SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Technical Debt Analysis In Software Systems: Implications For System Performance And Maintainability

Brijesh Vala[1], Jyoti Shekhawat[2], Dr. Swarna Swetha Kolaventi[3], Gunjan Bhatnagar[4], Pushpalatha P[5], Kanchana K[6], Leena Deshpande[7], Alok Kumar[8]

[1] Assistant Professor, Department of Computer Science and Engineering, Faculty of Engineering and Technology, Parul Institute of Technology, Parul University, Vadodara, Gujarat, India. Email: Brijesh.vala@paruluniversity.ac.in
ORCID: 0000-0001-7094-7871
[2] Assistant Professor, Department of Computer Science & Engineering, Vivekananda Global University, Jaipur, India. Email: jyoti.shekhawat@vgu.ac.in
ORCID: 0009-0002-9459-8818
[3] Assistant Professor, Department of uGDX School of Technology, Atlas SkillTech University, Mumbai, India. Email: swarna.kolaventi@atlasuniversity.edu.in
ORCID: 0000-0001-9892-847X
[4] Assistant Professor, Department of Computer Application, Dev Bhoomi Uttarakhand University, Dehradun, Uttarakhand, India. Email: socse.gunjan@dbuu.ac.in
ORCID: 0009-0006-3572-1440
[5] Assistant Professor, Department of Computer Science, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, India. Email: pushpalathap@maher.ac.in
[6] Assistant Professor, Department of Commerce, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, India. Email: kanchana@maher.ac.in
[7] Associate Professor, Department of Computer Engineering – Software Engineering, Vishwakarma Institute of Technology, Pune, Maharashtra 411037, India. Email: leena.deshpande@vit.edu
[8] Assistant Professor, Department of School of Engineering & Technology, Noida International University, India. Email: alok.kumar1@niu.edu.in

Abstract
Technical debt (TD) is a persistent challenge in software engineering that impacts system performance, maintainability, and long-term development cost. It arises from suboptimal design decisions, rushed implementations, and evolving requirements that gradually degrade software quality. Existing studies lack a comprehensive model for analyzing TD at the component level and linking it to maintainability degradation and performance loss. This research proposes the Tasmanian Devil mutated Weighted Random Forest (TD-WRF) model to improve understanding and prediction of maintainability outcomes influenced by accumulated TD. The WRF model assigns adaptive weights to features and decision trees to prioritize key TD indicators, while Tasmanian Devil optimization applies controlled stochastic perturbation to enhance pattern exploration and reduce overfitting. A comprehensive dataset consisting of technical debt indicators such as code smells, complexity metrics, and architectural violations is utilized for experimentation. Min-Max normalization is applied during data preprocessing to scale all features uniformly and reduce bias caused by differing metric ranges. Independent Component Analysis (ICA) is used for feature extraction to obtain statistically independent and informative components for model training.The results show that accumulated technical debt increases refactoring effort, reduces readability, and raises maintenance costs, while comparative evaluation demonstrates that the proposed model achieves strong performance with a precision of 0.934. The system is implemented using Python with Scikit-learn and NumPy libraries, ensuring efficient model development and evaluation. In conclusion, the proposed TD analysis model provides a reliable and scalable approach for understanding and predicting the impact of technical debt on software maintainability and system quality.

Keywords: Technical Debt (TD), Maintainability, System Performance, Code Refactoring, Software Quality, Precision, Readability.

1. Introduction

The concept of Technical Debt (TD) is one that uses a metaphor to explain software elements designed with short-term advantages in mind, and that inevitably impacts the future maintenance, enhancement, and introduction of new features in the system [1]. The main concern with TD involves the insufficient documentation of Non-functional Requirements (NFRs) that is pose certain risks [2]. Since TD is related to inadequate documentation of vital quality characteristics, it is possible to detect instances when the quality and maintainability of software are gradually declining. Requirements assume crucial importance for the development of software products because their proper collection is imperative for maximizing the benefits of end users [3, 4]. TD management is an effective process that involves constant interaction between engineers and management with respect to resource distribution among maintenance, Code Refactoring, and scientific software engineering tasks such as tools and algorithms. TD is managed timely to mitigate its adverse consequences on development and software. In most cases, TD can be detected by the developers due to code comments or commit logs, which is known as Self-Admitted Technical Debt (SATD) [5, 6]. In practice, software developers strive to produce high-quality software, and the traditional development environments, including object-oriented, commercial, and scientific software, practical constraints often force developers to prioritize adding new functionality over Code Refactoring or addressing underlying debt [7]. Consequently, the maintenance and evolution of software systems involve extensive effort by programmers, including modifying existing code to add new functionality, fix bugs, or improve specific quality attributes. However, certain activities must be prioritized by the developer, and adding functionality must take precedence over Code Refactoring tasks [8].

Research Aim: The aim of this research is to utilize the proposed Tasmanian Devil mutated Weighted Random Forest (TD-WRF) model to identify debt-prone software components, predict maintainability degradation, and effectively analyze the Technical debt's effects on software development and performance.

Research Organization: Section 1 provides the background of the research. Section 2 focuses on the existing research. Section 3 discusses the methodology of data collection, preprocessing, and future extraction, as well as the proposed model. Section 4 includes results obtained from experimental analysis. Section 5 shows the conclusion with the performance, limitations, and future work.

2. Related Work

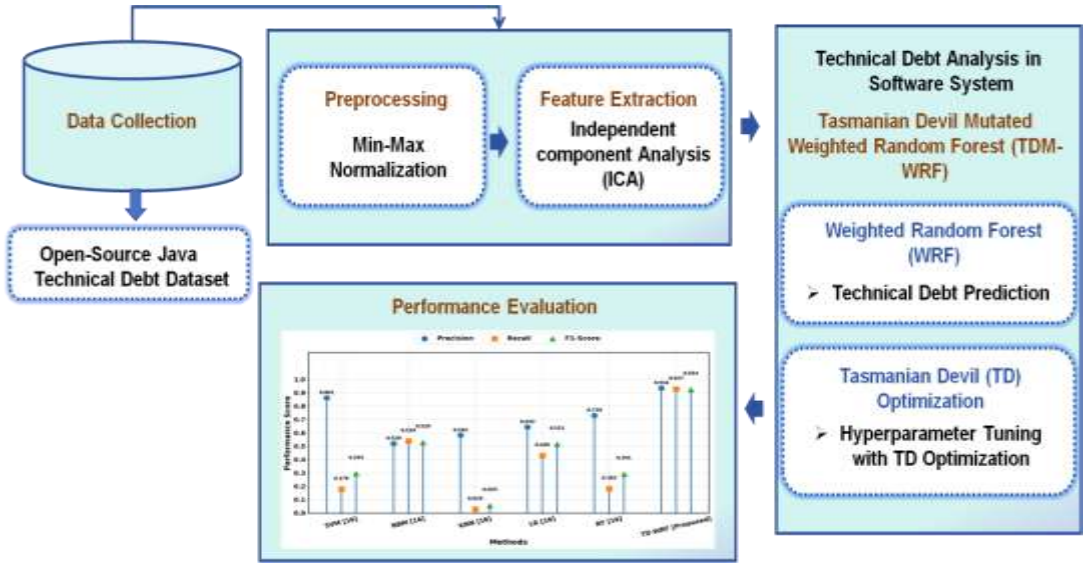
The process of categorizing SATDs in Dockerfiles and identifying the general programming language source code for the TD analysis was developed in this research [9]. The implementation of the Docker-specific type of SATD method provides better readability and maintainability of code. The outcome demonstrates that such a system allows achieving an accuracy of 86.5% of comments being categorized as SATD. Dealing with the Social Debt and its relation to TD in software development initiatives for enhancing the system efficiency was conducted [10] based on Large Scale Agile Development (LSAD) for creating efficient software quality and services, and secure 5G networks. The result identifies the low accuracy, and the 68% of the problems associated with LSAD are caused by social debt, mainly 3C and organization code-related issues. A deep learning-based just-in-time prediction model was examined in [11] to assess the impact of technical debt on software source code changes, leveraging the Java Code (JC) model, which incorporates process quality understanding, reduces readability, and tracks ad-hoc metrics over time. The results show a validation accuracy of 82%. The findings also indicate that the cost of rework increases when an easy but limited solution is chosen, leading to reduced overall effectiveness. TD analysis considers the accumulation of non-optimal solutions in the development and maintenance of software systems that reduce maintainability [12]. The SonarQube tool is used to assess and improve software readability, while also evaluating TD levels associated with function reimplementation and the introduction of various code smells. SonarQube provides reliable and credible metrics within defined confidence intervals for assessing software quality. This research [13] uses a dual technique of manual assessment of TD with the polarity of comments associated with functionality and maintainability of software systems. These techniques involve the use of SATD annotation classification techniques to minimize TD and improve software quality. According to the findings, SATD comments mainly discuss bad implementation decisions (41%) and incomplete implementation (22%). Identification and analysis of SATD clones in software build systems was explored in [14] with respect to their influence on the maintainability and performance reliability of software systems. This research used the Cross-platform Make (CMake) software build system to identify the presence of SATD clones and examine their

properties, particularly in terms of readability. SATD is found to be present in about 76% of the cases examined, suggesting that current methods used to handle SATD comments are inadequate. Understanding the practitioners' reasoning and requirements for efficient tool support in technical debt management to deployment and maintenance was conducted [15] using the Technical Debt Management (TDM) tool for the detection of the TD in relation to system performance and to reduce readability. The TDM tool concentrates on evaluating their functionality without considering practitioners' views. The identification of Self-Admitted Technical Debt (SATD) in issue-tracking systems was investigated to TDs effects on software performance and evolution assist software systems' long-term maintainability and evolution [16]. A traditional machine learning technique, namely the Support Vector Machine (SVM), was employed to predict technical debt in software projects, achieving a precision of 0.861 for maintainability and readability evaluation. To overcome the limitations of existing technical debt prediction methods, this research proposes the Technical Debt–Weighted Random Forest (TD-WRF) method. The proposed approach integrates SATD features, software quality metrics, and debt indicators to improve prediction accuracy, maintainability evaluation, and readability analysis, enabling more effective technical debt management in software systems.

3. Methodology

Figure 1 shows the suggested TD analysis model's methodology for software systems. TD attributes, including code smells, complexity metrics, and architectural issues, are collected from software repositories at the component level. Data preprocessing is a combination of cleaning and normalization procedures, which are then followed by feature extraction that helps to determine debt-prone features. This is developed by the TD-WRF model, which is applied for prediction.

Figure 1: Methodology Workflow for the TDdetection in Software Systems



3.1 Data Collection

The present research focuses on TD within software applications, with an analysis of how it impacts system performance and maintainability using an available dataset from Kaggle (https://www.kaggle.com/datasets/programmer3/technical-debt-maintainability-dataset/data). The database consists of 10,221 instances of software components with 54 features, which include both code metrics at the class and method levels, code smells, architectural aspects, software evolution, and performance-based features. The database is partitioned to use 80% of the data for training purposes and the remaining 20% for testing to enable the proposed TD-WRF model.

3.2 Normalizing the Debt Data Points using Min-Max Normalization

TD analysis is a highly important task where data preprocessing becomes necessary to ensure that all TD measurement tools, including code smells, metrics of complexity, and others, are ready for the learning algorithm. In this research, the Min-Max scaling method is used to standardize feature values and avoid any dominant features with large ranges. This technique transforms all feature values into the range of 0 to 1 while preserving their relative distributions, as defined in Equation (1).

$$Z_{new} = \frac{(Z - Z_{min})}{Z_{max} - Z_{min}} \quad (1)$$

Where Z denotes the TD measure under consideration (such as code smell, complexity), Z_{min} and Z_{max} refer to the minimum and maximum debt values of the measure, respectively, while Z_{new} refers to the standardized debt value within the range of [0, 1].

3.3 Extracting Software Debt Component Featuresusing ICA for System Performance and Maintainability

The TD analysis on software systems ICA is used in extracting the relevant features from the preprocessed dataset, and ICA is a statistical method that uncovers the underlying variables.ICA provides the independence of those statistically independent features that contribute towards TD, like code smell, complexity, etc., improving the software performance and maintenance representation at the component level. The mathematical form of ICA is shown in Equation (2).

$$y(u) = Cs(u) \quad (2)$$

Where y(u) stands for the feature vector that is observed for TD (code smells, complexity measures), C is the mixing debt matrix that remains unknown, and s(u) stands for the statistically independent debt components that make up the TD, which is expressed in Equation (3).

$$\hat{t}(u) = Iy(u) \quad (3)$$

$\hat{t}(u)$ refers to the estimation of the original independent debt features using I as the separation debt matrix.

3.4 Tasmanian Devil Mutated Weighted Random Forest (TD-WRF) for TD Analysis in Software System

The TD-WRF model integrates Weighted Random Forest (WRF) with Tasmanian Devil Optimization (TDO) for hyperparameter tuning to effectively evaluate technical debt in software systems. The TD factors considered in this modeling include code smells, complexity measures, architecture problems, and others. As such, the model highlights those parts of the codebase that are vulnerable to debt and contribute greatly to system maintainability, readability, and performance. This model allows for a detailed assessment of technical debt at the level of individual software components. Using an adaptive method for feature weighting, together with mutation optimization, allows the model to perform better predictions than traditional methods in evaluating maintainability deterioration due to TD.

Weighted Random Forest (WRF) for TD Analysis: The Random Forest (RF) methodology is a type of ensemble method that uses bootstrap sampling and aggregating to merge many decision tree techniques for achieving better predictive accuracy. Nevertheless, classical RF may show poor results in case of imbalanced data sets. To solve this problem, WRF uses weighted decision trees and weighted sampling, focusing more on those parts responsible for TD. The model evaluates debt prediction error using Mean Squared Error (MSE), which is expressed in Equation (4).

$$MSE = \frac{1}{M}\sum_{l=0}^{m}\binom{m}{l}(G_j - Z_j)c^2 \quad (4)$$

Where M refers to the number of software components (data points) involved in the analysis, l represents the indexing variable in the sum, and m denotes the number of observations or debt predictions from the decision tree that contribute to the weighted error. $\sum_{l=0}^{m}\binom{m}{l}$ refers to the binomial coefficient corresponding to the number of debt ways of choosing l objects from m objects, and it shows the weight of the debt contributions from different debt predictions or combinations. G_j denotes the maintainability prediction of the j^{th} software debt component by the WRF algorithm, Z_j represents the observed value of the maintainability of the j^{th} software component, and c^2 refers to the squared error between the debt predicted with better readability and the observed maintainability of each software component.

Hyperparameter Tuning with Tasmanian Devil Optimization (TDO): To enhance the prediction capability of the WRF model for technical debt (TD) estimation, hyperparameter optimization is performed using TDO. TDO is a population-based metaheuristic algorithm inspired by the feeding and scavenging behaviors of Tasmanian

devils. The optimization process begins with the population initialization stage, which is mathematically represented in Equation (5).

TD Initialization Stage:

$$Y_{ji} y_i^{min} + \text{rand}: y_i^{max} - y_i^{min} \quad (5)$$

Where, Y_{ji} denotes the initial value of the i^{th} hyperparameter in the j^{th} candidate solution of the TDO population, j represents the index of the candidate solution (Tasmanian devil) in the population.i represents the index of the hyperparameter being optimized, y_i^{min} denotes the lower bound of the i^{th} Hyperparameter, y_i^{max} denotes the upper bound of the i^{th} Hyperparameter, $y_i^{max} - y_i^{min}$defines the allowable search range of the i^{th}hyperparameter. rand is a uniformly distributed random number within the interval [0,1], which introduces stochasticity and ensures diverse initialization of candidate solutions.

TDExploration Phase:

$$y_{j,i}^{new,t1} = \begin{cases} y_{ji} + r \cdot (d_{ji} - J, y_{ji}), G_{d_j} < G_j \\ y_{ji} + r \cdot (x_{ij} - d_{ji}), \text{otherwise} \end{cases} \quad (6)$$

Where $y_{j,i}^{new,t1}$represent the value of the updated hyperparameter while exploring debt, d_{ji}denotes the location of a good candidate debt solution, and is equivalent to a better module or a possible Code RefactoringJ.rrepresents the random number scaling the step size, and determines the aggressiveness of the debt exploration process.G_{d_j}represents the objective functions of the candidate debt solution, and evaluates improvement (such as decreasing prediction error or the effect of debt).x_{ij} represents the i^{th} dimension of an alternative search position used to diversify exploration when the selected candidate solution does not provide a better fitness value.$G_j$refers to the objective function of the current debt solution, and evaluates the current performance of the system expressed in Equation (7).

$$y_j = \begin{cases} y_j^{new,t1}, G_j^{new,t1} < G_j \\ y_j, \quad \text{otherwise} \end{cases} \quad (7)$$

y_j represents the solution vector for the j^{th} candidate, and the set of hyperparameters after updating. $y_j^{new,t1}$ denotes the updated debt solution vector based on debt exploration. $G_j^{new,t1}$refers to the objective function value for the updated solution, and shows if the new solution improves by lowering the error rate.In the exploitation phase, devils hunt debt prey, refining solutions by considering neighbors' positions expressed in Equation (8).

TDExploitation Phase:

$$y_{j,i}^{new,t2} = \begin{cases} y_{ji} + r \cdot (d_{ji} - J, y_{ji}), G_{q_j} < G_j \\ y_{ji} + r \cdot (y_{ji} - q_{ji}), \text{otherwise} \end{cases} \quad (8)$$

Where $y_{j,i}^{new,t2}$ represent the new value of the hyperparameter when debt exploiting, and concentrates on improving good solutions. q_{ji} denotes the peer solution (neighbor configuration), and is similar to borrowing from others to eliminate debt. G_{q_j} refers to the objective function of peer solution, and measures its performance or TD expressed in Equation (9).

$$y_j = \begin{cases} y_j^{new,t2}, G_j^{new,t2} < G_j \\ y_j, \quad \text{otherwise} \end{cases} \quad (9)$$

Where $Y_j^{new,t2}$ represent the objective function of the new debt solution, and guarantees that only improvements are accepted. $G_j^{new,t2}$ refers to the current objective function value for the updated debt solution, and it is expressed in Equation (10).

$$R = 0.01\ (1 - s/S) \quad (10)$$

R represents the step control parameter for exploitation, and decreases as the number of iterations increases to prevent risk-taking (similar to reducing debts gradually). s denotes the current debt iteration number, and indicates progress made towards debt optimization.Srefers to the total number of debt iterations, and determines debt optimization duration expressed in Equation (11).

$$y_{j,i}^{new} = y_{ji} + (2r - 1).\, R.\, y_{ji} \quad (11)$$

Where $y_{j,i}^{new}$ represent the Hyperparameter obtained after random debt perturbation, and mimics slight changes in the system. R refers to the step-size reduction debt factor, and tends to decrease with time. $(2r - 1)$ denotes the transform of the random number into the range $[-1,1]$ expressed in Equation (12).

$$y_j = \begin{cases} y_j^{new}, G_j^{new} < G_j \\ y_j, \quad \text{otherwise} \end{cases} \tag{12}$$

y_j^{new} represents the perturbed solution debt vector, G_j^{new} refers to the debt objective function for the new debt solution, and guarantees that the system's performance is enhanced or TD decreased.

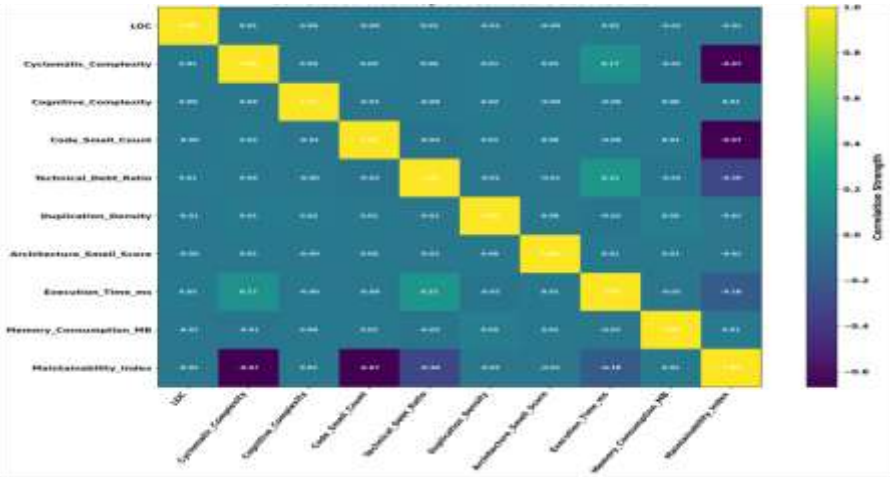
Hyperparameters of Proposed TD-WRF: The research concerning the implications of TD Analysis in Software Systems on System Performance and Maintainability, the suggested TD-WRF model undergoes fine-tuning through the adjustment of several essential hyperparameters to enhance the accuracy of the predictions. Such important hyperparameters are the number of trees (50-500), maximum tree depth (5-50), minimum number of samples per split (2-20), mutation rate (0.05-0.5), mutation power (0.1-1.0), and learning iteration number (10-200). Some other hyperparameters include the number of features, adaptive weight update rate, pruning factor, and voting strategy.

4. Result

Analysis of TD in software components demonstrates a strong correlation between increased TD, decreased system performance, and decreased code readability. The presented TD-WRF framework successfully predicts software maintainability and helps to detect software components that have TD. Results from empirical evaluations show that the proposed TD-WRF model is successful in terms of prediction performance compared to current methods in the field, and moreover, TD contributes to increased refactoring effort, decreased code quality, and increased costs. The implementation is conducted in Python 3.10+ in Jupyter Notebook/VS Code via Scikit-learn, NumPy, Pandas, Matplotlib, SciPy, and optional TensorFlow/PyTorch.

Exploratory Data Analysis: Figure 2 is a correlation graph between TD metrics in software engineering. From the figure below, there is a highly negative correlation between Cyclomatic Complexity and Code Smells with Maintainability Index metrics at -0.67. This means that increased code smells and Cyclomatic Complexity results in decreased Maintainability. Also, there is a moderate correlation between Execution Time and Software Complexity.

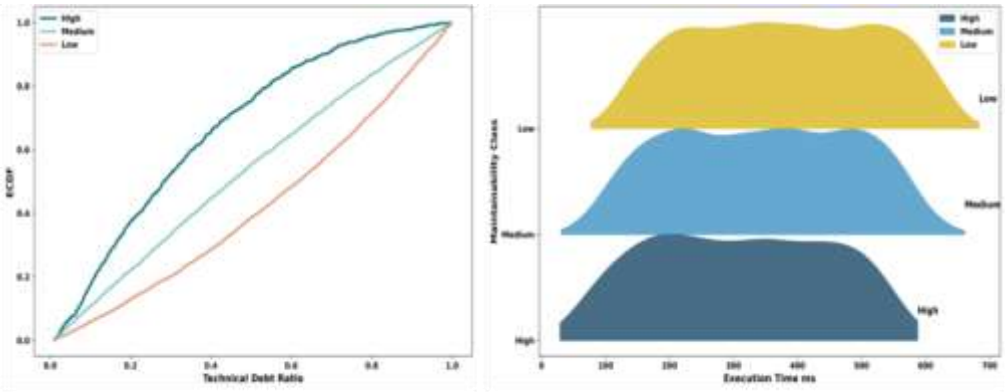
Figure 2: TD Representation in Software Applications (Code Smell, Running Time, Memory Usage)



The Empirical Cumulative Distribution Function (ECDF) of the TD ratio for different software elements is represented in Figure 3(a). The graph displays the percentage of software components having the ratio of TD to the value smaller than the one taken on the horizontal axis. The sharper and higher the graph is, the more the percentage of elements accumulating TD. As depicted in Figure 3(b), a strong correlation can be seen between the software maintenance level and its performance measured by the execution time, the components accumulating

the highest level of TD with high execution time. Conversely highly maintainable components are a low execution time.

Figure 3: Graphical presentation of (a) performance of the TD Ratio, and (b) Execution Time of the System Software TD Maintainability



(a) (b)

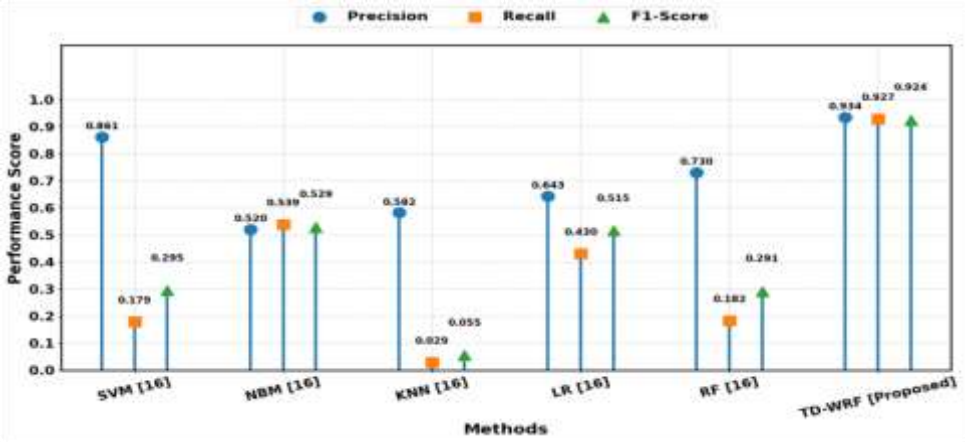
Evaluation Metrics: The evaluation of the efficiency of the proposed TD-WRF method in terms of analysis and prediction of TD and software maintainability relies on the use of objective-oriented criteria such as Precision, Recall, and F1-Score. The metric Precision is used to measure the degree of identification of debt prone modules, whereas the metric Recall reflects the ability of the proposed method to include all relevant cases of debt affecting system performance and maintainability. The F1-Score combines both metrics into an efficient assessment model.

Performance Evaluation: Table 1 and Figure 4 below illustrate the comparison of prediction models, showing that the proposed TD-WRF model performs significantly better than other models, including SVM, Naïve Bayes Model (NBM), K-Nearest Neighbor (KNN), Logistic Regression (LR) and Random Forest (RF) models according to all performance criteria. Though the SVM [16] and RF [16] models provide decent precision rates, their recall value is considerably poor, while the KNN [16] model gives very poor results when it comes to the F1-Score metric. The NBM [16] and LR [16] models demonstrate acceptable results regarding the balance between precision and recall values, and the RF model performs poorly regarding precision when compared to the SVM model. The proposed TD-WRF model demonstrates the best result for precision (0.934), recall (0.927) and F1-Score (0.94), concerning maintainability deterioration and proactive management of TD.

Table 1: Comparison of Existing and Proposed Methods

Methods	Precision	Recall	F1-Score
SVM [16]	0.861	0.179	0.295
NBM [16]	0.520	0.539	0.529
KNN [16]	0.582	0.029	0.055
LR [16]	0.643	0.430	0.515
RF [16]	0.730	0.182	0.291
TD-WRF [Proposed]	0.934	0.927	0.924

Figure 4: Performance Evaluation of TD Analysis in Software Systems Using Existing and Proposed Techniques



Discussion: The main objective of the present research is to conduct a detailed investigation of TD in software systems and measure its influence on system performance and maintainability through an intelligent predictor. There are several problems with the current machine learning techniques such as SVM, NBM, KNN, LR, and RF concerning the identification of debt-prone components. Even though the SVM [15] method gave better results regarding accuracy, it is very poor values concerning recall for debt identification, whereas the RF [15] method faced problems with the detection of all maintainability problems. However, the algorithm utilizing the KNN [15] method is very poor predictive stability due to the influence of data distribution and the high-dimensional nature of software metrics. Meanwhile, although the NBM [15] technique is a good predictor, its efficiency depends on code refactoring. Finally, while LR [15] outperforms some of the other methods in terms of predicting technical debt, it still is some drawbacks related to modeling the nonlinear relations between TD factors. The suggested TD-WRF model is aimed at addressing these issues by implementing an optimal feature weighting mechanism and mutation learning. As a result, the new model accounts for complex non-linear relations with the factors affecting software maintainability.

5. Conclusion

The TD analysis of software systems and its impact on software evolution performance and maintenance is carried out in this research using an intelligent predictive modeling method. Data regarding TDs, including code smells, complexity, and architecture-related issues, is obtained and systematically preprocessed to eliminate inconsistencies and improve data quality. Features about software quality are extracted from the data to accurately predict maintainability degradation. The proposed ML TD-WRF method is used for identifying debt-prone software components and predicting their impact on software evolution, with the achieved precision of 0.934, recall of 0.927, and F1-score of 0.924, and the research is constrained by dataset dependency and the changing nature of software. A key limitation is the dependence on dataset characteristics and static software metrics, which may affect generalizability across diverse projects. Future research will focus on real-time technical debt monitoring, large-scale industrial validation, cross-project prediction, and advanced deep learning techniques for enhanced accuracy.

REFERENCES

1. Borowa, K., Ratkowski, A. and Verdecchia, R., 2025. The technical debt gamble: A case study on technical debt in a large-scale industrial microservice architecture. Journal of Systems and Software, p.112547. https://doi.org/10.1016/j.jss.2025.112547
2. Scott, E., Robiolo, G., Matalonga, S., Felderer, M. and Pfahl, D., 2025. A study on reported under-documented non-functional requirements as an indicator of technical debt. Software Quality Journal, 33(3), p.28. https://doi.org/10.1007/s11219-025-09725-4
3. Perera, J., Tempero, E., Tu, Y.C. and Blincoe, K., 2024. Modelling the quantification of requirements technical debt. Requirements Engineering, 29(4), pp.421-458. https://doi.org/10.1007/s00766-024-00424-3

4. Verdecchia, R., Malavolta, I., Lago, P. and Ozkaya, I., 2022. Empirical evaluation of an architectural technical debt index in the context of the Apache and ONAP ecosystems. PeerJ Computer Science, 8, p.e833. https://doi.org/10.7717/peerj-cs.833
5. Vidoni, M. and Cunico, M.L., 2022. On technical debt in mathematical programming: An exploratory study. Mathematical Programming Computation, 14(4), pp.781-818. https://doi.org/10.1007/s12532-022-00225-1
6. Muse, B.A., Nagy, C., Cleve, A., Khomh, F. and Antoniol, G., 2022. FIXME: synchronize with database! An empirical study of data access self-admitted technical debt. Empirical Software Engineering, 27(6), p.130. https://doi.org/10.1007/s10664-022-10119-4
7. Sharma, R., Shahbazi, R., Fard, F.H., Codabux, Z. and Vidoni, M., 2022. Self-admitted technical debt in R: detection and causes. Automated Software Engineering, 29(2), p.53. https://doi.org/10.1007/s10515-022-00358-6
8. Rantala, L., Mäntylä, M. and Lenarduzzi, V., 2024. Keyword-labeled self-admitted technical debt and static code analysis have significant relationship but limited overlap. Software Quality Journal, 32(2), pp.391-429. https://doi.org/10.1007/s11219-023-09655-z
9. Azuma, H., Matsumoto, S., Kamei, Y. and Kusumoto, S., 2022. An empirical study on self-admitted technical debt in dockerfiles. Empirical Software Engineering, 27(2), p.49. https://doi.org/10.1007/s10664-021-10081-7
10. Saeeda, H., Ovais Ahmad, M. and Gustavsson, T., 2024. Navigating social debt and its link with technical debt in large-scale agile software development projects. Software quality journal, 32(4), pp.1581-1613. https://doi.org/10.1007/s11219-024-09688-y
11. Ardimento, P., Aversano, L., Bernardi, M.L., Cimitile, M. and Iammarino, M., 2022. Using deep temporal convolutional networks to just-in-time forecast technical debt principal. Journal of Systems and Software, 194, p.111481. https://doi.org/10.1016/j.jss.2022.111481
12. Levén, W., Broman, H., Besker, T. and Torkar, R., 2024. The broken windows theory applies to technical debt. Empirical Software Engineering, 29(4), p.73. https://doi.org/10.1007/s10664-024-10456-6
13. Cassee, N., Zampetti, F., Novielli, N., Serebrenik, A. and Di Penta, M., 2022. Self-admitted technical debt and comments' polarity: an empirical study. Empirical Software Engineering, 27(6), p.139. https://doi.org/10.1007/s10664-022-10183-w
14. Xiao, T., Zeng, Z., Wang, D., Hata, H., McIntosh, S. and Matsumoto, K., 2024. Quantifying and characterizing clones of self-admitted technical debt in build systems. Empirical Software Engineering, 29(2), p.54. https://doi.org/10.1007/s10664-024-10449-5
15. Biazotto, J.P., Feitosa, D., Avgeriou, P. and Nakagawa, E.Y., 2025. Understanding practitioners' reasoning and requirements for efficient tool support in technical debt management. Empirical Software Engineering, 30(5), p.134. https://doi.org/10.1007/s10664-025-10691-5
16. Li, Y., Soliman, M. and Avgeriou, P., 2022. Identifying self-admitted technical debt in issue tracking systems using machine learning. Empirical Software Engineering, 27(6), p.131. https://doi.org/10.1007/s10664-022-10128-3