The three most important contributions of this work are as follows: (1) a rigorous comparative analysis of the classifiers, which proved that the best model is the Random Forest with a 91.25% accuracy and a 0.964 ROC-AUC; (2) the novel translation of predictions of the ML into the baseline Six Sigma measures, which showed that the process can be improved, and the Sigma Level can be increased by 3.20; and (3) formalizing this predictive-analytical pipeline into the DMAIC cycle, which is a clear action. The framework is a good link between abstracted defect prediction and quantifiable process improvement. It offers a purely empirical reason to allocate resources to software project managers and a common language to monitor improvement by shifting the traditional performance metrics to global quality measures such as DPMO and Sigma Level. The exhibited synergy provides a more holistic approach to software quality management which is shifting the software defect prediction mechanism into a dynamic and statistically validated engine in continuous process enhancement. The research, thus, provides a solid basis in integrating data-driven and statistical control procedures directly into the current software development processes.

Conflict of Interest

The authors declare no conflict of interest.

REFERENCES

[1] B. Boehm and V. R. Basili, “Software Defect Reduction Top 10 List,” IEEE Computer, vol. 34, no. 1, pp. 135-137, Jan. 2001.

[2] T. Menzies, J. Greenwald, and A. Frank, “Data Mining Static Code Attributes to Learn Defect Predictors,” IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2-13, Jan. 2007.

[3] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings,” IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 485-496, July-Aug. 2008.

[4] M. Harry and R. Schroeder, Six Sigma: The Breakthrough Management Strategy Revolutionizing the World’s Top Corporations. New York, NY, USA: Currency, 2000.

[5] S. L. Tang and Y. T. Lee, "An Integrated Methodology for Software Process Improvement Based on Six Sigma and CMMI: A Case Study," Journal of Software: Evolution and Process, vol. 25, no. 9, pp. 1001-1018, Sep. 2013.

[6] Y. Zhang, J. Yin, and S. Wang, "A Software Quality Improvement Framework Based on Six Sigma and Machine Learning," in Proc. 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2019, pp. 194-198.

[7] M. A. Hall, T. M. Khoshgoftaar, and R. Wald, "Ensemble Machine Learning for Software Fault Prediction," in Proc. 2012 11th International Conference on Machine Learning and Applications, Boca Raton, FL, USA, 2012, pp. 266-271.

[8] Y. Pang, X. Xue, and A. S. Namin, "Identifying Effective Software Metrics using Feature Selection and Ensemble Learning for Defect Prediction," in Proc. 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 2018, pp. 957-960.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321-357, 2002.

[10] D. C. Montgomery, Introduction to Statistical Quality Control, 7th ed. Hoboken, NJ, USA: John Wiley & Sons, 2012.

[11] A. A. Alothman, "A Six Sigma DMAIC Framework for Software Process Improvement and its Implementation," International Journal of Advanced Computer Science and Applications, vol. 11, no. 5, 2020.

[12] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[13] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.

[14] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 2016, pp. 785-794.

[15] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in Proc. 31st International Conference on Neural Information Processing Systems (NIPS'17), Long Beach, CA, USA, 2017, pp. 4768-4777.

[16] A. G. Koru, D. Zhang, K. El Emam, and H. Liu, "An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules," IEEE Transactions on Software Engineering, vol. 35, no. 2, pp. 293-304, Mar.-Apr. 2009.

[17] M. D'Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," in Proc. 7th IEEE Working Conference on Mining Software Repositories (MSR), Cape Town, South Africa, 2010, pp. 31-41.