

UNIVERSITY OF WALES TRINITY SAINT DAVID(UWTSD)

Assignment Title: Research Proposal

Md. Nabil Rahman

ID: 2504501

Course Code: ACFA7004

Course Name: Research Methods and Professional Development

Lecturer: Dr. Seena Joseph

MSc Software Engineering and Artificial Intelligence

May 8th, 2026

Table of Contents

1	Introduction	3
1.1	Context and Background.....	3
1.2	Research Problem.....	4
1.3	Research Aim	4
1.4	Research Objectives	4
1.5	Significance of the Research	5
2	Literature Review	6
2.1	Single-Classifer Approaches and Their Limitations	6
2.2	Ensemble Learning and Feature Selection	6
2.3	Class Imbalance Handling.....	7
2.4	Explainability and Interpretability	8
2.5	Identified Gaps and Relation to This Study	8
3	Research Design and Methodology	9
3.1	Overall Research Design.....	9
3.2	Research Methods and Data Analysis	10
3.3	System Architecture and Artefact Design.....	11
3.4	Research Materials: Data Sources, Collection, and Justification.....	12
3.5	Ethical Considerations.....	12
3.6	Software, Libraries, and Hardware	13
4	Expected Outcomes and Limitations	13
5	Project Timeline	14
	References.....	15

1 Introduction

1.1 Context and Background

Despite the industry's long history of software defects, the software sector has emerged as an integral backbone of the modern economic infrastructure, and software defects remain a major cost to organisations and end-users. In 2022, the cost of poor software quality in the United States alone is estimated to be at least \$2.41 trillion due to failures in the software, the cybersecurity risks associated with undetected defects, and the associated operational failures and accumulated technical debt, according to the Consortium for Information and Software Quality (CISQ) [1]. Automated software defect prediction (SDP) has become a data-driven proactive tool that allows one to identify modules in software that are prone to fault early in the development process, thereby significantly lowering the cost of discovering faults in late stages of software development. A study by Zaidi et al. [2] indicates that SDP is one of the most popular research areas in software quality engineering because it can significantly cut down the cost of maintenance and remediation efforts after software has been released, if fault-prone modules can be identified early. The field trains machine learning (ML) classifiers using historical software metrics from version control systems and defect repositories for the purpose of generating probabilistic estimates of which modules are most likely to contain latent defects.

Aggarwal and Kaur [3] have concluded that Support Vector Machines (SVM) and Random Forest have been the most consistently high-performing individual classifiers on various public benchmark datasets in the accuracy, precision, recall and F1-score. The most important input features of most SDP studies have been shown to be object-oriented (OO) software metrics, such as cyclomatic complexity, coupling between objects (CBO), depth of inheritance tree (DIT) and lines of code. Despite this, the current approaches based on ML remain unreliable and are not easily deployable in practice because of structural issues. As a result of the systematic review conducted by Goyal [4] it has been found that the publicly available SDP datasets, including those available in NASA PROMISE, are consistently and severely imbalanced, with the ratio of defective modules as low as 5-20% of all instances. This is an asymmetry that makes the normal classifiers show strong bias towards the non-defective class, thereby severely compromising their sensitivity towards real failures. Mehmood et al. [5] also show that the use of redundant software metrics results in significant reduction in the prediction accuracy, and that systematic feature selection significantly improves the accuracy of the predictions. The other dimension of the field, which is more recent, relates to transparency: practitioners must be able to explain why

they chose to act in a certain way, if they wish to take action based on a defect prediction. Interpretability is starting to be understood as a prerequisite for trustworthy incorporation of ML-based tools into industrial software development pipelines.

1.2 Research Problem

In spite of extensive research efforts, there are three critical gaps in the state of the art in ML-based SDP. Three gaps in the state of the art of ML-based SDP become evident when looking at the research investment made. To begin with, Destefanis et al. [6] reviewed 101 ML-based SDP studies from 2019 to 2023, pinpointing 427 methodological problems, the most common of which included inadequate reproducibility, experimental designs lacking consistency, and the lack of baseline comparisons - highlighting the gap between the number of proposed models and methodological rigor. Second, most SDP studies utilize single-classifier models, and when ensemble models are used, they are not using a unified preprocessing approach that incorporates feature selection and class imbalance handling into one process, so that they are unable to identify specific design choices that contribute to observed improvements. Begum et al. [7] show that black-box ML models are a practical hurdle to adoption because predictions without feature-level explanation are hard to use to clue the developers into the characteristics of the code that needs to be remediated. Third, Varma and Giri [8] found that class imbalance is the primary cause of the systematic failure of traditional ML approaches to detect defective modules, and Dong et al. [9] identified the lack of a known and reliable ensemble selection strategy as an important limitation for reliable performance generalisation across a heterogeneous software repository. There is no existing work that tackles all three challenges in an experimental setting that is reproducible.

1.3 Research Aim

To develop and evaluate an explainable ensemble learning framework for software defect prediction that integrates optimised feature selection and class imbalance handling, evaluated on NASA PROMISE benchmark datasets.

1.4 Research Objectives

The research aim is decomposed into four measurable, time-bound objectives structured in accordance with the Design Science Research cycle:

Objective 1: To critically analyse existing literature on ML-based software defect prediction, examining the limitations of single-classifier models, current ensemble approaches, feature selection strategies, class imbalance handling techniques, and the role of explainability methods.

Objective 2: To design and develop an ensemble learning framework for SDP incorporating Recursive Feature Elimination for feature selection and SMOTE-based class imbalance handling as an integrated preprocessing pipeline, with the stacking ensemble constituting the primary research artefact.

Objective 3: To implement and evaluate the proposed ensemble framework against established single-classifier and ensemble baselines using NASA PROMISE datasets (KC1, JM1, CM1, PC1) and evaluation metrics including Accuracy, Precision, Recall, F1-score, AUC-ROC, and Matthews Correlation Coefficient (MCC).

Objective 4: To assess the interpretability of the proposed ensemble framework using SHAP and LIME analyses, identifying the most influential defect-predictive software metrics and contributing to transparent, trustworthy defect prediction.

1.5 Significance of the Research

This study has meaning in academic, practice and social aspects. Academically, this study offers a sound and auditable methodological baseline for the SDP literature in response to the reproducibility crisis identified by Destefanis et al. [6] by presenting a transparent and fully documented experimental design and highlighting explicitly specified datasets, preprocessing protocols and evaluation criteria. In practice, an ensemble method that combines feature selection, class imbalance processing, and XAI in a single pipeline can yield more interpretable and actionable predictions, which can be incorporated into CI processes, thereby saving time and labor for software development teams in manual quality assurance. On a societal level, the proposed framework could help limit the economic impact associated with undetected defects in deployed software applications, as reported by CISQ [1] in the sectors of healthcare, financial services, transport infrastructure, and more generally in the software industry. The release of the artefact and evaluation protocol in an open, replicable way, also directly contributes to methodological development in the wider SDP research community.

2 Literature Review

It reviews the peer-reviewed research on ML-based SDP, mainly in the time frame from 2020 to 2026, and categorizes them into five classes: limitations of single classifiers; advances in ensemble learning; methods for dealing with class imbalance; feature selection; and explainability. The following keywords were used to find sources in IEEE Xplore, Scopus and the PROMISE benchmark community: software defect prediction, ensemble SDP, class imbalance machine learning, and XAI software fault. The review is designed to identify converging results, methodologic tensions, and gaps that inspire the current study.

2.1 Single-Classifier Approaches and Their Limitations

Single-classifier comparative studies have historically taken the lion's share of the SDP literature. Aquil and Ishak [10] compared various ML methods on thirteen defect datasets, but they did not obtain best performance on all the test cases-showing that the choice of the classification methods is inherently data dependent and that any conclusions drawn based on best results from single datasets evaluations cannot be extended. This was expanded by Pathak et al. [11], who tested the accuracy and F-score of neural networks on various subsets of the NASA MDP and found that for a specific subset, the neural network scores 94.2% accuracy and 92.5% F-score, and that cross-project generalisation is an underlying and unsolved problem. In the review of 45 articles done by Aggarwal and Kaur [3], they have established that Random Forest and SVM are generally the most robust individual classifiers, but they also noted that this is very dependent on the dataset, class distribution and evaluation protocol. Taken together, these studies find that results obtained with one dataset cannot be reliably applied to another, and that single-classifier comparative conclusions are not necessarily generalizable without independent replication across multiple datasets.

2.2 Ensemble Learning and Feature Selection

To counteract the lack of consistency in the results of a single-classifier, ensemble methods using several base learners have been shown to yield more consistent and stable prediction results. Classifiers and classifiers based on clustering techniques were found to give less consistent accuracy than ensemble techniques such as stacking and bagging by Aquil and Ishak [10]. In the present study, the combination of SMOTE and Recursive Feature Elimination (RFE) with the classification algorithms was sought, as such hybrid approach was found empirically beneficial by Tamrakar et al. [12] in this study, where an accuracy of 0.88 was achieved with statistically

significant improvement when compared with any individual classifier; in this study, the classifiers used were also Random Forest, Gradient Boosting, AdaBoost and Logistic Regression. Ensemble design was further improved by Dong et al. [9] who introduced a Double Fault Disagreement diversity measure for base learners selection that outperforms other measures on various datasets (NASA and PROMISE) on AUC, G-Mean, MCC and F1-score and explicitly stated that their approach does not involve systematic feature selection. The independent validation of the critical preprocessing step of feature selection was carried out by Mehmood et al. [5] who showed that prediction accuracy consistently improved with appropriate feature selection across five NASA datasets, and Rahmayanti et al. [15] who reported a highest average AUC of 0.7855 for Recursive Feature Elimination plus Extra Trees on heterogeneous MDP datasets, establishing direct empirical evidence for the adopted feature selection approach, namely Recursive Feature Elimination here. In addition to the semantic metrics, Abdu et al. [16] proved that hybrid feature strategies that include both semantic and static metrics are better than metric-based strategies in cross-project prediction. The common thread is that ensemble performance is thus limited by the quality of the preprocessing prior to feature learning, especially the selection of features and handling of imbalanced data.

2.3 Class Imbalance Handling

The most structurally challenging problem in SDP is the class imbalance situation, where faulty modules occur naturally at a much lower rate than valid ones, leading to a systematic under-detection of faults by a naively trained classifier. The effectiveness of hybridisation of data-sampling techniques with ensemble learning was confirmed by Goyal et al. [4] in a PRISMA-guided systematic review of 41 studies, and that AUC-ROC is the most reliable evaluation metric under imbalanced conditions, because it is insensitive to distribution. On four real-world Java projects, Phung et al. [13] showed that SMOTE can boost performance by up to 6.6% in cases of severe imbalance, with improvements depending on the data and sometimes even causing instability due to the potential for multi-collinearity of over-sampled features: caution should be taken when applying SMOTE, as it may not be suitable for all projects and settings. Although generative augmentation is usually significantly more expensive, Varma and Giri [8] present an augmentation framework based on a GAN, which coupled with a deep residual network voting ensemble achieves better performance than the SMOTE-Tomek baselines on several metrics. In an extension of the problem, Pachouly et al. [14] analysed 14,415 projects to show that 90% of all projects had multiple overlapping defects that were classified at the same time, making binary classification of defects seem too easy.

2.4 Explainability and Interpretability

As the use of ML-based tools moves towards industrial deployment, the interpretability of SDP models has become an urgent issue. The four datasets used by Begum et al. [7] for LIME and SHAP came from NASA, where they found that NOIV (number of instance variables), DIT (dependency between instance variables), and CBO (cyclomatic complexity) are the three most influential defect-predictive metrics with LIME attribution values of 40%, 15%, and 20% respectively, and concluded that black-box predictions without feature attribution fail to guide developers to specific remediation actions. On NASA JM1, Asal and Yalciner [18] compared advanced deep-learning tabular architectures, using SHAP and LIME for interpretability evaluation and reporting, and reported that 88% of accuracy was obtained with well-calibrated explainability outputs with FT-Transformer, while there is no systematic benchmarking under consistent experimental conditions. Arora et al. [19] polled the distribution of research on XAI techniques throughout the Software Development Lifecycle, revealing that 68% of the research on XAI in Software Engineering is conducted during the maintenance phase, with only 8% coming from the pre-release quality assurance phase - underscoring the importance of this study. It has been shown that the semantic intelligence of feature representations has a significantly larger impact on the performance than model complexity [17] demonstrates. This further motivates informed feature design. On the other hand, MSTDP [20] is an instance of temporal DL for commit-level Just-in-Time SDP, which, although a much-needed improvement for the change-level prediction (JIT) problem, still concerns a different sub-problem than the present study, which is predict the module-level SDP in the static-metric prediction setting.

2.5 Identified Gaps and Relation to This Study

There are three cross-cutting gaps that are consistent across the thematic domains reviewed. There is no existing work that combines ensemble learning, RFE feature selection, class imbalance handling using SMOTE and interpretability using SHAP and LIME within a single reproducible experimental framework that is tested across a variety of NASA PROMISE datasets using consistent protocols. Individual studies concentrate on one or two aspects: Tamrakar et al. [12] are trying to integrate SMOTE and RFE with stacking without incorporating XAI; Begum et al. [7] uses only individual classifiers but incorporates XAI (SHAP and LIME); Dong et al. [9] improve the design of ensembles but do not include feature selection or explainability. In addition, the methodological audit conducted by Destefanis et al. [6] showed that reproducibility should be considered as a key design requirement. The current study will be able to fill all three

gaps at once. It also directly addresses the examiner's recommendation in CW1 to go beyond the basic single-model comparison to an ensemble or explainability method, consistent with the findings of the literature.

3 Research Design and Methodology

3.1 Overall Research Design

The overall framework of this study is the Design Science Research (DSR) paradigm which is widely used in Computer Science for a systematic design, development and evaluation of a technological artefact to solve a well-formulated problem. In this case, the goal is not only to test an existing model, but to create a new ensemble framework (the artefact) that is rigorously tested with multiple datasets and proven to be useful. In Figure 1, we show the six phases of DSR cycle, which are adopted in this work: problem identification, objective definition, dataset acquisition and preprocessing, artefact design and development, evaluation and testing, and XAI interpretation and communication. This study follows a positivist research philosophy: Knowledge is based on tangible, measurable facts, and ML model performance is measured on a standardized, repeatable basis using evaluation metrics. The research method is deductive: hypotheses about the comparative superiority of an integrated ensemble framework over single-classifier baselines are drawn from the current literature [3, 9, 12] and tested empirically. The method of research used is fully quantitative because all the data given is numerical (the input data are the OO software metrics, and the output is the binary defect labels and the performance statistics) and it can be analysed statistically. Research strategy: Experimental approach, which changes the independent variables such as classifiers composition, feature selection strategy, and imbalance handling method in order to assess their impact on performance dependent variables, and to perform ablation analysis for causal inference.

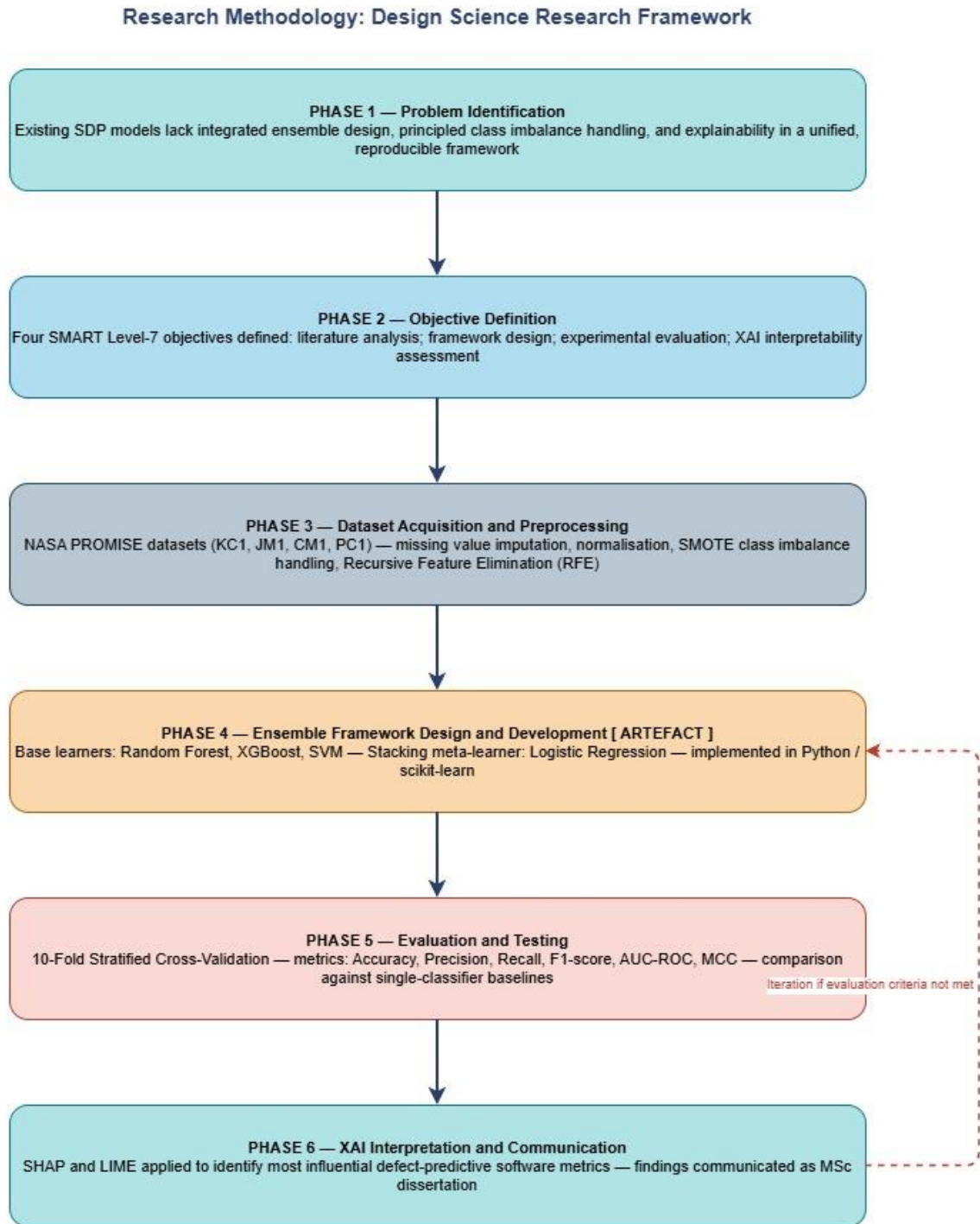


Figure 1: Research Methodology Flowchart - Design Science Research Framework

3.2 Research Methods and Data Analysis

The method of research used is quantitative which is based on the positivist philosophy and experimental approach. The main evaluation protocol adopted is 10-Fold Stratified Cross-Validation where the folds are stratified in order to maintain the ratio of defects to non-defects

in each fold as is documented in the NASA datasets [4, 13]. Another advantage of stratification is that SMOTE-generated synthetic samples are generated only using training data in each fold, and not test-set data, which may cause test-set contamination as documented in the literature by Destefanis et al. [6]. The Wilcoxon signed-rank test, a non-parametric test suitable for comparing the performance of paired classification results without distributional assumptions, will be used to determine the statistical significance of any differences in performance between the proposed ensemble and all the baseline classifiers. The following six evaluation metrics are computed for each model/dataset combination: Accuracy, Precision, Recall, F1-score, AUC-ROC, and Matthews correlation coefficient (MCC). Specifically, MCC is added to the set of measures because it is robust and informative even in strong class imbalance scenarios, as it is a balanced measure that accounts for all four of the confusion matrix cells [4]. Before modeling, imbalance ratios, distributions of features and the rate of missing values in the dataset are described. Post-training, SHAP global attributions and LIME local explanations are applied to quantify the contribution of each software metric to the ensemble predictions as a feature importance ranking, which is compared with the findings of Begum et al. [7] and Asal and Yalciner [18].

3.3 System Architecture and Artefact Design

The research artefact, which is shown in Figure 2, is an integrated ensemble learning pipeline developed in Python with the frameworks scikit-learn and imbalanced-learn. The pipeline is divided into 7 stages. The pipeline has 7 stages. Stage 1 loads four NASA PROMISE datasets (KC1, JM1, CM1 and PC1) and provides documentation for class imbalance ratios, feature distributions, and missing value patterns. In stage 2, the data is pre-processed: missing values are replaced by the median value of the column, features are scaled to a min-max range, and binary defected labels are encoded. In this case, SMOTE is applied only to the training split within each cross-validation fold, which also replicates real deployment, where data leakage is a critical constraint in the implementation [13]. In Stage 4, Recursive Feature Elimination with Random Forest as the internal estimator is applied separately for each dataset to find the optimal feature subset, which was found to be the best, based on empirical studies [15] conducted by Rahmayanti et al. on stability of AUC for different NASA heterogeneous datasets. In stage 5, three base learners (Random Forest, XGBoost and SVM) are trained on the balanced reduced feature space, and their class probabilities are sent to a meta-learner (Logistic Regression) in a stacking configuration. Random Forest is chosen because it is a well established algorithm in the SDP literature [3, 15]; XGBoost because it is gradient boosting and therefore sensitive to patterns in the minority class and resistant to overfitting due to regularisation; and SVM because it has

been proven to perform well in high-dimensional metric spaces [3]. Logistic Regression is used as the meta-learner, because its interpretability and stability in aggregating probabilistic outputs of various base learners. In stage 6, SHAP and LIME are applied to the trained ensemble. Stage 7 calculates all six of the evaluation metrics, and conducts Wilcoxon signed-rank testing for all baselines.

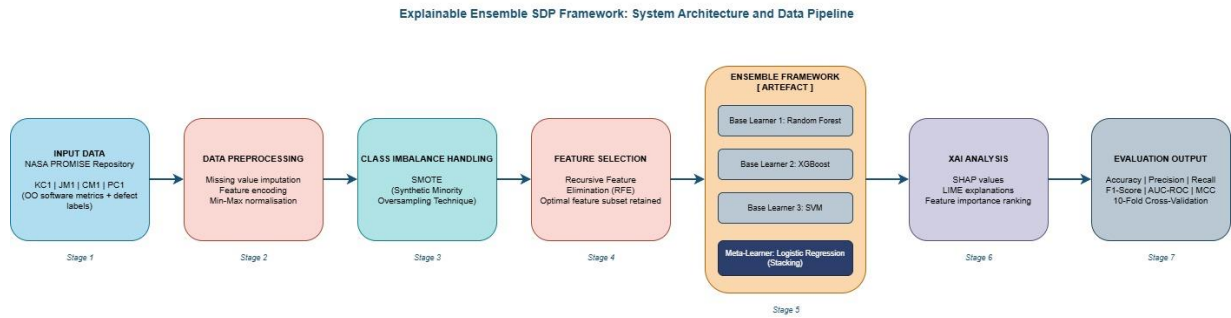


Figure 2: System Architecture and Data Pipeline - Explainable Ensemble SDP Framework

3.4 Research Materials: Data Sources, Collection, and Justification

The data have been gathered from the NASA Metrics Data Program (NASA MDP) part of the PROMISE software engineering repository, which is openly accessible and is established as a de facto comparative standard in SDP research [3, 5]. The selection of four datasets is made: KC1 (C++ ground-system software), JM1 (C real-time predictive ground system), CM1 (NASA spacecraft instrument software), and PC1 (C flight software for Earth-orbiting satellite). They are software systems developed in diverse programming languages, applications domains and organisational teams, offering a diverse test bed for the generalisability of the proposed framework. There are 20 to 22 OO software metrics (Halstead volume, McCabe cyclomatic complexity, and Chidamber-Kemerer metrics), and a binary defect label in each data set. These are selected because they are frequently used within the SDP literature [5, 15] and since there are established baselines available to compare, and these were free and openly licensed to be distributed, with no personally identifiable information, there are no ethical barriers to being used in the data.

3.5 Ethical Considerations

This study does not require human subjects, human data or sensitive data, because it is based exclusively on publicly available, fully anonymised software engineering datasets, which contains no personal data nor sensitive data. Under UWTSD research governance guidelines

there is no requirement for institutional ethical approval for data collection. The study adheres to the UWTSD research ethics framework and the required ethics form has been submitted separately through the Moodle submission link as per the requirements of the module assessment.

3.6 Software, Libraries, and Hardware

The artefact will be written in Python 3.11; the main libraries that will be used are the following: scikit-learn (v1.4): used for ensemble construction, RFE, and stratified cross validation; imbalanced-learn (v0.12): used for SMOTE; XGBoost (v2.0): used as base learner for the ensemble; shap (v0.45): global feature attributions; and LIME (v0.2.0.1): local interpretability. Pandas (v2.2), NumPy (v1.26) will be used for data handling and analysis, and matplotlib, seaborn (v0.12) for visualisation. Structured notebooks will be used to document all experimental steps to guarantee complete reproducibility and code will be publicly accessible and available on a GitHub repository at the time of the dissertation submission. There is no special GPU hardware necessary, the classifiers selected run well on CPU for this size of data. The experiments will be carried out using a personal computer with at least 16G of RAM running windows 11 operating system.

4 Expected Outcomes and Limitations

This research is expected to yield three main outcomes. The first is that the proposed explainable ensemble framework is likely to outperform each of the three single classifiers (Random Forest, XGBoost and SVM) in each of the four NASA datasets on each of the six evaluation metrics as per empirical insights from similar ensemble research works [9, 12]. The ablation study will also show that the combination of the integrated SMOTE preprocessing and RFE feature selection steps are both individually and additively effective in performance improvement, emphasizing the significance of the whole process instead of any individual part. Second, in the stacking ensemble setting, two additional metrics that are found by Begum et al. [7] to be among the most predictive of defect proneness are expected to be among the most predictive according to SHAP: coupling measures (CBO) and complexity measures (lines of code and cyclomatic complexity). Third, the integrated SMOTE and RFE pipeline is expected to yield statistically significant

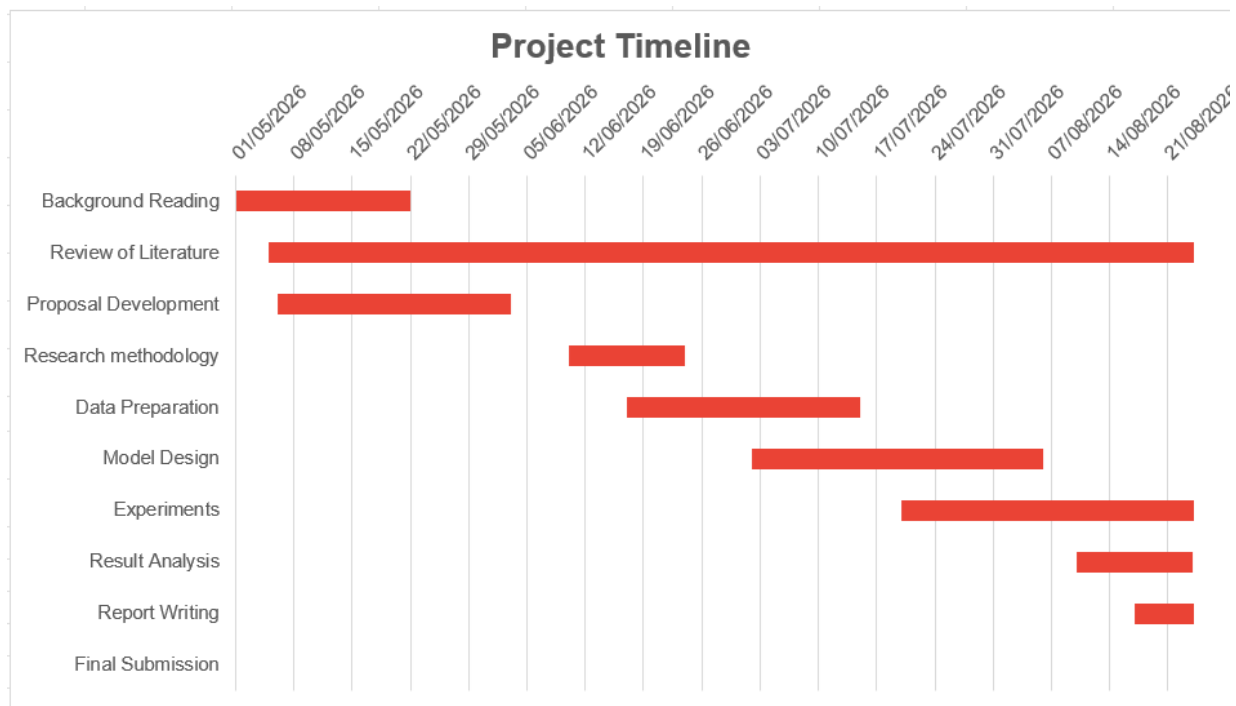
improvement over baseline models trained on unbalanced and unreduced data in terms of AUC-ROC and MCC, thus establishing the meaning of preprocessing design for SDP.

At the proposal stage, there are several restrictions that are recognized. The study's results are limited to the four NASA PROMISE datasets considered in this study; the results may not be generally applicable to commercial and closed-source software projects with varying development practices, programming languages, or defect density profiles. However, SMOTE produces synthetic training instances that might not be a perfect replica of the statistical characteristics of real defective code modules, particularly when the defect pattern in the feature space is structurally complicated or spatially concentrated [13]. The stacking ensemble requires more time to train than each individual classifier, and more time to make inferences, so may be less useful in the contexts of continuous integration in resource-constrained settings. Lastly, SHAP and LIME attributions are model-specific and post-hoc explanations, and should be interpreted as predictive associations between software metrics and fault proneness, not causal statements about how software defects are introduced.

5 Project Timeline

This project will run for 16 weeks from the start of the MSc Research Project module to the completion of the dissertation, which covers the entire length of the MSc project. The milestone schedule is displayed in visual Gantt format in Table 1, where the dark blue cells represent periods of activity for each of the milestones. In the first two weeks, the systematic review is completed and the formal search strategy recorded. In weeks 3 and 4, acquisition, profiling, and exploratory data analysis of datasets is performed. From Week 5, model development begins by implementing SMOTE and RFE pipeline and constructing the ensemble framework, which follows in Weeks 7 and 8. Experimental evaluation (baseline comparison and ablation studies) is in Weeks 9 and 10, followed by analysis of XAI in Week 11 and testing of statistics in Week 12. The last four weeks are dedicated for writing dissertation, structuring review and submission. Tasks are overlapped where dependencies allow, especially when literature engagement and model development are ongoing tasks.

Table 1: Sixteen-Week Project Milestone Schedule



Dark orange cells indicate active phases for each task across the sixteen-week schedule.

References

- [1] Consortium for Information and Software Quality (CISQ), "The Cost of Poor Software Quality in the US: A 2022 Report," CISQ, 2022. [Online]. Available: <https://www.it-cisq.org/the-cost-of-poor-software-quality-in-the-us-a-2022-report/>
- [2] H. Z. Zaidi et al., "Machine Learning Approaches for Software Defect Prediction," Applied Computational Intelligence and Soft Computing, vol. 2025, Article ID 7933078, 2025, doi: 10.1155/acis/7933078.
- [3] R. Aggarwal and K. Kaur, "Machine Learning Approaches in Software Fault Prediction: A Review," Cureus Journal of Computer Science, vol. 3, 2026, doi: 10.7759/s44389-026-00021-7.

- [4] S. R. Goyal, "Current Trends in Class Imbalance Learning for Software Defect Prediction," IEEE Access, vol. 13, pp. 16896-16920, 2025, doi: 10.1109/ACCESS.2025.3532250.
- [5] I. Mehmood et al., "A Novel Approach to Improve Software Defect Prediction Accuracy Using Machine Learning," IEEE Access, vol. 11, pp. 63579-63597, 2023, doi: 10.1109/ACCESS.2023.3287326.
- [6] G. Destefanis et al., "An Audit of Machine Learning Experiments on Software Defect Prediction," Empirical Software Engineering, vol. 31, p. 83, 2026, doi: 10.1007/s10664-025-10797-w.
- [7] M. Begum, M. H. Shuvo, I. Ashraf, A. Al Mamun, J. Uddin, and M. A. Samad, "Software Defects Identification: Results Using Machine Learning and Explainable Artificial Intelligence Techniques," IEEE Access, vol. 11, pp. 132750-132765, 2023, doi: 10.1109/ACCESS.2023.3329051.
- [8] M. M. M. K. Varma and M. Giri, "Deep Residual Network Voting Ensemble for Software Defect Prediction Enhanced by GAN-Based Data Augmentation," International Journal of Computing and Digital Systems, vol. 17, no. 1, pp. 1-10, 2024, doi: 10.12785/ijcds/XXXXXX.
- [9] X. Dong, J. Wang, and Y. Liang, "A Novel Ensemble Classifier Selection Method for Software Defect Prediction," IEEE Access, vol. 13, pp. 25578-25595, 2025, doi: 10.1109/ACCESS.2025.3537658.
- [10] M. A. I. Aquil and W. H. W. Ishak, "Predicting Software Defects using Machine Learning Techniques," International Journal of Advanced Trends in Computer Science and Engineering, vol. 9, no. 4, pp. 6609-6616, 2020, doi: 10.30534/ijatcse/2020/352942020.
- [11] H. Pathak, S. Deshpande, and R. Bernard, "AI Models for Software Defect Prediction: Comparative Study," TechRxiv, 2025, doi: 10.36227/techrxiv.174537505.57148999/v1.
- [12] P. K. Tamrakar et al., "Innovative Approaches to Software Defect Prediction Using Ensemble Learning Models," Future Technology, vol. 5, no. 1, pp. 38-46, 2026, doi: 10.55670/fpll.futech.5.1.4.

- [13] K. Phung, M. E. Aydin, and E. Ogunshile, "Deep Learning Architectures for Software Fault Prediction: The Impact of Error-Type Metrics and Class Imbalance," *Concurrency and Computation: Practice and Experience*, vol. 38, Art. e70472, 2026, doi: 10.1002/cpe.70472.
- [14] J. Pachouly, S. Ahirrao, K. Kotecha, A. Kulkarni, and S. Alfarhood, "Multilabel Classification for Defect Prediction in Software Engineering," *Scientific Reports*, vol. 15, p. 8739, 2025, doi: 10.1038/s41598-025-93242-8.
- [15] R. Rahmayanti, R. Herteno, S. W. Saputro, T. H. Saragih, and F. Abadi, "Comparative Study of Filter, Wrapper, and Hybrid Feature Selection Using Tree-Based Classifiers for Software Defect Prediction," *Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 8, no. 1, pp. 1-16, 2026, doi: 10.35882/ijeeemi.v8i1.294.
- [16] A. Abdu et al., "Cross-project Software Defect Prediction Based on the Reduction and Hybridization of Software Metrics," *Alexandria Engineering Journal*, vol. 112, pp. 161-176, 2025, doi: 10.1016/j.aej.2024.10.034.
- [17] Z. Mujaddid and E. Utami, "Enhancing the GLANCE Framework for Line-Level Defect Prediction: An Empirical Study of Semantically-Aware Metrics and Non-Linear Classifiers," *Telematika*, vol. 18, no. 2, pp. 172-189, 2025, doi: 10.35671/telematika.v18i2.3196.
- [18] B. Asal and B. Yalciner, "Benchmarking TabNet, NODE, and FT-Transformer for Software Defect Prediction: An Empirical Comparison and Explainability Analysis," *IEEE Access*, vol. 14, pp. 11660-11682, 2026, doi: 10.1109/ACCESS.2026.3656247.
- [19] L. Arora et al., "Explainable Artificial Intelligence Techniques for Software Development Lifecycle: A Phase-specific Survey," in *Proc. 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2025, pp. 2281-2288, doi: 10.1109/COMPSAC65507.2025.00321.
- [20] Z. Liu et al., "MSTDP: A Multi-Scale Temporal Deep Learning Framework for Just-in-Time Software Defect Prediction with Cross-Attention Fusion," *Journal of King Saud University Computer and Information Sciences*, vol. 38, p. 68, 2026, doi: 10.1007/s44443-025-00401-y.