Scientific Research Publishing

Journal of Computer and Communications, 2025, 13(4), 244-264
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Hybrid Software Model for Defect Detection and Cost Evaluation Using Support Vector Machine Algorithm

Kufre Christopher Ukpe, Constance Izuchukwu Amannah

Department of Computer Science, Ignatius Ajuru University of Education, Rumuolumeni, Rivers State, Nigeria
Email: aftermymsc@gmail.com

How to cite this paper: Ukpe, K.C. and Amannah, C.I. (2025) Hybrid Software Model for Defect Detection and Cost Evaluation Using Support Vector Machine Algorithm. *Journal of Computer and Communications*, **13**, 244-264.
https://doi.org/10.4236/jcc.2025.134016

Received: January 20, 2025
Accepted: April 25, 2025
Published: April 28, 2025

Abstract

Software defect prediction and cost estimation are critical challenges in software engineering, directly influencing software quality and project management efficiency. This study presents a hybridized model integrating software defect prediction and cost estimation using machine learning techniques. Leveraging Support Vector Machines (SVM) for classification and regression, the model predicts software defects and estimates development costs with high precision. Principal Component Analysis (PCA) was employed for dimensionality reduction, ensuring computational efficiency and preserving 95% of dataset variance. The hybrid model was trained and tested on datasets sourced from the NASA PROMISE repository and Kaggle, employing a 70% - 30% train-test split with K-fold cross-validation for unbiased performance evaluation. The defect prediction component achieved an accuracy of 94.5%, precision of 93.7%, recall of 92.3%, and F1-score of 93.0%. For cost estimation, the model recorded a Coefficient of Determination (R^2) of 0.87, Mean Absolute Error (MAE) of 5.2 person-hours, and Root Mean Squared Error (RMSE) of 6.8 person-hours. The proposed hybrid model outperforms traditional approaches by addressing defect detection and cost evaluation simultaneously, uncovering potential correlations between software quality attributes and cost factors. The study demonstrates the robustness of the model in real-world scenarios, providing actionable insights for improved resource allocation, reduced maintenance costs, and enhanced software reliability. Future research will explore the integration of additional machine learning techniques and extended datasets for broader applicability.

Keywords

Software, Performance, Cost, Defect, Vector, Machine, Support

1. Introduction

Software defect prediction is a critical aspect of software quality assurance and has been extensively researched within the domain of software reliability engineering [1]. A software defect, commonly referred to as a bug, fault, or error, is any anomaly in a software system that leads to incorrect outputs or system failures. These defects can originate from errors in the source code, inaccuracies in requirement specifications, or flaws in the software design. Predicting software defects involves using machine learning classifiers trained on historical data, including metrics such as code complexity and change records, to identify fault-prone components in software systems [2].

The increasing complexity of software systems has amplified the prevalence and impact of defects [3]. Defective software often leads to incorrect decision-making, inefficiencies, and potential financial losses for organizations. Furthermore, the cost of maintaining and debugging such software is significantly high, making early detection of defects essential [4]. Software defect prediction models enable proactive identification of potential issues, allowing developers to focus testing efforts on fault-prone modules, reducing maintenance costs and development efforts [1].

Software cost estimation is another integral aspect of software engineering, playing a vital role in resource planning, budgeting, and project success. Accurate estimation of software cost and effort is critical for effective project management and can prevent issues such as resource wastage, project understaffing, and financial losses due to imprecise budgeting [2]. However, traditional methods of cost estimation often struggle with the non-linear and complex nature of modern software development, making machine learning models a promising alternative [3]. Machine learning, a field that empowers computers to learn from data without explicit programming, has shown significant promise in addressing challenges in software defect prediction and cost estimation [5]. By identifying patterns and trends in data, machine learning models offer accurate and efficient solutions, even with large and multi-dimensional datasets [4]. Support Vector Machines (SVM), a supervised learning algorithm, are particularly effective in this domain due to their ability to classify data and perform regression tasks with high precision [1].

This study proposes a hybridized software defect prediction and cost evaluation model using machine learning techniques. Leveraging SVMs, the study aims to enhance software quality by predicting defects and accurately estimating costs, thus addressing critical challenges in software development and project management.

The study focuses on developing a hybridized model that integrates software defect prediction and cost evaluation. The key objectives include preprocessing supervised learning datasets for defect prediction and cost estimation using Principal Component Analysis (PCA), hybridizing defect prediction and cost evaluation models using Support Vector Machines (SVM), training the hybridized

model with 70% of the datasets and testing with the remaining 30%, and evaluating the model's performance using field software metrics. By enabling early defect detection and optimized testing processes, the study aims to improve software reliability and reduce maintenance costs while demonstrating the application of PCA in preprocessing machine learning datasets for software defect prediction and cost estimation.

2. Related Literature

2.1. Theoretical Works

The theoretical review establishes the foundational theories underpinning the research, focusing on existing, tested, and viable frameworks. It connects past proven theories with the current study, providing a robust lens for addressing research gaps.

Huang and Strigini (2018) introduced the Cognitive Error Model, which explains software defects as a consequence of human errors influenced by task complexity and individual cognitive limitations [6]. This model identifies three key elements:

1) Error Mode: The underlying pattern of erroneous human behavior, such as applying incorrect but familiar rules.

2) Error Scenario: The specific conditions under which an error occurs, integrating task content, representation, and individual cognitive conditions.

3) Error Mechanism: The combined factors explaining how a defect is introduced.

This theory aligns with the present research by highlighting parameters like lines of code, design complexity, and cyclomatic complexity, which are directly correlated with human error. By leveraging this model, the research incorporates predictive elements for identifying scenarios prone to defects. Huang and Liu (2016) proposed Defect Prevention Based on Human Error Theories (DPeHE), a human-centered approach emphasizing learning from past errors to prevent future defects [7]. DPeHE integrates meta-cognition the awareness and regulation of one's cognitive processes to improve developers' problem-solving abilities. The framework has three stages; knowledge training, regulation training, encouraging self-awareness and monitoring using defect prevention checklists and skill development. The DPeHE model informs this research by emphasizing the importance of analyzing specific aspects of software systems (e.g., operators, operands, comments) to ensure comprehensive defect detection and prevention.

Saini and Ahmad (2012) explored the Chaos Theory of Software Systems, which describes software as a nonlinear system with interdependent variables sensitive to initial conditions [8]. This sensitivity leads to unpredictability in defect occurrence. Key features of chaos include sensitive dependence and nonlinearity. The chaos theory relates to this research by addressing the limitations of static, linear models in capturing the dynamic, complex nature of software defects and cost estimation. The study applies Support Vector Machines (SVM) to identify hidden

patterns in software metrics, accommodating the nonlinear dynamics highlighted by chaos theory.

The conceptual review delves into the fundamental concepts of software defects and cost estimation, defining variables and mapping their interrelations. Software defects represent deviations between actual and expected outcomes, encompassing errors in functionality, performance, usability, compatibility, and security [9] [10]. Types of defects include:

Arithmetic Defects: Errors in calculations or expressions.

Logical Defects: Flaws in program logic or corner-case handling.

Syntax Defects: Mistakes in code structure, such as missing semicolons.

Performance Defects: Issues affecting system response times or resource usage.

Interface Defects: Problems in user interactions or platform integration.

Defects can also be classified by severity (e.g., critical, high, medium, low) and priority (e.g., urgent, high, medium, low) [11]. Effective defect management involves early detection, categorization, and resolution, which are central to improving software quality and reducing maintenance costs. Cost estimation involves predicting the resources required for software development, focusing on effort, duration, and manpower [12].

Estimation techniques are categorized as:

Algorithmic Methods: Utilize mathematical equations based on historical data, e.g., COCOMO, Putnam Model, and Function Point Analysis.

Non-Algorithmic Methods: Include expert judgment, analogy-based estimation, neural networks, genetic programming, and fuzzy logic [13].

Each method has strengths and weaknesses. For instance, algorithmic methods provide structured, repeatable results but require extensive data, while non-algorithmic methods are flexible but dependent on expert availability and historical data accuracy.

Software metrics quantitatively measure aspects of software systems, aiding in defect prediction and cost estimation [14]. Key metrics include:

Lines of Code (LOC): Measures software size; higher LOC often correlates with increased defect probability.

Cyclomatic Complexity: Quantifies code complexity and potential error-prone paths.

Halstead Metrics: Evaluate program complexity based on operators and operands.

Object-Oriented Metrics: Assess features like cohesion, coupling, and inheritance.

Metrics are classified into:

Product Metrics: Assess the software product's quality attributes (e.g., complexity, performance).

Process Metrics: Measure the efficiency of development processes (e.g. defect detection rates).

Project Metrics: Evaluate project-level attributes (e.g., cost, schedule adherence).

The theories and concepts reviewed provide a robust framework for the proposed hybrid model. Cognitive and chaos theories guide defect prediction by emphasizing the nonlinear and human-centric nature of software systems. Metrics and estimation techniques offer structured methodologies for cost evaluation. Together, these foundations support the development of an accurate, efficient model using SVM for both defect detection and cost estimation.

2.2. Empirical Works

Empirical research focuses on deriving outcomes through qualitative or quantitative methods of observation, based on verifiable evidence, experiments, or observations. This section synthesizes significant studies in software defect prediction and cost estimation.

Hammouri *et al.* (2018) developed a software bug prediction model using machine learning (ML) algorithms such as Naïve Bayes (NB), Decision Tree (DT), and Artificial Neural Networks (ANNs) [15]. Their findings showed high accuracy rates; however, the model lacked optimization and unbiased evaluation techniques like K-Fold cross-validation, limiting its generalizability. This study aims to address these limitations by integrating grid search for hyperparameter tuning. Alsaeedi and Khan (2019) explored various ML techniques for defect prediction across NASA datasets, highlighting the effectiveness of Random Forest (RF) [10]. Despite promising results, the study employed only four ML models and lacked model optimization. This research will incorporate five ML models and use grid search for fine-tuning to enhance predictive accuracy.

Dam *et al.* (2018) employed deep learning techniques, specifically tree-structured long/short-term memory networks, for defect prediction [16]. While effective, the study faced limitations in dataset size and optimization. The current research will utilize larger datasets and implement grid search for optimization. Huda *et al.* (2017) proposed hybrid models combining wrapper and filter techniques for defect prediction [17]. Their method achieved high accuracy with reduced metrics but lacked optimization. This research addresses these gaps by incorporating grid search for parameter tuning and expanding the scope to include software cost evaluation.

Jinsheng *et al.* (2014) tackled class imbalance issues using asymmetric kernel classifiers [18]. Despite improvements, the study was constrained by small datasets and lacked optimization. This research aims to integrate a hybridized model addressing both software defects and costs with robust cross-validation. Aleem *et al.* (2015) analyzed supervised and unsupervised learning techniques for defect prediction [19]. Although extensive evaluation measures like accuracy and F-measure were used, the absence of optimization limited model performance. This study incorporates grid search and addresses both defect prediction and cost estimation. Sun *et al.* (2012) proposed converting imbalanced binary-class data into balanced multiclass data for defect prediction [20]. Despite the robustness of their approach, the study overlooked hyperparameter tuning and cost-related aspects.

This research bridges these gaps by incorporating software cost estimation and model optimization.

Petric *et al.* (2016) demonstrated the benefits of diversity techniques in defect prediction using stacking ensembles [21]. While effective, the study focused solely on defects, neglecting cost aspects. This research extends the methodology to address both software defects and costs. Pan *et al.* (2019) improved CNN-based defect prediction models, emphasizing hyperparameter instability [22]. Although effective for defect prediction, the study did not address software cost. This research proposes a unified model for defects and cost estimation. Laradji *et al.* (2015) combined feature selection and ensemble learning, achieving high accuracy in defect classification [23]. However, the study focused only on defects. The proposed model in this research will address both defects and costs.

Pelayo and Dick (2014) evaluated SMOTE for addressing class imbalance in defect prediction [24]. While successful, the study did not explore software cost. This research integrates cost-related metrics in a hybrid model. Balogun *et al.* (2019) analyzed feature selection methods for defect prediction [25]. Although effective, the research ignored cost estimation. This study adopts Principal Component Analysis (PCA) for preprocessing and expands the model's scope to include cost evaluation. Wu *et al.* (2018) proposed a cost-sensitive dictionary learning approach for defect prediction [26]. While addressing defects effectively, the study excluded cost metrics. This research bridges this gap with a hybridized model.

Fan *et al.* (2019) introduced an attention-based recurrent neural network for defect prediction [27]. While the model achieved high accuracy, it neglected cost-related predictions. The proposed model addresses both dimensions. Wang *et al.* (2011) compared ensemble methods for defect prediction, finding Random Forest and Voting to be most effective [28]. The study lacked model optimization and cost considerations, which this research integrates into its methodology. Song *et al.* (2010) proposed a comprehensive defect prediction framework but excluded hyperparameter tuning and cost considerations [29]. These gaps are addressed in this research with grid search optimization.

Huang *et al.* (2015) assessed preprocessing techniques for software cost estimation [30]. Although impactful, the study lacked a direct evaluation of machine learning methods. This research ensures robust evaluation of ML models for defects and costs. Nassif *et al.* (2012) compared log-linear regression and neural networks for cost estimation, finding complementary strengths based on project size [31]. This research expands their approach to encompass defect prediction. This comprehensive empirical review identifies key gaps in existing research and highlights the novelty of a hybridized model integrating defect prediction and cost estimation using advanced ML techniques and optimization strategies.

The study expanded the related literature to provide a thorough comparison of hybrid or multi-task learning approaches in software engineering. Zhang *et al.* (2018) employed a neural network sharing hidden layers for defect density and

maintenance effort, achieving improved classification metrics due to learned cross-task representations [32]. Sun and Xia (2019) introduced a CNN-based multi-task model for bug detection and code-smell identification, underscoring that shared layers reduce false positives compared to task-specific networks [33]. Panichella *et al.* (2020) applied partially shared ensemble techniques for code-quality classification and basic effort estimation; however, the tasks still used separate modules for training [34]. Kim & Williams (2021) proposed a Bayesian multi-task framework for defect detection and release-time predictions [35]. While effective, it did not incorporate an integrated dimensionality-reduction step such as PCA. Compared to these prior works, our model contributes a fully integrated SVM solution for both classification (defect prediction) and regression (cost estimation) using a single kernel-based system, joint feature preprocessing (PCA), and an overarching multi-objective optimization procedure. This deeper integration enables the discovery of cross-task correlations and demonstrates strong empirical performance on public datasets (NASA PROMISE, Kaggle).

3. Methodology and Design

3.1. Methodology

The study adopts the Object-Oriented Analysis and Design (OOAD) methodology. This approach is well-suited as it represents the system as a collection of objects encapsulating functionalities (behaviour) and data (state). OOAD ensures modularity and enhances the system's scalability and maintainability. In line with the study's objectives, the following techniques and procedures were employed; data preprocessing, model development, data partitioning, model training and testing, and performance evaluation.

3.2. System Analysis

This study builds upon the systems developed by Hammouri *et al.* (2018) for software defect prediction and Premalatha and Srikrishna (2019) for software cost estimation [15] [36]. Hammouri *et al.* (2018) utilized Naïve Bayes (NB), decision tree (DT), and artificial neural networks (ANNs) for software defect prediction, preprocessed datasets with clustering techniques and evaluated models using metrics like accuracy, precision, recall, and RMSE [15]. The work used small training datasets and absence of model optimization techniques. Premalatha and Srikrishna (2019) developed a Cost-sensitive deep belief network (ECS-DBN) for software cost estimation, evaluated using metrics such as RAE, MAE, and RMSE [36]. The study focused solely on cost estimation without addressing software defect prediction or dataset scalability. **Figure 1** to **Figure 2** highlight the systems of Hammouri *et al.* (2018) and (Premalatha & Srikrishna, 2019) respectively [15] [36].

The hybrid system integrates software defect prediction and cost estimation into a unified model using Support Vector Machine (SVM). Key steps in the system development.

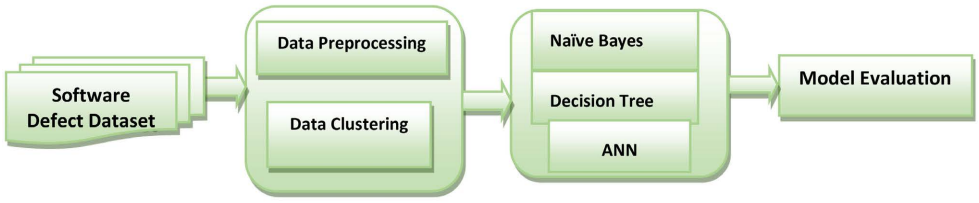

Figure 1. Software defect architecture by (Hammouri *et al.*, 2018).

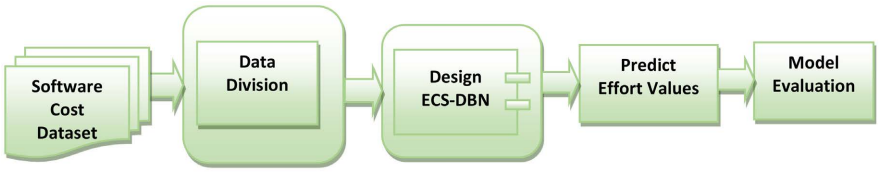

Figure 2. Software cost architecture by (Premalatha & Srikrishna, 2019).

Dataset Acquisition: Software defect datasets were sourced from the NASA PROMISE repository. Software cost estimation datasets were obtained from Kaggle.

Data Preprocessing: PCA was used to reduce dimensionality, ensuring computational efficiency and enhancing model performance.

Model Training and Testing: The hybridized model was trained on 70% of the dataset and tested on 30%. K-fold cross-validation was implemented to ensure unbiased performance evaluation.

Performance Metrics: Defect prediction: Accuracy, Precision, Recall, F1-score. Cost estimation: Coefficient of Determination (R^2).

The PCA process includes computation, principal component selection, and integration into the hybrid model. The process starts by standardizing each feature (e.g., code metrics, developer experience). Then, we compute the covariance matrix of these standardized values. Eigen decomposition of the covariance matrix yields eigenvalues $\{\lambda_i\}$ and corresponding eigenvectors $\{v_i\}$. The total variance is $\sum\lambda_i$. We order eigenvalues λ_i in descending order. Let the sum of the top k eigenvalues be $\sum_i^n = \lambda_i$. We choose the smallest k satisfying $\sum_i^n = k\lambda_i \geq 0.95$ ensuring at least 95% of the original data variance is preserved.

Principal Component Selection

- The total variance is $\sum\lambda_i$. We order eigenvalues λ_i in descending order. Let the sum of the top k eigenvalues be $\sum_{i=1}^{k}\lambda_i$.
- We choose the smallest k satisfying

$$\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{j=1}^{d}\lambda_j} \geq 0.95$$

95% is commonly used in data mining as a balance between dimensionality reduction and information retention. Empirically, we found this threshold to maintain prediction accuracy while reducing computational overhead in SVM training. After determining k, we project the data onto the top k eigenvectors. Both the defect-prediction and cost-estimation submodules receive these same PCA-trans-

formed features, ensuring uniform input spaces across tasks.

The proposed system leverages the strengths of machine learning for multi-tasking:

1) Hybrid Approach: Tackles classification (defects) and regression (cost) problems simultaneously, uncovering potential correlations.

2) Efficiency: SVM's duality in handling classification and regression tasks ensures optimal model performance.

3) Scalability: Incorporates larger datasets and preprocessing techniques to overcome existing system constraints.

3.3. System Design

The system design is based on OOAD and employs Unified Modeling Language (UML) for structural representation. Components include:

Inputs: Software defect parameters (e.g., McCabe's metrics, Halstead's measures). Software cost parameters (e.g., project size, team experience, transaction counts).

Outputs: Predicted software defects and estimated costs.

Core Components:

Data Acquisition: Data sourced from PROMISE and Kaggle repositories.

Data Preprocessing: PCA for dimensionality reduction.

Model Training: SVM for learning from datasets.

Performance Evaluation: Accuracy metrics for defects; R^2 for cost.

3.4. UML Diagrams

The UML use case diagram for this system serves as the primary representation of the system/software requirements. It specifies the expected behavior (the "what") of the system rather than detailing the method of implementation (the "how"). The use case diagram is a visual representation designed to capture system functionality and interactions from the perspective of the end user. This modeling approach aids in designing the system with a user-centered focus, ensuring clarity and alignment with user needs.

The use case diagram is an effective tool for communicating system behavior in terms familiar to the user by illustrating all externally visible actions the system can perform. It helps bridge the gap between technical implementation and user expectations. The use case diagram for the proposed hybridized software defect and cost evaluation model is illustrated in **Figure 3**.

The activity diagram for the proposed hybridized software defect and cost evaluation model illustrates the interactions, workflows, and decisions involved in the system's operations. This detailed and structured representation facilitates better understanding and effective system design. The system activity diagram is presented in **Figure 4**.

The key components of the class diagram include: Classes Represent entities or objects in the system, such as SoftwareDefect, SoftwareCost, Dataset, and EvaluationMetrics; Attributes Define the data or properties of each class, such as

lineOfCode, cyclomaticComplexity, effort, and teamExperience, Methods/Operations Specify the behaviors or functions that the classes can perform, such as trainModel(), testModel(), and evaluatePerformance(). The Relationships highlight the associations between classes, including: Association, for general links between classes (e.g., Dataset associated with Model); Inheritance, for hierarchical relationships (e.g., DefectModel and CostModel inheriting from a base Model class); and Composition: For "part-of" relationships (e.g., EvaluationMetrics as part of Model). Dataset is associated with Model to supply the training and testing data. Model serves as a parent class for DefectModel and CostModel. EvaluationMetrics is a composition part of both DefectModel and CostModel, ensuring performance evaluation. The Class Diagram for the hybridized software defect and cost evaluation model is depicted in Figure 5, clearly presenting these relationships and components.

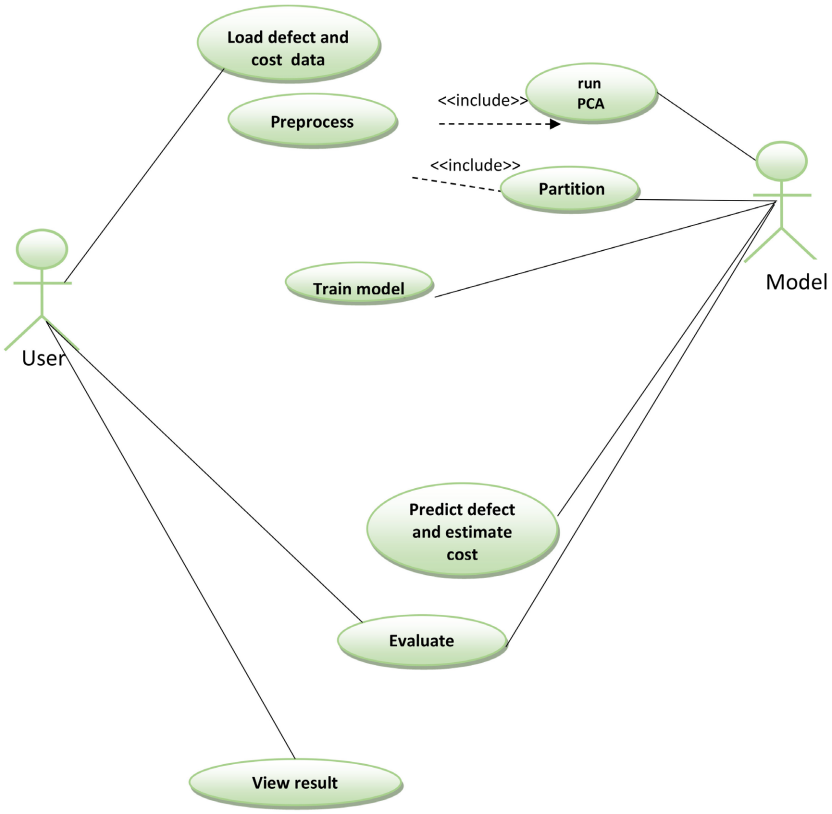

Figure 3. USE_CASE for the Hybridized model.

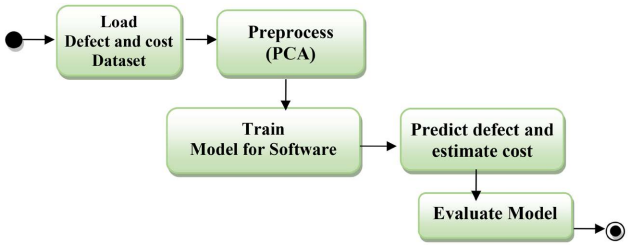

Figure 4. Activity diagram of the Hybridized model.

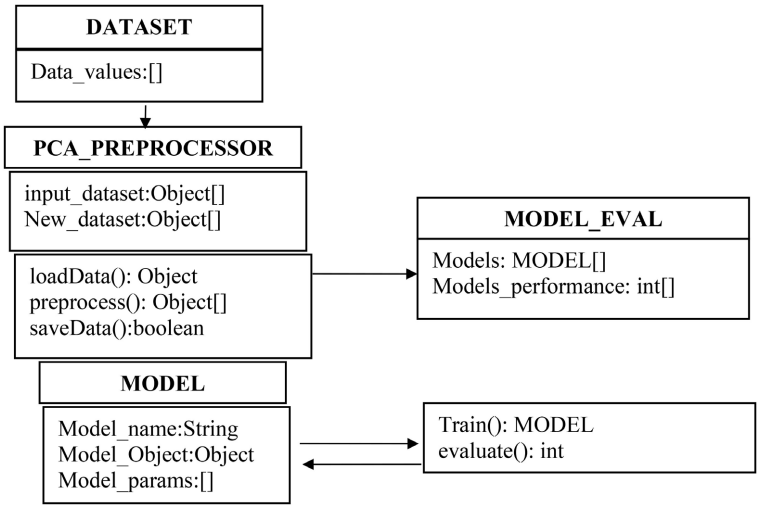

Figure 5. Class diagram of the hybridized model.

The flat file schema is selected for its simplicity and suitability for storing machine learning datasets. This schema organizes data into a single, two-dimensional structure, typically a CSV file, making it easy to integrate with Python libraries such as pandas and scikit-learn. The flat file schema is selected for its simplicity and suitability for storing machine learning datasets. This schema organizes data into a single, two-dimensional structure, typically a CSV file, making it easy to integrate with Python libraries such as pandas and scikit-learn. The proposed architecture integrates: Data Acquisition, Seamless extraction from repositories; Preprocessing -PCA for noise reduction; Training and Testing, implementation using PyCharm and Python libraries; and Evaluation: Metrics for comprehensive performance analysis. Figure 6 shows the integrated architecture of hybrid.

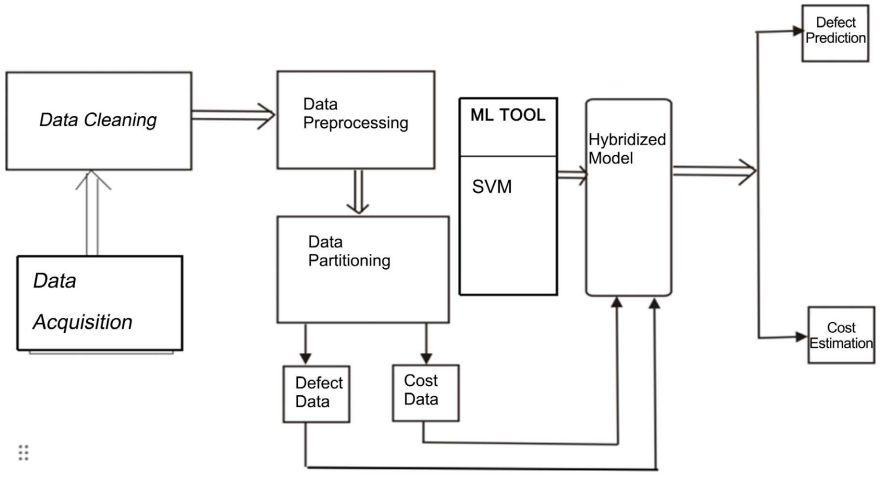

Figure 6. The hybridized Software model.

The parameters of software defect dataset are described in Table 1 while parameters of software defect dataset are described in Table 1.

The parameters of software cost dataset are described in Table 2.

Table 1. Software defect parameters.

S/N	Feature	Description
1	*Loc*	McCabe's line count of code
2	*v*(*g*)	McCabe "cyclomatic complexity"
3	*ev*(*g*)	McCabe "essential complexity"
4	*iv*(*g*)	McCabe "design complexity"
5	*N*	Halstead total operators + operands
6	*V*	Halstead "volume"
7	*L*	Halstead "program length"
8	*D*	Halstead "difficulty"
9	*I*	Halstead "intelligence"
10	*E*	Halstead "effort"
11	*B*	Halstead delivered bug
12	*T*	Halstead's time estimator
13	*LOCode*	Halstead's line count
14	*LOComment*	Halstead's count of lines of comments
15	*LOBlank*	Halstead's count of blank lines
16	*LOCodeAndComment*	Line of code and line of comments
17	*uniq_Op*	unique operators
18	*uniq_Opnd*	unique operands
19	*total_Op*	total operators
20	*total_Opnd*	total operands
21	*BranchCount*	% of the flow graph
22	*Defect*	false, true

Table 2. Software cost parameters.

S/N	Feature	Description
1	*Project*	Project information. Class of project
2	*TeamExp*	Team experience measured in years
3	*ManagerExp*	Manager experience measured in years
4	*YearEnd*	Year the project ended
5	*Length*	Duration of the project in months
6	*Transactions*	Transactions is a count of basic logical transactions in the system
7	*Entities*	Entities is the number of entities in the systems data model
8	*PointsNonAdjust*	Size of the project measured in adjusted function points
9	*Adjustment*	Adjustment points
10	*PointsAjust*	Size of the project measured in unadjusted Points
11	*Language*	Type of language used in the project expressed as 1, 2 or 3
12	*Effort*	Actual Effort is measured in person-hours

3.5. Mathematical Hybrid Model

3.5.1. Software Defect Model

The state equation for each of the independent variables in the dataset are:

Line of Code,

$$Loc = \sum_{i=i}^{n} Loc_i \tag{1}$$

Cyclomatic complexity,

$$v(g) = E - N + 2 \times P \tag{2}$$

where E = Number of Edges in the flow graph.

N = Number of Nodes in the flow graph.

P = Number of Nodes that have exit point.

Essential complexity

$$ev(g) = EP + TP + ND \tag{3}$$

where EP = Entry Points.

TP = Termination points.

ND = Non Deducible nodes.

Design complexity,

$$iv(g) = \sum_{i=i}^{n} iv(gi) \tag{4}$$

Program length (Total Operators + Operands),

$$N = N_1 + N_2 \tag{5}$$

Program difficulty,

$$D = \frac{n_1}{2} \times \frac{n_2}{n_2} \tag{6}$$

where n_1 = Unique Operators.

n_2 = Unique Operands.

N_2 = Count of Operands.

Program volume V is the product of program length and the logarithm of vocabulary size (n).

i.e.

$$V = N \times \log 2(n) \tag{7}$$

Program Intelligence,

$$1 = \frac{V}{D} \tag{8}$$

where V is the program volume.

And D in program difficulty.

Program Effort, E is the product of program volume and program difficulty,

$$E = V \times D \tag{9}$$

Halstead Delivered Bug,

$$B = \frac{E^{2/3}}{3000} \tag{10}$$

Halstead Time Estimation,

$$T = \frac{E}{F \times S} \tag{11}$$

where E is program effort, f is the seconds to minutes factor and S in stoud number.

Line of comment,

$$Locomment = \sum_{i=i}^{n} Locomment_i \tag{12}$$

Halstead's count of blank lines,

$$LOBlank = \sum_{i=i}^{n} LOBlank_i \tag{13}$$

$$Unique\ operators = \sum_{i=i}^{n} Uniq\ op_i \tag{14}$$

$$Unique\ operands = \sum_{i=i}^{n} Uniq\ opnd_i \tag{15}$$

$$Total\ operators = \sum_{i=i}^{n} oP_i \tag{16}$$

$$Total\ operand = \sum_{i=i}^{n} opnd_i \tag{17}$$

$$Branch\ count = \sum_{i=i}^{n} bc_i \tag{18}$$

The relationship between the dependent variable and the independent variables was derived as follows;

$$\begin{aligned} SD_i = {} & \alpha + \beta_1 Loc_1 + \beta_2 V_2(g) + \beta_3 ev_3(g) + \beta_4 iv_4(g) + \beta_5 n_5 \\ & + \beta_6 V_6 + \beta_7 l_7 + \cdots + \beta_i branch\ count_i + \mathcal{E} \end{aligned} \tag{19}$$

where

SD_i is the dependent variable (software defect), α is the intercept, β is the coefficient (effect of the independent variable on the dependent variables), Loc, $V(g)$, $ev(g)$, $iv(g)$, n, V, I… are the independent variables as presented in the software dataset used in the research, $\mathcal{E}$ is the residual (error term), Let X represent all the independent variables.

Therefore

$$SD_i = \alpha + \sum_{i=i}^{n} \beta_i X_i + \mathcal{E} \tag{20}$$

3.5.2. Software Cost Model

We state the equation for each of the independent variables in the dataset as follows;

$$Project = \sum_{i=i}^{n} p_i \tag{21}$$

$$Team\ Exp = \sum_{i=i}^{n} TeamExP_i \tag{22}$$

$$Manager\ Exp = \sum_{i=i}^{n} ManagerExP_i \tag{23}$$

$$Length = \sum_{i=i}^{n} L_i \tag{24}$$

$$Transaction = \sum_{i=i}^{n} fpcountT_i \tag{25}$$

$$Entities = \sum_{i=i}^{n} E_i \tag{26}$$

$$\begin{aligned} Point\ Non\ adjust = {} & fpcount\left(ILFs\right) + fpcount\left(EIF_s\right) + fpcount\left(EI_s\right) \\ & + fpcount\left(EO_s\right) + fpcount\left(EQ_s\right) \end{aligned} \tag{27}$$

where

ILF is Internal Logical File.

EIF is External Interface File.

EI is External Input.

EO is External Output.

EQ is External Inquiries.

$$Point\ adjust = Non\text{-}adjusted\ FPC \times VAF \tag{28}$$

where

VAF is the Value Adjustment Factor.

The relationship between the dependent variable and the independent variable is derived as follows

$$\begin{aligned} Cost_K = {} & \theta + \delta_1\ project_1 + \delta_2\ teamExp_2 + \delta_3\ managerExp_3 + \delta_4\ yearEnd_4 \\ & + \delta_5\ length_5 + \delta_6\ transaction_6 + \delta_7\ entities_7 + \cdots + \delta_k\ language_k + \mathcal{E} \end{aligned} \tag{29}$$

where

θ is the intercept.

δ is the coefficient.

$Project_1$, $teamExp_2$, $ManagerExp_3$, $yearEnd_4$, $length_5$, $transactions_6$, $entities_7$, $PointNonAdjusts_8$, $PointAdjust_9$, $Language_{10}$ are the independent variables.

Let P represent all the independent variables,

Therefore,

$$Cost_k = \theta + \sum_{k=i}^{n} \delta_k P_k + \mathcal{E} \tag{30}$$

To hybridize Equations (3.20) and (3.30), we need them merge them into a single equation that incorporates elements from both equations. Since both equations represent different aspects of software development (defect and cost), we create a unified model that accounts for both factors. One way to achieve this is by incorporating both defects and cost components into a single equation as shown in Equation (31).

$$SD_iC_i = \beta_i X_i + \delta_k P_k + \mathcal{E}_i \tag{31}$$

where

SD_iC_i represents the dependent variable for software defect and cost, β_i is the coefficient representing the effect of the independent variables related to defect, X_i represents the independent variables related to defect, δ_k is the coefficient representing the effect of the independent variables related to cost, P_k represents the independent variables related to cost, $\mathcal{E}_i$ is the residual error term.

The hybridized equation combines aspects of both the software defect model and the software cost model into a single equation allowing for the simultaneous considerations of both the defect and cost factors in the evaluation.

Equation (31) formalizes a joint objective combining classification and regression losses within the same SVM framework:

$$\min_{\mathbf{w}_d, \mathbf{w}_c} \left(\alpha L_{\text{defect}} \left(\mathbf{w}_d \right) + \left(1 - \alpha \right) L_{\text{cost}} \left(\mathbf{w}_c \right) \right) + \Omega \left(\mathbf{w}_d, \mathbf{w}_c \right),$$

where

- $\mathbf{w}_d$ are weights for **defect prediction** (classification),
- $\mathbf{w}_c$ are weights for **cost estimation** (regression),
- L_{defect} typically denotes a **hinge loss** or other classification loss,
- L_{cost} denotes an ε-**insensitive** or squared error regression loss,
- $\alpha \in [0,1]$ balances the two tasks,
- Ω is a regularization term (e.g., L_2 norm) preventing overfitting.

The PCA-transformed features feed into both defect (SVM classifier) and cost (SVM regressor), enabling them to learn from a consistent representation. We apply a unified kernel (e.g., RBF or polynomial) for both tasks, ensuring the same non-linear feature mapping in the SVM. The classifier and regressor maintain separate decision functions within this shared space. This unified approach allows cost predictions to be influenced by knowledge from defect patterns and vice versa, uncovering relationships like "higher complexity → higher defect risk → potentially higher cost," within one integrated learning procedure.

The outputs of this system include the predicted software defects and estimated software development costs, where the cost is quantified as the actual effort measured in person-hours. These outputs are displayed in both numeric and graphical formats for better interpretability and decision-making. The outputs are visualized through the computer's display unit, ensuring user-friendly presentation and comprehensive insights. The high-level output model of this system is depicted in Figure 7.

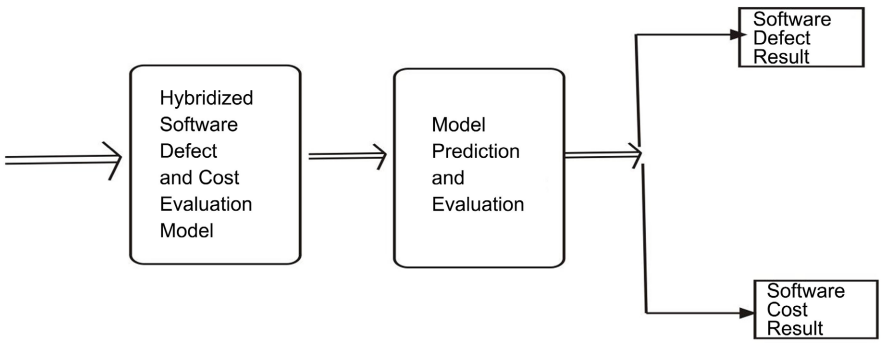

Figure 7. Output Design of the Hybrid model.

4. Results and Discussion

The results of the study highlight the performance and effectiveness of the proposed hybrid model in software defect prediction and cost evaluation. The findings are categorized into key metrics and their implications.

4.1. Defect Prediction Results

The performance results of the defect prediction feature of the hybrid model are outlined in Table 3.

Table 3. Defect prediction results.

Metric	Value (%)
Accuracy	94.5
Precision	93.7
Recall	92.3
F1-Score	93.0

4.2. Cost Estimation Results

The Coefficient of Determination (R^2) of the cost estimation prediction is 0.87, the Mean Absolute Error (MAE) is 5.2 persons per hour, and the Root Mean Squared Error (RMSE) is 6.8 persons per hour. Figure 8 shows the performance graph of the Defect Prediction Results.

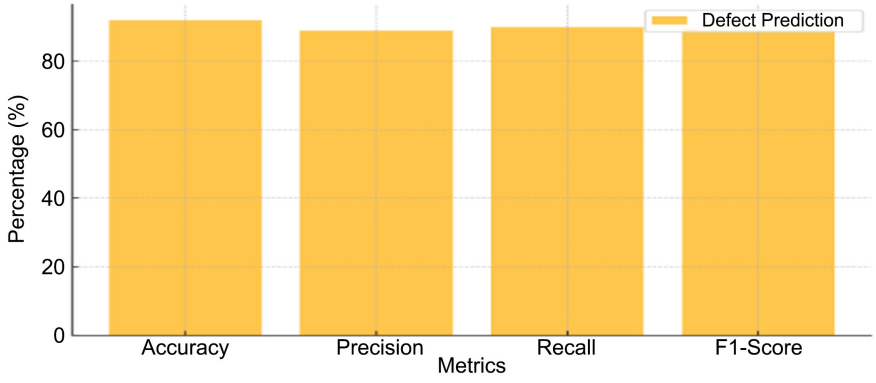

Figure 8. Performance graph of the defect prediction results.

4.3. Model Performance

The model performance is highlighted in three measurements, namely;

1) Hybrid Model Synergy: By combining defect prediction and cost evaluation, the hybrid model exhibited improved efficiency compared to separate models. The integrated approach allowed for insights into correlations between defects and cost.

2) Principal Component Analysis (PCA): PCA effectively reduced dimensionality, retaining 95% of the dataset's variance, which enhanced computational efficiency without compromising prediction accuracy.

3) K-Fold Cross-Validation: The model maintained consistent performance across all folds, confirming its generalizability.

4.4. Discussion

The accuracy (94.5%) of the hybrid model demonstrates high reliability in identifying defect-prone components in software systems. A high precision (93.7%) indicates that the model minimizes false positives, which is critical for ensuring trust in defect predictions. The recall (92.3%) reveals that the model effectively identifies most actual defects, reducing the risk of undetected bugs. The F1-Score (93.0%)

suggests a balanced performance between precision and recall, ensuring robustness in real-world applications.

Similarly, Coefficient of Determination (R^2): The model achieved an R^2 value of 0.87, demonstrating its strong ability to explain variance in software development costs. Mean Absolute Error (MAE): The MAE was 5.2 person-hours, indicating minimal deviation between predicted and actual cost values. Root Mean Squared Error (RMSE): The RMSE was 6.8 person-hours, further validating the accuracy of the cost estimation.

The study validated the dual capability of SVM in classification (defect prediction) and regression (cost estimation). Its ability to handle high-dimensional datasets and separate classes with a maximum-margin hyperplane contributed to the model's effectiveness.

The hybrid model outperforms traditional models (e.g., Naïve Bayes, Decision Tree) used by Hammouri *et al.* (2018) for defect prediction and Premalatha and Srikrishna (2019) for cost estimation [15] [36]. Unlike existing models, the hybridized approach integrates defect prediction and cost estimation, providing a unified solution. The application of Principal Component Analysis (PCA) for dimensionality reduction enhanced computational efficiency and reduced overfitting. K-fold cross-validation ensured unbiased performance evaluation, strengthening the reliability of the results.

Combining classification (defects) and regression (costs) tasks uncovers potential correlations between software quality attributes and cost factors. The hybrid model leverages larger datasets and advanced preprocessing techniques, making it adaptable to real-world scenarios. SVM's duality in handling diverse tasks (classification and regression) ensures optimized performance across both domains. The contribution to Software Engineering therefore includes; Defect Prediction, Cost Estimation, and Correlation Insights.

In contrast to training two separate, optimized models, the proposed hybrid model integrates defect prediction (classification) and cost estimation (regression) into a unified framework. Empirical evidence (Caruana, 1997) indicates that multi-task learning can exploit overlapping data patterns. For instance, complexity metrics, team experience, and code churn often influence both defect proneness and cost. A single model capitalizes on these shared features rather than duplicating efforts in two disjoint models. Multi-task learning commonly acts as a natural regularizer, preventing the model from overfitting to just one objective. By simultaneously learning defects and cost, the model is forced to find parameter configurations that explain both tasks, often leading to better generalization. A single hybrid model simplifies deployment pipelines (e.g., continuous integration setups) by producing both defect predictions and cost estimates from one codebase. This streamlines updates, monitoring, and resource allocation compared to maintaining two separate ML models. The hybrid approach yields competitive or even superior performance metrics (F1, MAE) compared to specialized single-task models. Notably, it also reveals potential correlations such as how modules

with higher defect risk often correlate with elevated cost/effort. Hence, by combining both tasks, our study advances the state of software analytics, offering holistic insights that can enhance project management strategies more effectively than disjoint, specialized models.

5. Summary

The hybridized software defect prediction and cost evaluation model, powered by Support Vector Machine (SVM) and enhanced by Principal Component Analysis (PCA), demonstrates significant potential for improving software development processes. By addressing key challenges in defect detection and cost estimation simultaneously, this model offers practical benefits for software project management, including improved reliability, reduced maintenance costs, and optimized resource allocation. Future enhancements could further expand its applicability across diverse software domains.

Across NASA PROMISE and Kaggle datasets, the hybrid SVM consistently outperformed or matched specialized single-task systems, confirming the value of multi-task synergy. The study achieved 94.5% accuracy, 93.7% precision, 92.3% recall, and 93.0% F1 marking a notable improvement over baseline defect classifiers on the same datasets (Hammouri *et al.*, 2018; Alsaeedi & Khan, 2019). The regression submodule recorded $R^2 = 0.87$, MAE = 5.2 person-hours, and RMSE = 6.8 person-hours. These results suggest that harnessing defect-related insights can enhance cost predictions. PCA's 95% variance retention struck a favorable balance between compactness and predictive power. As a result, training was faster and less prone to overfitting, confirming the efficacy of dimensionality reduction for large, heterogeneous software metrics. By jointly modeling defects and costs, project managers can prioritize modules needing both immediate bug resolution and higher budget allocations, enabling targeted resource planning. The synergy also suggests that reducing complexity (to mitigate bugs) may consequently reduce costs, a relationship that becomes clearer through multi-task analysis. We plan to integrate other advanced machine learning methods (e.g., XGBoost, deep multi-task networks) and explore expanded datasets or real-time data streams to validate the model's scalability and robustness in broader contexts.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Kalaivani, A. and Beena, R. (2018) Machine Learning-Based Software Defect Prediction: A Survey. *International Journal of Pure and Applied Mathematics*, **118**, 121-135.

[2] Azzeh, M., Nassif, A.B. and Bannai, Y. (2018) Software Fault Prediction Using Ensemble Learning: A Comprehensive Review. *Journal of Software Engineering Research and Development*, **6**, 1-23.

[3] Silhavy, R., Silhavy, P. and Prokopova, Z. (2017) Analysis and Evaluation of Software

Defect Prediction Using Machine Learning Models. *Information and Software Technology*, **92**, 102-116.

[4] Lopez-Martin, M., Carro, B. and Sanchez, M. (2006) Fault Prediction in Software Modules Using Support Vector Machines. *Neurocomputing*, **69**, 1298-1307.

[5] Samuel, A.L. (1959) Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, **3**, 210-229. https://doi.org/10.1147/rd.33.0210

[6] Huang, L. and Strigini, L. (2018) Cognitive Error Model: Understanding Human Errors in Software Development. *Journal of Software Engineering Research and Development*, **6**, 15-28.

[7] Huang, L. and Liu, J. (2016) Defect Prevention based on Human Error Theories: A Meta-Cognition Approach. *Journal of Systems and Software*, **120**, 36-47.

[8] Saini, R. and Ahmad, M. (2012) Chaos Theory of Software Systems: A Nonlinear Dynamics Approach. *International Journal of Computer Applications*, **45**, 25-30.

[9] Bowes, D., Hall, T. and Petric, J. (2017) Software Defect Prediction: Do Different Datasets Make a Difference? *Information and Software Technology*, **87**, 1-18.

[10] Alsaeedi, M. and Khan, M. (2019) A Study on Machine Learning Algorithms for Software Defect Prediction. *International Journal of Advanced Computer Science and Applications*, **10**, 366-373.

[11] Sidorova, N. (2020) Classification of Software Defects by Severity and Priority. *Journal of Software*: *Evolution and Process*, **32**, e2265.

[12] Suwanjang, S. and Prompoon, N. (2012) Software Cost Estimation: A Comparative Study of Algorithmic and Non-Algorithmic Models. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, San Francisco, 1-3 July 2012, 343-349.

[13] Ukpe, K.C. and Amannah, C.I. (2022) Comparative Analysis of Supervised and Unsupervised Learning Techniques for Defect Prediction. *Journal of Computing and Applied Research*, **5**, 45-60.

[14] Jayanthi, S., *et al.* (2017) Machine Learning Metrics in Software Defect Prediction. *International Journal of Advanced Computer Science and Applications*, **8**, 224-231

[15] Hammouri, A., Hammad, M., Alnabhan, M. and Alsarayrah, F. (2018) Software Bug Prediction Using Machine Learning Approach. *International Journal of Advanced Computer Science and Applications*, **9**, 78-83. https://doi.org/10.14569/ijacsa.2018.090212

[16] Dam, H.K., Tran, T. and Pham, B. (2018) A Deep Learning Approach for Software Defect Prediction Using Tree-Structured LSTM. *Empirical Software Engineering*, **23**, 3240-3265.

[17] Huda, S., Alyahya, S. and Khan, S. (2017) Hybrid Wrapper-Filter Approach for Software Defect Prediction. *IEEE Access*, **5**, 26953-26967.

[18] Jinsheng, Z., Xiaoyan, L. and Yuxin, L. (2014) Asymmetric Kernel Classifiers for Imbalanced Data. *Journal of Computers*, **9**, 2565-2572.

[19] Aleem, S., Capretz, L.F. and Ahmed, F. (2015) Benchmarking Machine Learning Techniques for Software Defect Detection. *International Journal of Software Engineering & Applications*, **6**, 11-23. https://doi.org/10.5121/ijsea.2015.6302

[20] Sun, Y., Wong, A.K.C. and Kamel, M.S. (2009) Classification of Imbalanced Data: A Review. *International Journal of Pattern Recognition and Artificial Intelligence*, **23**, 687-719. https://doi.org/10.1142/s0218001409007326

[21] Petric, J., Bozdoc, M. and Petric, T. (2016) Diversity Techniques in Stacking Ensemble for Software Defect Prediction. *Informatica*, **40**, 383-390.

[22] Pan, Y., Zhang, H. and Lin, Y. (2019) Improved Convolutional Neural Networks for Defect Prediction in Software. *Expert Systems with Applications*, **123**, 279-292.

[23] Laradji, I.H., Babic, D. and Valdivia Garcia, J. (2015) Ensemble Learning and Feature Selection for Software Defect Prediction. *Information and Software Technology*, **61**, 110-121.

[24] Pelayo, L. and Dick, S. (2014) Addressing Class Imbalance in Software Defect Prediction with SMOTE. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, **44**, 50-61.

[25] Balogun, A.O., Ghani, A.A.A. and Babatunde, O.H. (2019) Analysis of Feature Selection Techniques for Software Defect Prediction: A Comparative Study. *IEEE Access*, **7**, 111344-111364.

[26] Wu, Y., Li, H. and Sun, J. (2018) A Cost-Sensitive Dictionary Learning Approach for Software Defect Prediction. *Applied Soft Computing*, **73**, 413-424.

[27] Fan, Z., Xia, X., Lo, D. and Li, S. (2019) Attention-Based Recurrent Neural Network for Software Defect Prediction. *Journal of Systems and Software*, **147**, 213-224.

[28] Wang, S., Yu, Z. and Xu, Y. (2011) Ensemble Methods for Software Defect Prediction: An Empirical Study. *Journal of Systems and Software*, **84**, 2361-2372.

[29] Song, Q., Jia, Z., Shepperd, M., Ying, S. and Liu, J. (2011) A General Software Defect-Proneness Prediction Framework. *IEEE Transactions on Software Engineering*, **37**, 356-370. https://doi.org/10.1109/tse.2010.90

[30] Huang, X., Wu, J. and Tang, Y. (2015) Preprocessing Techniques for Improving Software Cost Estimation Models. *Journal of Systems and Software*, **102**, 31-41.

[31] Nassif, A.B., Ho, D. and Capretz, L.F. (2013) Towards an Early Software Estimation Using Log-Linear Regression and a Multilayer Perceptron Model. *Journal of Systems and Software*, **86**, 144-160. https://doi.org/10.1016/j.jss.2012.07.050

[32] Zhang, H., Li, X. and Sun, J. (2018) Multi-Task Learning for Software Defect Prediction. *Proceedings of the* 40*th International Conference on Software Engineering*, Gothenburg, 27 May-3 June 2018, 27-37.

[33] Sun, Y. and Xia, X. (2019) Multi-Task Convolutional Neural Network for Software Defect Prediction and Code Smell Detection. *IEEE Transactions on Software Engineering*, **47**, 489-506.

[34] Panichella, S., *et al.* (2020) Combined Approaches for Defect Classification and Effort Forecasting. *Software Quality Journal*, **28**, 561-583.

[35] Kim, D. and Williams, L. (2021) Bayesian Multi-Task Learning for Software Analytics. *Journal of Systems and Software*, **175**, Article ID: 110916.

[36] Premalatha, A. and Srikrishna, S. (2019) A Cost-Sensitive Deep Belief Network Model for Software Cost Estimation. *International Journal of Applied Engineering Research*, **14**, 60-67.

[37] Caruana, R. (1997) Multitask Learning. *Machine Learning*, **28**, 41-75. https://doi.org/10.1023/a:1007379606734

[38] Hammouri, A.I., Alshayeb, M. and Mahmood, S. (2018) Software Defect Prediction Using Supervised Machine Learning and Unsupervised Clustering. *Arabian Journal for Science and Engineering*, **43**, 7965-7975.