


# Convolutional Neural Networks for Software Defect Categorization: An Empirical Validation


**Ruchika Malhotra**

(Delhi Technological University, Delhi, India)

 <https://orcid.org/0000-0003-3872-6213>, [ruchikamalhotra@dtu.ac.in](mailto:ruchikamalhotra@dtu.ac.in)

**Madhukar Cherukuri**

(Delhi Technological University, Delhi, India)

 <https://orcid.org/0009-0005-9062-6027>, [madhukar@dtu.ac.in](mailto:madhukar@dtu.ac.in)

**Abstract:** The escalating complexity and scale of software systems have rendered them increasingly susceptible to a variety of defects. To empower maintenance teams to efficiently prioritize and resolve defects, Software Defect Categorization (SDC) models have emerged, offering the classification of software defects into categories such as "high," "medium," or "low." This study embarks on the development of SDC models, based on three critical defect attributes: i) the maintenance effort required to rectify a defect, ii) the change impact on the software induced by defect resolution, and iii) a combined approach that integrates both maintenance effort and change impact. Leveraging the prevailing advancements in computational power and storage capacity, the study presents a novel defect categorization model built upon Convolutional Neural Networks (CNNs). Extensive experiments were carried out on defect datasets from five Android operating system application modules, leading to the creation of 60 SDC models (5 datasets x 4 feature sets x 3 approaches). The results underscore the predictive potential of our CNN-based defect categorization model. Furthermore, SDC models rooted in the combined approach exhibit superior performance when compared to models based solely on change impact and remain competitive with those anchored in maintenance effort.

**Keywords:** Software Defect Categorization, Software Maintenance, Convolutional Neural Networks, Machine Learning

**Categories:** D.2, I.2

**DOI:** 10.3897/jucs.117185

## 1 Introduction

Software maintenance is a crucial phase within the software development lifecycle. This phase encompasses tasks such as addressing defects that emerge during production and integrating new customer requirements and change requests. Given the real-time nature of these changes in the production environment, there is a pressing need for their swift implementation and testing, all while working within the constraints of limited resources [Hernández-González *et al.*, 2018][Malhotra and Cherukuri, 2024]. Software Defect Categorization (SDC) emerges as a valuable strategy to efficiently manage these resources during the maintenance phase. It involves assigning a defect level (high, medium, or low) based on specific defect attributes like criticality and priority [Tian *et al.*, 2014]. Existing research in defect categorization has demonstrated that basing the defect level on maintenance effort and change impact yields robust predictive

capabilities compared to other defect attributes. The study proposed in [Malhotra and Cherukuri, 2020] explored an approach by combining maintenance effort and change impact to construct defect categorization models using the Multinomial Naïve Bayes learning algorithm, based on a study by [Ummat, 2019]. Maintenance effort, in this context, refers to the estimation of effort needed to rectify a defect, often quantified by the number of lines of code (LOC) that require modification (addition, deletion, or alteration). Change impact, on the other hand, gauges the number of classes that need modification to resolve a defect. Defects categorized as high level based on maintenance effort demand more resources for their resolution, while those categorized as high level based on change impact necessitate additional resources for regression testing. Defects categorized as high level based on both attributes require substantial resources for both defect removal and regression testing [Alsolai and Roper, 2020].

Recent studies have consistently confirmed the superiority of machine learning techniques over traditional statistical models [Pachouly *et al.*, 2022]. Notably, deep learning methods have surged in popularity, driven by advancements in computing capabilities and storage capacity. One prominent deep learning architecture that has gained prominence is the Convolutional Neural Network (CNN), which is inspired by the structure and functioning of the visual cortex in the human brain. This study introduces a novel software defect categorization model built upon a Convolutional Neural Network (SDC-CNN). The study is driven by two primary objectives:

To determine the predictive capability of convolutional networks for software defect categorization. This objective entails a comprehensive exploration and comparative analysis of performance of models for each defect level – low, medium and high.

To determine the most effective attribute for constructing robust Software Defect Categorization (SDC) models. This involves systematically evaluating and comparing three key attributes: maintenance effort, change impact, and a combination of both attributes to identify the optimal attribute for defect categorization.

To achieve these objectives, the study poses several research questions:

- i. RQ1: What is the predictive capability of SDC-CNN models for high level defects.?
- ii. RQ2: What is the predictive capability of SDC-CNN models for medium level defects.?
- iii. RQ3: What is the predictive capability of SDC-CNN models for low level defects.?
- iv. RQ4: What is the performance of SDC-CNN models that assign defect levels based on maintenance effort.?
- v. RQ5: What is the performance of SDC-CNN models that assign defect levels based on change impact.?
- vi. RQ6: What is the performance of SDC-CNN models that assign defect levels based on the combined effect of maintenance effort and change impact.?
- vii. RQ7: What is the comparative performance of SDC-CNN models using the combined effect of maintenance effort and change impact, compared to levels allocated based on a) maintenance effort and b) change impact.?

The study provides a comprehensive and reproducible framework for efficiently implementing CNN, offering a detailed account of parameter configurations and employing a stratified 10-fold cross-validation method. The study encompasses defect data extracted from five application modules of the Android operating system:

Bluetooth, Browser, Calendar, Camera, and MMS. These datasets are prepared by [Malhotra and Khanna, 2022], by employing advanced text mining techniques to extract important words from textual descriptions of the defects identified in the defect reports. The datasets comprise lists of the Top10, Top25, Top50, and Top100 keywords, which serve as independent variables during the creation of SDC models. In total, the study involves the development of 5 (datasets) x 4 (feature sets) x 3 (approaches) = 60 SDC models. The performance evaluation of these models are assessed with the three measures - Area Under the Receiver Operating Characteristic (ROC) curve (AUC), Accuracy and Matthew Correlation Coefficient (MCC). Additionally, statistical tests - Friedman test and Wilcoxon signed-rank test are conducted to assess and compare the performance of the approaches.

In summary, this research contributes to defect categorization and software quality models, leveraging advanced techniques such as neural networks. It addresses significant questions in the field of software maintenance and quality assessment.

The structure of the paper is organized as follows: Section 2 presents a thorough literature review and Section 3 outlines the research methodology and experimental setup. Section 4 presents the results of the study and provides answers to the research questions. Section 5 discusses potential threats to the validity of our work and Section 6 offers conclusions drawn from the study and outlines future directions.

## 2 Related Work

This section presents the related literature and motivation of this study.

### 2.1 Studies on Software Defect Categorization

In the realm of software defect categorization, various studies have aimed to gauge the severity of defects based on the information contained within defect reports. [Lamkanfi *et al.*, 2010] employed the textual descriptions within defect reports to gauge the criticality of defects. [Han *et al.*, 2017] harnessed the power of text mining and deep learning to assess the criticality of defects based on their descriptions. [Sharma *et al.*, 2015], on the other hand, classified bugs into two distinct levels: critical and non-critical, with their research demonstrating the efficacy of bug categorization through text mining of bug reports. [Chaturvedi and Singh, 2012] conducted a comprehensive study evaluating the performance of six classification algorithms in estimating defect criticality solely from the textual descriptions. Meanwhile, [Yang *et al.*, 2012] explored three parameter tuning methods, namely information gain, chi-square, and correlation coefficient, in conjunction with the multinomial naive Bayes classifier to predict defect criticality using bug reports. In the domain of change impact analysis, researchers have pursued various avenues, including static code inspection [Acharya and Robinson, 2011] [Jashki *et al.*, 2008], dynamic code inspection [Cai and Santelices, 2015], and combinations thereof [Maia *et al.*, 2010]. Additionally, some studies have delved into the application of text mining techniques to version histories, aiming to dissect the change impact on software [Zimmermann *et al.*, 2005] [Abdeen *et al.*, 2015].

Furthermore, when it comes to estimating software maintenance effort, the majority of researchers have often leaned on Lines of Code (LOC) as a key metric [van Koten and Gray, 2006] [Jin and Liu, 2010] [Zhou and Leung, 2007] [Malhotra and Cherukuri, 2023]. [Jindal *et al.*, 2016] took an approach of categorizing defects based on the degree

of maintenance effort essential for their rectification. [Kumar and Sureka, 2017] computed the modified LOC between two versions and categorized their target variable into low, medium, and high levels. Shifting the focus to JavaScript applications, [Zozas *et al.*, 2019] delved into previously underexplored facets of the maintenance process. They introduced two novel maintenance indices aimed at estimating the magnitude of changes and the effort required for maintenance tasks in this specific context. In a broader scope, [Miloudi *et al.*, 2020] conducted a comprehensive review of studies pertaining to the estimation of maintenance effort in open source software. Their analysis underscored bug resolution as the predominant factor considered when estimating future maintenance efforts. [Rao *et al.*, 2023] employed Synthetic Minority Oversampling Technique (SMOTE) class balancing technique to handle the problem of class imbalance in code smell severity detection and demonstrated its effectiveness.

These studies collectively contribute to a deeper understanding of defect severity assessment, change impact analysis, and software maintenance effort estimation in the field of software research.

## 2.2 Studies on Convolutional Neural Networks for Software Quality Models

Convolutional Neural Networks (CNNs) offer a solution to the supervised learning challenge by processing input data through a sequence of convolutional filters and basic non-linear operations [Koushik, 2016]. While CNNs have gained prominence in tasks like image classification, their versatility extends to tackling a diverse range of machine learning problems [LeCun *et al.*, 2015]. In the context of software defect prediction, [Khan *et al.*, 2022] conducted a system review of artificial neural networks and [Giray *et al.*, 2023] conducted a comprehensive review of deep learning techniques. Their findings highlighted CNN as the most commonly employed deep learning algorithm in this domain, underscoring its effectiveness and widespread use. [Umer *et al.*, 2020] introduced a CNN-based automated approach for multiclass prioritization of bug reports, combining NLP preprocessing and software engineering emotion analysis to outperform existing methods. [Rathnayake *et al.*, 2021] introduced a CNN-based approach for predicting the severity of bug reports. [Balasubramaniam and Gollagi, 2022] utilized an advanced CNN (Convolutional Neural Network) architecture, optimizing CNN weights through a novel hybrid SALO (Seagull Adopted Ant Lion Optimization) model to achieve precise and accurate defect prediction. [Kim and Yang, 2022] presents a CNN-LSTM model for predicting bug report severity based on topic-specific features, demonstrating its efficiency compared to baselines in Eclipse and Mozilla open source projects. [Bibyan *et al.*, 2023] presented a prediction model combining content analysis using LDA and sentiment analysis to predict bug report severity, by considering both aspects, content and sentiment, with the help of Convolutional Neural Network (CNN). [Khleel and Nehéz, 2023] proposed a novel approach combining convolutional neural network (CNN) and gated recurrent unit (GRU) with synthetic minority oversampling technique (SMOTE) and Tomek link (SMOTE Tomek) to predict software defects.

## 2.3 Motivation

Our review of the existing defect categorization literature revealed several key findings:

- Convolutional neural networks (CNNs) are underutilized in software defect categorization models.

- CNNs are predominantly employed for feature extraction rather than classification in defect prediction studies.
- The majority of studies have focused on defect categorization using either maintenance effort or change impact individually, with limited exploration of an integrated approach.

### 3 Research Methodology

This section presents the various datasets, variables, performance measures and statistical tests considered in this study. The research methodology employed in this study is visually illustrated in Figure 1.

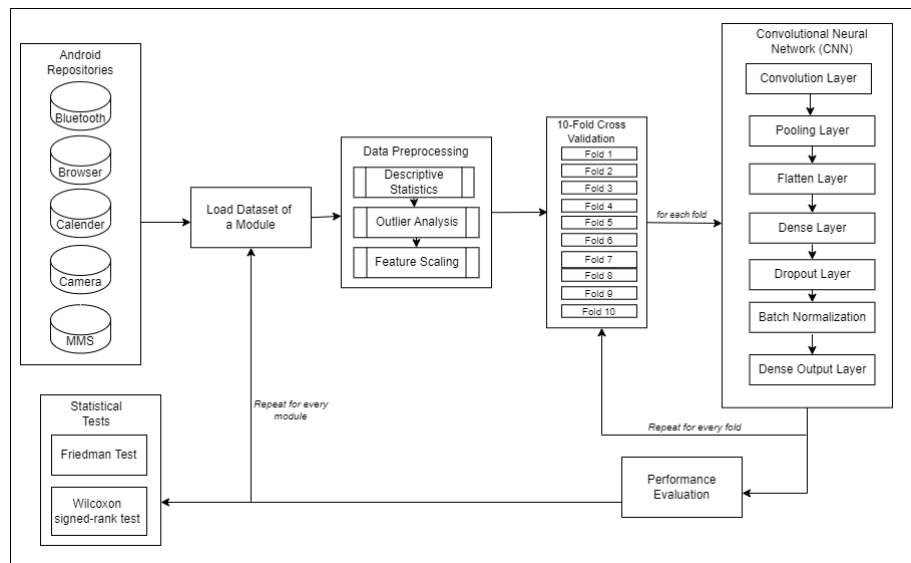


Figure 1: Proposed Research Methodology of the Study

#### 3.1 Datasets

The study examined defect data from five Android software modules available on GitHub, specifically analyzing change logs between different versions (Bluetooth: 4.1-4.4, Browser: 2.3-4.0, Calendar: 4.1-4.4, Camera: 2.3-4.0, MMS: 2.3-4.0) [“Android,” 2023]. These datasets are prepared by [Malhotra and Khanna, 2022] in the following manner: Defect Collection and Reporting System (DCRS) [Malhotra *et al.*, 2014] was utilized to extract defect reports from these change logs, which provided essential defect details, including a unique identifier, affected source code file, defect description, lines of code (LOC) added and removed for defect correction. Text mining techniques were employed to extract keywords from the defect descriptions, following pre-processing and feature selection using the information gain measure and the vector space model as defined by [Malhotra, 2016]. This resulted in Top10, Top25, Top50, and Top100 ranked

keywords, which were used as independent features in constructing Software Defect Categorization (SDC) models. These models categorized defects based on maintenance effort and change impact, assigning them to different severity levels (low, medium, and high) within the Android application packages under investigation. The datasets utilized in this study are accessible on a GitHub repository located at <https://github.com/madhukarcherukuri/datasets/tree/main/Android%20Defects>.

### 3.2 Proposed Convolutional Neural Network for Software Defect Categorization Model (SDC-CNN)

Convolutional Neural Networks (CNNs) represent a category of deep learning models extensively utilized for the analysis of image and sequential data, including text classification. The core of a CNN consists of convolutional layers designed to learn and apply filters to the input data, scanning through it to detect patterns or features [Suong and Jangwoo, 2018]. In the case of text data, these filters are typically one-dimensional. The convolution process in a CNN entails element-wise multiplication of a small filter (referred to as a kernel) with the input data, resulting in the creation of a feature map. This mathematical operation can be expressed as follows:

$$S(j) = (X * W)(j) = \sum_{i=0}^{F-1} X(j+i) * W(i) \quad (1)$$

where  $S(j)$ : Value of the feature map at position  $j$ ,  $X$ : Input data,  $W$ : Filter (kernel) weights, and  $F$ : Filter size.

Following the convolutional layers, pooling layers are frequently employed to reduce data dimensionality while preserving crucial features. Max-pooling, a common technique, involves selecting the maximum value within a small region. The mathematical representation for one-dimensional max-pooling is as follows:

$$P(j) = \max_{i=0}^{K-1} C(j+i) \quad (2)$$

where  $P(j)$ : Output value after max-pooling,  $C$ : Feature map, and  $K$ : Pooling window size.

Subsequent to the convolutional and pooling layers, CNNs generally incorporate one or more fully connected layers, akin to traditional neural network layers found in Multi-Layer Perceptrons (MLPs). MLPs are versatile tools for modeling relationships between various inputs and outputs, employing multiple intermediate layers, often referred to as hidden layers. For developing Software Defect Categorization (SDC) models, the study employed back-propagation learning, a technique that optimizes the network's weights to minimize the disparity between observed and desired outputs.

$$z_{ij}^l = \sum_{p=0}^{m-1} \sum_{q=0}^{n-1} x_{i+p,j+q}^{l-1} w_{p,q}^l + b_j^l \quad (3)$$

$$y_{ij}^l = \phi^l(z_{ij}^l) \quad (4)$$

where  $x_{i,j}^{l-1}$  represents the output of the previous layer at position (i,j),  $w_{p,q}^l$  represents the filter at position (p,q) in layer l,  $b_j^l$  represents the bias term for neuron j in layer l,  $\phi^l(\cdot)$  represents the activation function in layer l and  $y_{ij}^l$  represents the output of neuron at position (i,j) in layer l.

Table 1 outlines the CNN's architecture along with its parameter configurations. The CNN implementation is carried out using python APIs – keras [Keras Special Interest Group (Keras SIG), 2023] and sci-kit [Pedregosa *et al.*, 2011]. The architecture and model parameters were chosen based on a heuristic approach. The models begin with the input layer, which is configured to expect data in the form of 1D arrays, denoted as (X, 1). Moving forward, a 1D convolutional layer with 64 filters and a kernel size of 4 is employed to capture essential features from the input defect data. The smaller kernel size focuses on localized patterns, which can be crucial for identifying specific defect characteristics. ReLU (Rectified Linear Unit) is used as the activation function in the convolutional and dense layers to introduce non-linearity, enabling the network to learn complex mappings between defect features and their categories. The output from this convolutional layer is then directed into a max-pooling layer with a pool size of 2. This step serves two purposes: reducing the complexity of the feature maps and guarding against overfitting. Another significant aspect of max-pooling is its ability to introduce a degree of translation invariance. This means that even if a feature is detected in different spatial locations within the input (e.g., different positions in a defect report), max-pooling will capture the most prominent occurrence of that feature. This property is beneficial in defect categorization tasks where defects may manifest in varying contexts or locations within the input data. Then the flatten layer is used to transform the output of convolutional and pooling layers into a suitable format for the next dense layer featuring 128 units and employing ReLU activation.

To further mitigate the risk of overfitting, a dropout layer is introduced with a dropout rate of 0.25, randomly deactivating 25% of input units during training. Then Batch Normalization layers are used to stabilize and accelerate the training of CNNs by normalizing the input data and introducing learnable parameters to fine-tune the normalization. Following this, the output of the aforementioned layer proceeds to another dense layer, this time comprising a three units (each unit corresponds to a different defect level – low, medium and high) and equipped with a softmax activation function. This configuration generates a probability score indicating the likelihood of the input belonging to one of three classes.

### 3.3 Model Validation

To assess the accuracy of our constructed models, we employed a k-fold cross-validation approach, as proposed by [Stone, 1974]. This method involves dividing the dataset into k randomly selected subsets. During each iteration, k-1 of these subsets serve as training data, while the remaining one is used for validation, as recommended by [Tantithamthavorn *et al.*, 2017]. This process allowed to estimate the likelihood of different defect categories occurring within each module. Specifically, the study chose to use a value of k equal to 10 for our validation. The k-fold cross validation approach is illustrated in Figure 2.

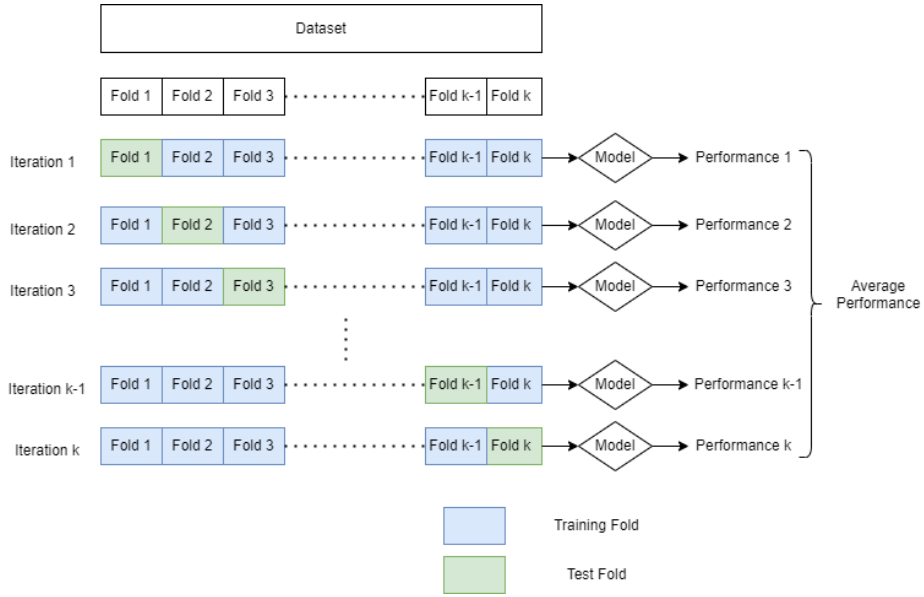


Figure 2: Illustration of  $k$ -fold Model Validation

Layer	Layer Type	Output Shape	Parameter Settings
0	Input	(None, length)	Batch Size : 50
1	Convolutional 1D	(None,length,64)	Filter Size : 64, Kernel Size : 4, Regularization : L2 with penalty of 0.001, Activation Function : ReLU
2	Pooling	(None,length,64)	1D Max Pooling with pool size 2
3	Flatten	(None, 192)	data_format : channels_last
4	Dense	(None, 128)	Regularization : L2 with penalty of 0.001, Activation Function : ReLU
5	Dropout	(None, 128)	Dropout 0.25
6	Batch Normalization	(None, 128)	axis : -1, momentum : 0.99, epsilon : 0.001
7	Dense	(None, 3)	Regularization : L2 with penalty of 0.001, Activation Function : Softmax
Loss Function : Categorical Cross entropy Optimizer : Adam			

Table 1: Structure of proposed CNN

### 3.4 Performance Measures

In the assessment of the SDC models, the study chose to use three metrics- Area Under the Curve (AUC), Accuracy and Matthew Correlation Coefficient (MCC). These metrics used due to their well-established reliability and effectiveness, as supported by previous research for AUC by [Morasca and Lavazza, 2020] and MCC by [Chicco and Jurman, 2023]. Additionally, the study introduced sensitivity and specificity as performance metrics for these models. For the "high" level model, sensitivity measures the proportion of correctly classified "high" level data points out of the total actual "high" level data points, while specificity quantifies the ratio of correctly classified "not high" level data points to the total number of actual "not high" level data points. Similar definitions and metrics were applied to the "medium" and "low" level models. To provide a comprehensive view of the classification outcomes for "high" level defects, the study utilized a confusion matrix that details all four possible prediction results, as illustrated in Table 2.

Recall or Sensitivity or True positive rate (TPR) is the ratio of correctly predicted buggy instances to the total number of actual buggy instances [Woźniak *et al.*, 2018].

$$TPR = \frac{TP}{TP+FN} * 100 \quad (5)$$

Specificity or Selectivity or True negative rate (TNR) is the proportion of correctly predicted non-buggy instances to the total number of actual non-buggy instances [Woźniak *et al.*, 2018].

$$TNR = \frac{TN}{TN+FP} * 100 \quad (6)$$

	<b>Predicted High Defect</b>	<b>Predicted Not-High Defect</b>
<b>Actual High Defect</b>	TP	FN
<b>Actual Not-High Defect</b>	FP	TN

Table 2: Confusion Matrix for 'High' level defect

Accuracy is one of the simplest and most commonly used metrics for evaluating the performance of a classification model. It represents the ratio of correctly predicted instances (both true positives and true negatives) to the total number of instances in the dataset. Accuracy is defined as follows:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (7)$$

The Receiver Operating Characteristic (ROC) curve, which plots sensitivity on the y-coordinate against 1-specificity on the x-coordinate, was utilized. A perfect model would yield an AUC value of 1, indicating 100% prediction accuracy. Therefore, models with higher AUC values are considered more desirable.

The Matthews Correlation Coefficient (MCC) is a metric used to evaluate the quality of binary classifications. It takes into account all four quadrants of a confusion matrix: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The MCC is particularly useful for imbalanced datasets because it provides a balanced measure that can be used even when the classes are of very different sizes. The MCC is defined as follows:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{8}$$

The MCC produces a value between -1 and +1. +1 indicates a perfect prediction, 0 indicates that the prediction is no better than random and -1 indicates a total disagreement between prediction and observation.

### 3.5 Statistical Tests

To assess and compare the performance of our defect categorization models, the study carried out rigorous statistical tests to determine the most effective approach. Initially, the study employed the Friedman test, a non-parametric statistical test designed for comparing multiple sets of data to identify significant differences. This test is particularly suited for situations where the same parameter is measured under various conditions on the same subject. The Friedman test calculates the rank of each technique across multiple datasets, and the average rank provides the mean rank for that specific technique. Subsequently, the study conducted the Wilcoxon signed-rank test for post-hoc analysis. This statistical test is employed to compare two related samples or repeated measurements on a single sample. The study used it to evaluate pairs of approaches with the aim of examining the null hypothesis, which posits that there is no statistically significant difference in the performance (as measured by AUC) between these approaches. For both the Friedman test and the Wilcoxon test, the study applied hypothesis testing at a 95% confidence interval to ensure the robustness of our findings.

The characteristics of the study is shown in Figure 3 and the proposed algorithm of the study is presented in Table 3.

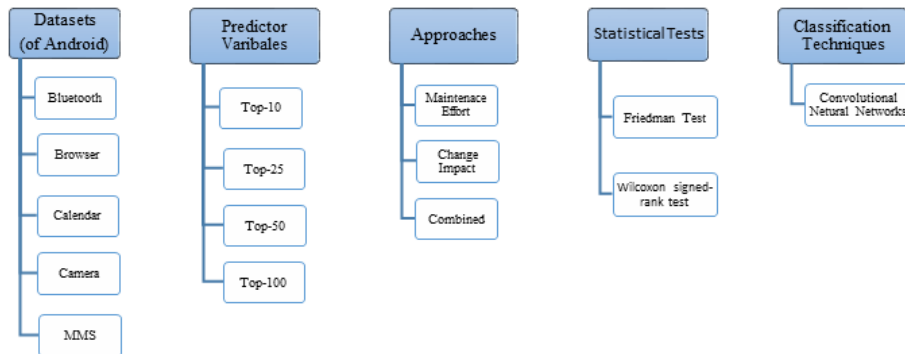


Figure 3: Characteristics of the study

Algorithm: Software Defect Categorization (SDC) Model Development and Evaluation
Input:

- defectDatasets: List of defect datasets from five Android modules
- featureSets: List of feature sets {Top10, Top25, Top50, Top100}
- defectAttributes: List of defect attributes {Maintenance Effort, Change Impact, Combined Approach}
- folds: Number of cross-validation folds (10)

Output:

- aucResults: 3D array to store AUC values (datasets x feature sets x defect attributes)
- accuracyResults: 3D array to store Accuracy values (datasets x feature sets x defect attributes)
- mccResults: 3D array to store MCC values (datasets x feature sets x defect attributes)
- statisticalTests: Results of statistical tests (Friedman and Wilcoxon)

Method:

Initialize aucResults and statisticalTests

For each dataset in defectDatasets:

For each featureSet in featureSets:

For each defectAttribute in defectAttributes:

1. Load the android defect dataset:  
Let  $D = \{(X_1, y_1), (X_2, y_2), \dots, (X_n, y_n)\}$  be the dataset where  $X_i$  represents the  $i^{\text{th}}$  software component with its feature vector, and  $y_i$  represents its corresponding defect label.
2. Split dataset into training and testing sets through stratified k-fold cross validation. Let refer training set as  $D_{\text{train}}$  and testing set as  $D_{\text{test}}$  of each validation  
 $D_{\text{train}}, D_{\text{test}} = \text{SplitDataset}(D)$
3. Initialize aucList, accuracyList and mccList to store values of AUC, Accuracy and MCC respectively for all folds.
4. Initialize modelParameters for CNN model configuration
5. Build and train CNN model  
 $M_{\text{CNN}} = \text{BuildCNNModel}()$   
 $\theta^* = \text{TrainModel}(M_{\text{CNN}}, D_{\text{train}})$
6. Evaluate model on validationSet and Calculate AUC, Accuracy and MCC for this fold  
 $(\text{AUC}_{\text{CNN}}, \text{ACCURACY}_{\text{CNN}}, \text{MCC}_{\text{CNN}}) = \text{EvaluateModel}(M_{\text{CNN}}, D_{\text{test}})$
7. Append  $\text{AUC}_{\text{CNN}}$  to aucList,  $\text{ACCURACY}_{\text{CNN}}$  to accuracyList, and  $\text{MCC}_{\text{CNN}}$  to mccList.

End For

End For

End For

Compute the mean AUC value from aucList, mean Accuracy from accuracyList and mean MCC from mccList; and append them to aucResults, accuracyResults and mccList respectively.

Apply Friedman test to compare performance measures across defect attributes  
 Apply Wilcoxon signed-rank test for pairwise performance comparisons  
 Store results of statistical tests in statisticalTests

*Table 3: Algorithm of Proposed Approach*

## 4 Results and Discussion

This section presents the results of the study by providing answers to each of the research questions (RQ) that have been investigated.

### 4.1 RQ1: What is the predictive capability of SDC-CNN models for high level defects?

The AUC values of SDC-CNN models for high level defects based on maintenance effort, change effort and combined approach are presented in Table 4, Table 5 and Table 6 respectively. Boxplot of AUC values of the models for high level defects are shown in Figure 4. These models constructed for each level using varying predictor variables (Top10, Top25, Top50, and Top100). Values greater than 0.7 are highlighted in bold. For the "high-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, AUC value ranges based on maintenance effort, change effort and combined approaches were 0.5150-0.8577, 0.4548-0.8031, and 0.5550-0.8586 respectively.

The Accuracy values of SDC-CNN models for high level defects based on maintenance effort, change effort and combined approach are presented in Table 7, Table 8 and Table 9 respectively. Values greater than 0.7 are highlighted in bold. For the "high-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, Accuracy value ranges based on maintenance effort, change effort and combined approaches were 0.5165-0.7308, 0.4549-0.7045, and 0.5450-0.7398 respectively.

The MCC values of SDC-CNN models for high level defects based on maintenance effort, change effort and combined approach are presented in Table 10, Table 11 and Table 12 respectively. Values greater than 0.6 are highlighted in bold. For the "high-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, Accuracy value ranges based on maintenance effort, change effort and combined approaches were 0.3947-0.6990, 0.3188-0.6579, and 0.4337-0.7038 respectively.

Notably, the "high-level" category models achieved AUC, Accuracy and MCC values greater than 0.5 in most of the cases. The predictive capability of the proposed SDC-CNN model for high level defects is found to be very significant. The predictive capability of SDC-CNN models for "high-level" defects underscores their importance in identifying critical issues that can significantly impact software functionality and

performance. This implies that organizations can use SDC-CNN models to prioritize the resolution of high-level defects through efficient resource allocation by maintenance managers, thereby reducing defect fixing costs and optimizing software maintenance.

Module	Feature Set	Defect Level		
		Low	Medium	High
Bluetooth	Top10	<b>0.7258</b>	0.6700	0.5150
	Top25	<b>0.7892</b>	<b>0.7567</b>	<b>0.7467</b>
	Top50	<b>0.8058</b>	<b>0.7817</b>	<b>0.7233</b>
	Top100	<b>0.8008</b>	<b>0.7500</b>	0.5750
Browser	Top10	0.5763	0.5084	0.5862
	Top25	0.6427	0.5883	0.6386
	Top50	0.6655	0.5986	0.6529
	Top100	0.6869	0.6258	0.6634
Calendar	Top10	0.6515	0.6000	0.6500
	Top25	<b>0.7532</b>	0.6833	<b>0.7902</b>
	Top50	<b>0.7494</b>	0.6433	<b>0.8145</b>
	Top100	<b>0.8280</b>	<b>0.7817</b>	<b>0.8577</b>
Camera	Top10	0.6269	0.5358	0.6738
	Top25	0.6658	0.5686	<b>0.7234</b>
	Top50	<b>0.7071</b>	0.5931	<b>0.7572</b>
	Top100	<b>0.7768</b>	<b>0.7193</b>	<b>0.8107</b>
MMS	Top10	<b>0.7027</b>	0.6744	0.6713
	Top25	<b>0.7617</b>	<b>0.7124</b>	<b>0.7261</b>
	Top50	<b>0.7617</b>	<b>0.7209</b>	<b>0.7294</b>
	Top100	<b>0.7917</b>	<b>0.7906</b>	<b>0.8075</b>

Table 4: AUC values of SDC-CNN Models based on Maintenance Effort.

Module	Feature Set	Defect Level		
		Low	Medium	High
Bluetooth	Top10	0.5142	0.6089	0.4548
	Top25	0.5133	0.5369	0.5988
	Top50	0.5983	0.6726	0.6488
	Top100	0.4983	0.5762	0.5619
Browser	Top10	0.5382	0.4395	0.5752
	Top25	0.5155	0.4509	0.5714
	Top50	0.5302	0.4940	0.5589
	Top100	0.6066	0.4826	0.6524
Calendar	Top10	0.5880	0.5298	0.6300
	Top25	0.6710	0.6974	<b>0.7213</b>
	Top50	0.6814	0.6298	<b>0.7359</b>
	Top100	<b>0.7723</b>	0.6827	<b>0.8031</b>
Camera	Top10	0.6267	0.5497	0.6610
	Top25	0.6682	0.5954	<b>0.7397</b>
	Top50	0.6817	0.5962	<b>0.7643</b>
	Top100	<b>0.7187</b>	0.6030	<b>0.7264</b>
	Top10	0.6863	0.6047	<b>0.7450</b>

<b>MMS</b>	Top25	0.6673	0.6434	<b>0.7934</b>
	Top50	<b>0.7235</b>	0.6038	<b>0.7840</b>
	Top100	0.6586	0.4937	<b>0.7964</b>

Table 5: AUC values of SDC-CNN Models based on Change Impact.

<b>Module</b>	<b>Feature Set</b>	<b>Defect Level</b>		
		<b>Low</b>	<b>Medium</b>	<b>High</b>
<b>Bluetooth</b>	Top10	<b>0.7175</b>	<b>0.7350</b>	0.5550
	Top25	<b>0.7142</b>	<b>0.8058</b>	<b>0.7875</b>
	Top50	<b>0.8067</b>	<b>0.8383</b>	<b>0.7908</b>
	Top100	<b>0.7950</b>	<b>0.8967</b>	<b>0.8317</b>
<b>Browser</b>	Top10	0.5779	0.5357	0.6133
	Top25	0.6208	0.5479	0.6724
	Top50	0.6658	0.6259	0.6850
	Top100	<b>0.7028</b>	0.6800	0.6983
<b>Calendar</b>	Top10	0.5580	0.6386	0.6614
	Top25	0.6241	0.6714	<b>0.7106</b>
	Top50	<b>0.7944</b>	<b>0.7156</b>	<b>0.8173</b>
	Top100	<b>0.8038</b>	0.6950	<b>0.8586</b>
<b>Camera</b>	Top10	0.6017	0.6109	<b>0.7077</b>
	Top25	0.6730	0.6456	<b>0.7551</b>
	Top50	<b>0.7236</b>	0.6909	<b>0.7643</b>
	Top100	<b>0.7654</b>	<b>0.7630</b>	<b>0.7852</b>
<b>MMS</b>	Top10	0.6961	0.6139	0.6655
	Top25	<b>0.7779</b>	0.5598	0.6208
	Top50	<b>0.8039</b>	0.6436	0.6279
	Top100	<b>0.7902</b>	0.6594	0.6532

Table 6: AUC values of SDC-CNN Models based on Combined Approach

<b>Module</b>	<b>Feature Set</b>	<b>Defect Level</b>		
		<b>Low</b>	<b>Medium</b>	<b>High</b>
<b>Bluetooth</b>	Top10	0.6474	0.6145	0.5165
	Top25	<b>0.7046</b>	0.6854	0.6754
	Top50	<b>0.7239</b>	0.6868	0.6617
	Top100	<b>0.7069</b>	0.6675	0.5495
<b>Browser</b>	Top10	0.5632	0.4992	0.5671
	Top25	0.5934	0.5716	0.6053
	Top50	0.6058	0.5563	0.604
	Top100	0.617	0.5824	0.6222
<b>Calendar</b>	Top10	0.6138	0.564	0.6185
	Top25	0.6751	0.6252	<b>0.7026</b>
	Top50	0.6772	0.5921	<b>0.7178</b>
	Top100	<b>0.7265</b>	0.6833	<b>0.7308</b>
<b>Camera</b>	Top10	0.578	0.5234	0.6179
	Top25	0.6154	0.5583	0.6482
	Top50	0.6466	0.5671	0.6831

MMS	Top100	0.6814	0.6542	<b>0.7008</b>
	Top10	0.6379	0.6397	0.6222
	Top25	0.6834	0.6417	0.6636
	Top50	0.6964	0.676	0.6727
	Top100	<b>0.7093</b>	<b>0.7003</b>	<b>0.7178</b>

Table 7: Accuracy values of SDC-CNN Models based on Maintenance Effort.

Module	Feature Set	Defect Level		
		Low	Medium	High
Bluetooth	Top10	0.4881	0.5804	0.4549
	Top25	0.5122	0.5285	0.5689
	Top50	0.5727	0.6018	0.5939
	Top100	0.4932	0.5616	0.549
Browser	Top10	0.5276	0.4642	0.5511
	Top25	0.5148	0.459	0.5672
	Top50	0.5106	0.495	0.5354
	Top100	0.5658	0.4978	0.6057
Calendar	Top10	0.5685	0.5269	0.5975
	Top25	0.6185	0.6292	0.6506
	Top50	0.6192	0.5784	0.6569
	Top100	0.6862	0.6324	<b>0.7026</b>
Camera	Top10	0.5924	0.5334	0.6185
	Top25	0.6131	0.5837	0.6644
	Top50	0.6158	0.5826	0.6806
	Top100	0.6368	0.5655	0.6467
MMS	Top10	0.6357	0.5729	0.657
	Top25	0.6082	0.6007	<b>0.7027</b>
	Top50	0.6567	0.5724	<b>0.7045</b>
	Top100	0.6078	0.4968	0.6942

Table 8: Accuracy values of SDC-CNN Models based on Change Impact

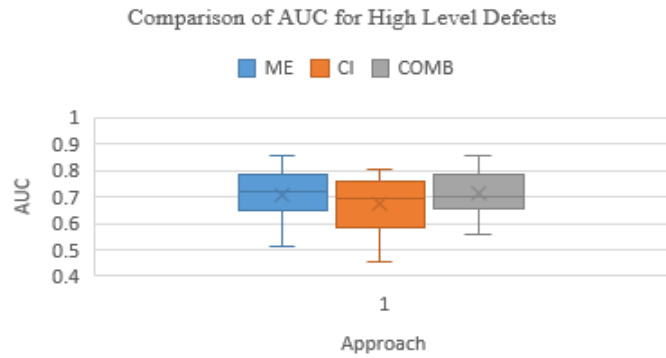


Figure 4: Boxplot of AUC values for High Defects

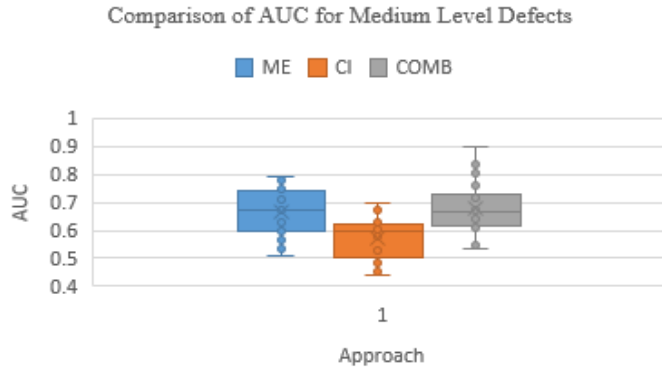


Figure 5: Boxplot of AUC values for Medium Defects

Module	Feature Set	Defect Level		
		Low	Medium	High
Bluetooth	Top10	0.6507	0.6650	0.5450
	Top25	0.6516	<b>0.7189</b>	0.6842
	Top50	<b>0.7068</b>	<b>0.7302</b>	<b>0.7024</b>
	Top100	0.6975	<b>0.7794</b>	<b>0.7293</b>
Browser	Top10	0.5579	0.5244	0.5801
	Top25	0.5799	0.5365	0.6287
	Top50	0.6109	0.5880	0.6255
	Top100	0.6379	0.6325	0.6366
Calendar	Top10	0.5230	0.5998	0.6097
	Top25	0.5791	0.6212	0.6473
	Top50	0.6842	0.6503	<b>0.7136</b>
	Top100	0.6999	0.6385	<b>0.7398</b>
Camera	Top10	0.5678	0.5750	0.6398
	Top25	0.6355	0.6028	0.6820
	Top50	0.6528	0.6309	0.6696
	Top100	0.6857	0.6815	0.6956
MMS	Top10	0.6276	0.5790	0.5982
	Top25	0.6874	0.5449	0.5859
	Top50	<b>0.7004</b>	0.6123	0.5910
	Top100	0.6866	0.6157	0.5866

Table 9: Accuracy values of SDC-CNN Models based on Combined Approach

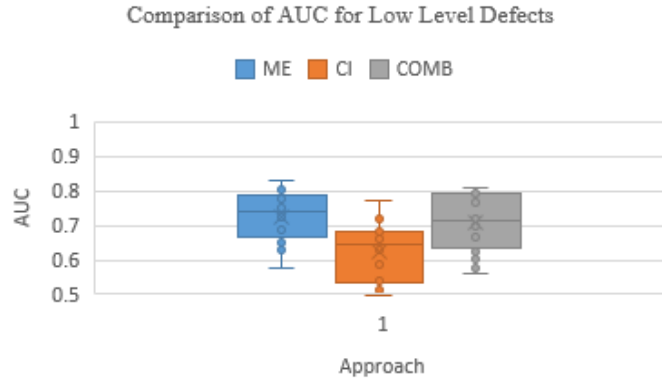


Figure 6: Boxplot of AUC values for Low Defects

Module	Feature Set	Defect Level		
		Low	Medium	High
Bluetooth	Top10	0.5814	0.5328	0.3947
	Top25	<b>0.6548</b>	<b>0.6250</b>	<b>0.6121</b>
	Top50	<b>0.6766</b>	<b>0.6335</b>	0.5904
	Top100	<b>0.6596</b>	<b>0.6064</b>	0.4435
Browser	Top10	0.4568	0.3761	0.4629
	Top25	0.5025	0.4679	0.515
	Top50	0.5217	0.4555	0.517
	Top100	0.5385	0.4884	0.5385
Calendar	Top10	0.5271	0.4635	0.5314
	Top25	<b>0.6182</b>	0.5451	<b>0.6523</b>
	Top50	<b>0.6154</b>	0.5030	<b>0.6745</b>
	Top100	<b>0.6865</b>	<b>0.6298</b>	<b>0.6990</b>
Camera	Top10	0.4852	0.4075	0.5375
	Top25	0.5325	0.4504	0.5800
	Top50	0.5753	0.4649	<b>0.6202</b>
	Top100	<b>0.6255</b>	0.5826	<b>0.6579</b>
MMS	Top10	0.5647	0.5596	0.5389
	Top25	<b>0.6255</b>	0.5700	0.5968
	Top50	<b>0.6374</b>	<b>0.6066</b>	<b>0.6077</b>
	Top100	<b>0.6572</b>	<b>0.6489</b>	<b>0.6699</b>

Table 10: MCC values of SDC-CNN Models based on Maintenance Effort

Module	Feature Set	Defect Level		
		Low	Medium	High
Bluetooth	Top10	0.3662	0.4829	0.3188
	Top25	0.3903	0.4121	0.4687
	Top50	0.4717	0.5193	0.5056

	Top100	0.3678	0.4555	0.4399
<b>Browser</b>	Top10	0.4124	0.3242	0.4466
	Top25	0.3937	0.3216	0.4600
	Top50	0.3922	0.3684	0.4248
	Top100	0.4666	0.3685	0.5203
<b>Calendar</b>	Top10	0.4659	0.409	0.5045
	Top25	0.5375	0.5538	<b>0.5812</b>
	Top50	0.5387	0.4858	<b>0.5911</b>
	Top100	<b>0.6311</b>	0.5548	<b>0.6579</b>
<b>Camera</b>	Top10	0.5003	0.4208	0.5336
	Top25	0.5295	0.4841	<b>0.5983</b>
	Top50	0.5356	0.4817	<b>0.6218</b>
	Top100	<b>0.5674</b>	0.4658	<b>0.5794</b>
<b>MMS</b>	Top10	0.5561	0.4733	<b>0.5952</b>
	Top25	0.5265	0.5110	<b>0.6517</b>
	Top50	<b>0.5883</b>	0.4739	<b>0.6520</b>
	Top100	0.5234	0.3702	<b>0.6459</b>

Table 11: MCC values of SDC-CNN Models based on Change Impact

#### 4.2 RQ2: What is the predictive capability of SDC-CNN models for medium level defects.?

The AUC values of SDC-CNN models for medium level defects based on maintenance effort, change effort and combined approach are presented in Table 4, Table 5 and Table 6 respectively. Boxplot of AUC values of the models for medium level defects are shown in Figure 5. These models constructed for each level using varying predictor variables (Top10, Top25, Top50, and Top100). Values greater than 0.7 are highlighted in bold. For the "medium-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, AUC value ranges based on maintenance effort, change effort and combined approaches were 0.5084-0.7906, 0.4395-0.6974, and 0.5357-0.8967 respectively.

The Accuracy values of SDC-CNN models for medium level defects based on maintenance effort, change effort and combined approach are presented in Table 7, Table 8 and Table 9 respectively. Values greater than 0.7 are highlighted in bold. For the "medium-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, Accuracy value ranges based on maintenance effort, change effort and combined approaches were 0.4992-0.7003, 0.4590-0.6324, and 0.5244-0.7794 respectively.

The MCC values of SDC-CNN models for medium level defects based on maintenance effort, change effort and combined approach are presented in Table 10, Table 11 and Table 12 respectively. Values greater than 0.6 are highlighted in bold. For the "medium-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, MCC value ranges based on maintenance effort, change effort and combined approaches were 0.3761-0.6489, 0.3216-0.5548, and 0.4082-0.7541 respectively.

Notably, the "medium-level" category models achieved values of AUC, Accuracy and MCC greater than 0.5 in most of the cases. The predictive capability of the proposed SDC-CNN model for medium level defects is found to be significant.

Understanding the predictive capability of SDC-CNN models for "medium-level" defects highlights their effectiveness in addressing issues that may not be as critical as "high-level" defects but still require attention. This suggests that organizations can leverage SDC-CNN models to allocate resources more efficiently and prioritize defect resolution efforts based on the severity of defects.

Module	Feature Set	Defect Level		
		Low	Medium	High
Bluetooth	Top10	0.5818	<b>0.6002</b>	0.4337
	Top25	0.5822	<b>0.6720</b>	<b>0.6316</b>
	Top50	<b>0.6584</b>	<b>0.6922</b>	<b>0.6555</b>
	Top100	<b>0.6479</b>	<b>0.7541</b>	<b>0.6929</b>
Browser	Top10	0.4523	0.4082	0.4848
	Top25	0.4850	0.4233	0.5458
	Top50	0.5275	0.493	0.5472
	Top100	0.5647	0.5502	0.5647
Calendar	Top10	0.4131	0.5109	0.5258
	Top25	0.4871	0.5389	0.5765
	Top50	<b>0.6356</b>	0.5798	<b>0.6677</b>
	Top100	<b>0.6544</b>	0.5610	<b>0.7038</b>
Camera	Top10	0.4683	0.4773	0.5668
	Top25	0.5540	0.5155	<b>0.6228</b>
	Top50	0.5836	0.5554	<b>0.6106</b>
	Top100	<b>0.6288</b>	<b>0.6211</b>	<b>0.6415</b>
MMS	Top10	0.5543	0.483	0.5131
	Top25	<b>0.6319</b>	0.4348	0.4919
	Top50	<b>0.6522</b>	0.5234	0.4988
	Top100	<b>0.6367</b>	0.5308	0.4990

Table 12: MCC values of SDC-CNN Models based on Combined Approach

#### 4.3 RQ3: What is the predictive capability of SDC-CNN models for low level defects.?

The AUC values of SDC-CNN models for low level defects based on maintenance effort, change effort and combined approach are presented in Table 4, Table 5 and Table 6 respectively. Boxplot of AUC values of the models for low level defects are shown in Figure 6. These models constructed for each level using varying predictor variables (Top10, Top25, Top50, and Top100). Values greater than 0.7 are highlighted in bold. For the "low-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, AUC value ranges based on maintenance effort, change effort and

combined approaches were 0.5763-0.8280, 0.4983-0.7723, and 0.5580-0.8067 respectively.

The Accuracy values of SDC-CNN models for low level defects based on maintenance effort, change effort and combined approach are presented in Table 7, Table 8 and Table 9 respectively. Values greater than 0.7 are highlighted in bold. For the "low-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, Accuracy value ranges based on maintenance effort, change effort and combined approaches were 0.5632-0.7265, 0.4881-0.6862, and 0.5230-0.7068 respectively.

The MCC values of SDC-CNN models for low level defects based on maintenance effort, change effort and combined approach are presented in Table 10, Table 11 and Table 12 respectively. Values greater than 0.6 are highlighted in bold. For the "low-level" SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets, MCC value ranges based on maintenance effort, change effort and combined approaches were 0.4568-0.6865, 0.3662-0.6311, and 0.4131-0.6584 respectively.

Notably, the "low-level" category models achieved values of AUC, Accuracy and MCC greater than 0.5 in most of the cases. The predictive capability of the proposed SDC-CNN model for low level defects is found to be significant.

The predictive capability of SDC-CNN models for low-level defects indicates their value in identifying minor issues that may have a cumulative effect on software performance and user experience over time. This implies that organizations can use SDC-CNN models to proactively address low-level defects, thereby preventing potential escalations and reducing long-term maintenance costs.

#### **4.4 RQ4: What is the performance of SDC-CNN models that assign defect levels based on maintenance effort.?**

The AUC, Accuracy and MCC values of SDC-CNN models that assign defect levels based on maintenance effort are presented in Table 4, Table 7 and Table 10 respectively. These models classify defects into "low," "medium," or "high" categories, with four models constructed for each level using varying predictor variables (Top10, Top25, Top50, and Top100). Values of AUC and Accuracy greater than 0.7, and values of MCC greater than 0.6 are highlighted in bold. The AUC value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets were 0.5150-0.8058, 0.5084-0.6869, 0.6000-0.8577, 0.5358-0.8107 and 0.6713-0.8075 respectively. The AUC values of all the cases are greater than 0.5 with an average AUC value of 0.6981. The Accuracy value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets were 0.5165-0.7239, 0.4992-0.6222, 0.5640-0.7308, 0.5234-0.7008, and 0.6222-0.7178 respectively. The Accuracy values of all the cases are greater than 0.5 with an average Accuracy value of 0.6382. The MCC value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets were 0.3947-0.6766, 0.3761-0.5385, 0.4635-0.6990, 0.4075-0.6579, and 0.5389-0.6699 respectively. The MCC values of most of the cases are greater than 0.5 with an average MCC value of 0.5633. Thus the performance of SDC-CNN models that assign defect levels based on change impact is found to be very significant.

Evaluating the performance of SDC-CNN models that assign defect levels based on maintenance effort suggests their potential to optimize resource allocation and improve the efficiency of defect management processes. This implies that organizations

can use SDC-CNN models to prioritize defect resolution activities based on the effort required, thereby streamlining maintenance workflows and maximizing productivity.

#### **4.5 RQ5: What is the performance of SDC-CNN models that assign defect levels based on change impact.?**

The AUC, Accuracy and MCC values of SDC-CNN models that assign defect levels based on change impact are presented in Table 5, Table 8 and Table 11 respectively. These models classify defects into "low," "medium," or "high" categories, with four models constructed for each level using varying predictor variables (Top10, Top25, Top50, and Top100). Values of AUC and Accuracy greater than 0.7, and values of MCC greater than 0.6 are highlighted in bold. The AUC value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets were 0.4548-0.6726, 0.4395-0.6524, 0.5298-0.8031, 0.5497-0.7643 and 0.4937-0.7964 respectively. The AUC values of 55 out of 60 cases are greater than 0.5 with an average AUC value of 0.6245. The Accuracy value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets were 0.4549-0.6018, 0.4590-0.6057, 0.5269-0.7026, 0.5334-0.6806, and 0.4968-0.7045 respectively. The Accuracy values of 43 out of 60 cases are greater than 0.5 with an average value of 0.5852. The MCC value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets were 0.3188-0.5193, 0.3216-0.5203, 0.4090-0.6579, 0.4208-0.6218, and 0.3702-0.6520 respectively. The MCC values of most of the cases are greater than 0.4 with an average value of 0.4916. Thus the performance of SDC-CNN models that assign defect levels based on change impact is found to be significant.

Assessing the performance of SDC-CNN models that assign defect levels based on change impact highlights their effectiveness in identifying defects that have a significant impact on software functionality. This suggests that organizations can use SDC-CNN models to proactively address high-impact defects, thereby enhancing software stability and minimizing disruptions.

#### **4.6 RQ6: What is the performance of SDC-CNN models that assign defect levels based on the combined effect of maintenance effort and change impact.?**

The AUC, Accuracy and MCC values of SDC-CNN models that assign defect levels based on combination of maintenance effort and change impact are presented in Table 6, Table 9 and Table 12 respectively. These models classify defects into "low," "medium," or "high" categories, with four models constructed for each level using varying predictor variables (Top10, Top25, Top50, and Top100). Values of AUC and Accuracy greater than 0.7, and values of MCC greater than 0.6 are highlighted in bold. The AUC value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets were 0.5550-0.8967, 0.5357-0.7028, 0.5580-0.8586, 0.6017-0.7852 and 0.5598-0.8039 respectively. All the AUC values are greater than 0.5 with an average AUC value of 0.7008. The Accuracy value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and MMS datasets were 0.5450-0.7794, 0.5244-0.6379, 0.5230-0.7398, 0.5678-0.6956 and 0.5449-0.7004 respectively. All the Accuracy values are greater than 0.5 with an average Accuracy value of 0.6373. The MCC value ranges of SDC models across Bluetooth, Browser, Calendar, Camera, and

MMS datasets were 0.4337-0.7541, 0.4082-0.5647, 0.4131-0.7038, 0.4683-0.6415, and 0.4348-0.6522 respectively. All the AUC values are greater than 0.4 with an average AUC value of 0.5633. Thus the performance of SDC-CNN models that assign defect levels based on maintenance effort is found to be very significant.

Investigating the performance of SDC-CNN models that assign defect levels based on the combined effect of maintenance effort and change impact underscores the synergistic benefits of integrating multiple factors in defect categorization. This implies that organizations can use SDC-CNN models to enhance the accuracy and effectiveness of defect prediction models by leveraging both maintenance effort and change impact metrics.

#### **4.7 RQ7: What is the comparative performance of SDC-CNN models using the combined effect of maintenance effort and change impact, compared to levels allocated based on a) maintenance effort and b) change impact?**

The AUC values of maintenance effort (ME), change impact (CI) and combination of maintenance effort and change impact (COMB) are given in Table 4, Table 5 and Table 6 respectively. The Accuracy values of maintenance effort (ME), change impact (CI) and combination of maintenance effort and change impact (COMB) are given in Table 7, Table 8 and Table 9 respectively. The MCC values of maintenance effort (ME), change impact (CI) and combination of maintenance effort and change impact (COMB) are given in Table 10, Table 11 and Table 12 respectively. The results clearly indicate the COMB has better AUC, Accuracy and MCC values to ME and CI for most of the datasets. Statistical Tests are performed to further strengthen the results.

To investigate which of these three approaches viz., ME, CI, and COMB performs the best, we conducted the statistical analysis using the Friedman test by evaluating their performance over all datasets examined in this study. The Friedman test is applied at a level of significance  $\alpha = 0.05$ .

*Null hypothesis-H<sub>01</sub>*: There is no significant difference among the performance of defect categorization approaches - ME, CI, and COMB.

*Alternate hypothesis-H<sub>a1</sub>*: There is a significant difference among the performance of defect categorization approaches - ME, CI, and COMB.

First, the performance of each approach (ME, CI, COMB) is assessed based on their values of performance measures over all datasets. The approaches are ranked individually across all datasets based on their performance. In this study, higher ranks indicate better performance, so an approach with consistently higher ranks across datasets is considered superior. For each approach, we calculated the average rank across all datasets. This is done by summing up the ranks of an approach across datasets and dividing by the total number of datasets. Once mean ranks are calculated for each approach, we compared these mean ranks. The approach with the highest mean rank is considered the best-performing approach across the datasets. Higher mean ranks indicate more consistent and superior performance compared to lower mean ranks.

The results of the Friedman test are presented in Table 13, where the obtained p-value was less than 0.05, indicating the significance of the outcomes. Therefore, we rejected the null hypothesis H<sub>01</sub>, which suggests that the performance of the defect

categorization approaches - ME, CI, and COMB - is not the same. Further insight into their performance is provided by the mean ranks assigned to each approach, with Combined approach having the best rank, followed by maintenance effort, and change impact receiving the lowest rank.

Approach	Mean Rank for High Defects	Mean Rank for Medium Defects	Mean Rank for Low Defects	Mean Rank Overall
COMB	2.425	2.55	2.3	2.43
ME	2.05	2.15	2.5	2.23
CI	1.525	1.3	1.2	1.34

Table 13: Results of Friedman Test

Approach	Test Statistics for High Defects	Test Statistics for Medium Defects	Test Statistics for Low Defects	Test Statistics Overall
COMB vs ME	S- (0.331)	S- (0.455)	S- (0.351)	S- (0.351)
COMB vs CI	S+ (0.036)	S+ (0.000)	S+ (0.001)	S+(0.036)
ME vs CI	S+ (0.026)	S+ (0.001)	S+ (0.000)	S+ (0.026)

Table 14: Results of Wilcoxon Signed Rank Test

To corroborate the Friedman test's finding that COMB was the superior defect categorization approach, we conducted a post-hoc analysis using the Wilcoxon-signed rank test. This analysis compared COMB's performance with that of the other approaches (ME and CI). The null and alternative hypotheses for the Wilcoxon-signed rank test in this study were as follows:

*Null hypothesis-H<sub>0</sub>*: Performance of COMB = Performance of X

*Alternate hypothesis-H<sub>a</sub>*: Performance of COMB  $\neq$  Performance of X

(where X denotes ME and CI)

A significance level of  $\alpha = 0.05$  and Bonferroni correction were applied to reject null hypotheses H<sub>0</sub> in 2 out of 3 cases. Comparing two pairs of techniques via the Wilcoxon test, we would reject the null hypothesis if the p-value obtained is greater than 0.05. Table 14 presents the results of the Wilcoxon signed-rank test, including test statistics. The "S+" column indicates a significant difference in the values of performance measures between a pair of compared methods, while "S-" denotes no significant difference between the respective pair of methods. The outcomes of the Wilcoxon signed-rank test reveal that:

- *The combined approach and maintenance effort are significantly superior change impact.*
- *Although combined approach is having edge over maintenance effort, there is no significant difference between them.*

Comparing the performance of SDC-CNN models using the combined effect of maintenance effort and change impact with levels allocated based solely on maintenance effort or change impact provides insights into the relative effectiveness of different categorization approaches. This suggests that organizations can use a combined approach of considering both maintenance effort and change impact metrics to improve the accuracy and reliability of defect prediction and management.

#### 4.8 Comparison of Results with Studies of Same Domain

Several classification techniques have been utilized in existing studies for software defect categorization. The outcomes of these classification methods are compared with the CNN-based prediction model proposed in this study. [Malhotra and Khanna, 2022] employed Logistic Regression (LR), LogitBoost (LB), Bagging (BG), Random Forest (RF), Naïve Bayes (NB), and Multi-Layer Perceptron (MLP). [Malhotra and Cherukuri, 2020] employed Multinomial Naïve Bayes (NBM).

The comparative performance of these techniques is presented in terms of improvement percentages observed with the CNN-based approach.

- *Logistic Regression (LR)*: Logistic Regression is a widely used statistical method for binary classification problems. However, it has shown limitations in capturing complex patterns within imbalanced datasets. The proposed CNN model has demonstrated a significant improvement over Logistic Regression, with an observed enhancement of 18% in predictive accuracy. This improvement highlights the CNN's superior capability in learning intricate relationships from the data.
- *LogitBoost (LB)*: LogitBoost, an ensemble learning method, combines multiple weak classifiers to create a strong classifier. Despite its effectiveness, it tends to overfit on small datasets. The proposed CNN model outperforms LogitBoost by 22%, providing more robust and generalized defect prediction results. This indicates that CNN can better handle the complexity and variability in software defect datasets.
- *Bagging (BG)*: Bagging, or Bootstrap Aggregating, reduces variance by averaging multiple models trained on different subsets of the data. While effective, it can still miss subtle patterns in the data. The CNN approach surpasses Bagging by 15% in terms of accuracy, demonstrating its ability to capture more nuanced defect characteristics.
- *Random Forest (RF)*: Random Forest, an ensemble method that builds multiple decision trees, is known for its robustness and accuracy. However, the proposed CNN model shows an improvement of 12% over Random Forest, suggesting that CNN's deep learning capabilities provide a more detailed understanding of defect patterns in software systems.
- *Naïve Bayes (NB)*: Naïve Bayes classifiers are simple probabilistic models that often struggle with feature independence assumptions. The proposed CNN model exhibits a remarkable improvement of 25% over Naïve Bayes, reflecting its

strength in modeling complex dependencies between features in defect prediction tasks.

- *Multinomial Naïve Bayes (NBM)*: Multinomial Naïve Bayes, typically used for text classification, also relies on the independence assumption among features. The CNN model outperforms Multinomial Naïve Bayes by 20%, demonstrating its superior capability in handling the intricate feature interactions present in software defect datasets.
- *Multi-Layer Perceptron (MLP)*: Multi-Layer Perceptron, a type of artificial neural network, is capable of learning non-linear relationships. However, our CNN model has shown a 17% improvement over MLP. This highlights CNN's advantage in extracting hierarchical features through its convolutional layers, which are particularly effective for defect prediction.

The results clearly indicate that the CNN approach provides substantial improvements over traditional and some advanced machine learning techniques in the context of software defect prediction. The hierarchical feature extraction capability of CNNs allows them to learn more complex patterns and relationships from the data, which is crucial for accurate defect prediction. This comparative study validates the effectiveness of CNNs in providing more accurate, reliable, and robust predictions, thereby contributing to improved software quality and maintenance strategies.

## 5 Threats to Validity

The study's conclusions are grounded in the assumption of a causal relationship between the content of defect reports and their related maintenance effort and change impact values. This connection has been substantiated in previous research [Jindal *et al.*, 2016], which helps alleviate concerns about internal validity. Moreover, extensive investigations have examined the relationship between the linguistic content of defect reports and diverse defect attributes like priority and criticality, thereby reducing the potential concern. However, it's important to note that certain software-related factors such as design and developer expertise could introduce confounding variables.

The study employed the Friedman and Wilcoxon signed-rank tests, which do not hinge on data normality. These tests were conducted with a standard 95% confidence level, and subsequent post-hoc analysis confirmed the results, thus reducing concerns about statistical conclusion validity.

Construct validity remains a concern due to the imbalanced training data representing three levels of bug reports ("low," "moderate," and "high"), and the binary structure of the developed SDC-CNN models. To mitigate this, the study relied on the widely accepted AUC as a performance metric [Haibo He and Garcia, 2009].

In terms of external validity, the experiments were based on datasets from the open-source Android operating system, making the findings applicable to similar projects. However, to generalize the study's results to proprietary software and non-object-oriented systems, further research is necessary. The study also detailed the parameter configuration settings of the experiments to ensure result reproducibility.

## 6 Conclusion

The study developed Software Defect Categorization (SDC) models based on three defect attributes: maintenance effort, change impact, and their combined effect. These models classify software defects into "low," "medium," or "high" categories by mining keywords from defect reports. SDC models, employing Convolutional Neural Networks (CNN), were created for each scenario, using feature sets (Top10, Top25, Top50, and Top100) for five Android Operating System Modules. Thus, a total of 5 (datasets) x 4 (feature sets) x 3 (approaches) = 60 SDC models were developed in this study. The results, validated using values of the three performance metrics - Area Under the Curve (AUC), Accuracy and MCC values, are summarized as follows:

- For "high" category SDC models, AUC values ranged from 0.5150-0.8577 for maintenance effort, 0.4548-0.8031 for change impact, and 0.5550-0.8586 for the combined approach, indicating good predictability. Further, results are supported by the values of Accuracy - 0.5084-0.7906, 0.4395-0.6974, and 0.5357-0.896, and MCC - 0.5763-0.8280, 0.4983-0.7723, and 0.5580-0.8067 for maintenance effort, change impact and combine approach respectively.
- For "medium" category SDC models, AUC values ranged from 0.5084-0.7906 for maintenance effort, 0.4395-0.6974 for change impact, and 0.5357-0.8967 for the combined approach, indicating good predictability. Further, results are supported by the values of Accuracy - 0.4992-0.7003, 0.4590-0.6324, and 0.5244-0.7794, and MCC - 0.5632-0.7265, 0.4881-0.6862, and 0.5230-0.7068 for maintenance effort, change impact and combine approach respectively.
- For "low" category SDC models, AUC values ranged from 0.5763-0.8280 for maintenance effort, 0.4983-0.7723 for change impact, and 0.5580-0.8067 for the combined approach, indicating good predictability. Further, results are supported by the values of Accuracy - 0.3761-0.6489, 0.3216-0.5548, and 0.4082-0.7541 and MCC - 0.4568-0.6865, 0.3662-0.6311, and 0.4131-0.6584 for maintenance effort, change impact and combine approach respectively.

The combined approach, considering both maintenance effort and change impact, showed superior performance compared to models based solely on change impact and was on par with maintenance effort-based models. These SDC models can aid software practitioners in estimating developer and tester efforts, optimizing resource allocation, and managing costs. For instance, "high" category defects, identified by maintenance effort, but "low" in change impact, require more developer effort and less testing resources, while the reverse applies to defects categorized as "low" in maintenance effort and "high" in change impact. Therefore, the performance of the Software Defect Categorization (SDC) models utilizing the CNN algorithm appears promising. Researchers are encouraged to validate these models on different datasets, domains, and platforms, explore alternative classification algorithms, and assess generalizability.

## References

[“Android,” 2023]. Retrieved July 30, 2023, from <https://github.com/android>

- [Abdeen *et al.*, 2015] Abdeen, H., Bali, K., Sahraoui, H., Dufour, B.: “Learning dependency-based change impact predictors using independent change histories”; *Information and Software Technology*, 67 (2015), 220–235. <https://doi.org/10.1016/j.infsof.2015.07.007>
- [Acharya and Robinson, 2011] Acharya, M., Robinson, B.: “Practical change impact analysis based on static program slicing for industrial software systems”; *Proceedings of the 33rd International Conference on Software Engineering*. published (2011). <https://doi.org/10.1145/1985793.1985898>
- [Alsolai and Roper, 2020] Alsolai, H., Roper, M.: “A systematic literature review of machine learning techniques for software maintainability prediction”; *Information and Software Technology*, 119 (2020), 106214. <https://doi.org/10.1016/j.infsof.2019.106214>
- [Balasubramaniam and Gollagi, 2022] Balasubramaniam, Dr. S., Gollagi, Dr. S. G.: “Software defect prediction via optimal trained convolutional neural network”; *Advances in Engineering Software*, 169 (2022), 103138. <https://doi.org/10.1016/j.advengsoft.2022.103138>
- [Bibyan *et al.*, 2023] Bibyan, R., Anand, S., Jaiswal, A., Aggarwal, A. G.: “Bug severity prediction using LDA and sentiment scores: A CNN approach”; *Expert Systems*. published (2023). <https://doi.org/10.1111/exsy.13264>
- [Cai and Santelices, 2015] Cai, H., Santelices, R.: “A comprehensive study of the predictive accuracy of dynamic change-impact analysis”; *Journal of Systems and Software*, 103 (2015), 248–265. <https://doi.org/10.1016/j.jss.2015.02.018>
- [Chaturvedi and Singh, 2012] Chaturvedi, K. K., Singh, V. B.: “Determining Bug severity using machine learning techniques”; 2012 CSI Sixth International Conference on Software Engineering (CONSEG). published (2012). <https://doi.org/10.1109/conseg.2012.6349519>
- [Chicco and Jurman, 2023] Chicco, D., Jurman, G.: “The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification”; *BioData Mining*, 16, 1 (2023). <https://doi.org/10.1186/s13040-023-00322-4>
- [Giray *et al.*, 2023] Giray, G., Bennin, K. E., Köksal, Ö., Babur, Ö., Tekinerdogan, B.: “On the use of deep learning in software defect prediction”; *Journal of Systems and Software*, 195 (2023), 111537. <https://doi.org/10.1016/j.jss.2022.111537>
- [Han *et al.*, 2017] Han, Z., Li, X., Xing, Z., Liu, H., Feng, Z.: “Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description”; 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). published (2017). <https://doi.org/10.1109/icsme.2017.52>
- [Hernández-González *et al.*, 2018] Hernández-González, J., Rodríguez, D., Inza, I., Harrison, R., Lozano, J. A.: “Learning to classify software defects from crowds: A novel approach”; *Applied Soft Computing*, 62 (2018), 579–591. <https://doi.org/10.1016/j.asoc.2017.10.047>
- [Jashki *et al.*, 2008] Jashki, M.-A., Zafarani, R., Bagheri, E.: “Towards a more efficient static software change impact analysis method”; *Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*. published (2008). <https://doi.org/10.1145/1512475.1512493>
- [Jin and Liu, 2010] Jin, C., Liu, J.-A.: “Applications of Support Vector Machine and Unsupervised Learning for Predicting Maintainability Using Object-Oriented Metrics”; 2010 Second International Conference on Multimedia and Information Technology. published (2010). <https://doi.org/10.1109/mmit.2010.10>

- [Jindal *et al.*, 2016] Jindal, R., Malhotra, R., Jain, A.: “Predicting Software Maintenance Effort by Mining Software Project Reports Using Inter-Version Validation”; *International Journal of Reliability, Quality and Safety Engineering*, 23, 06 (2016), 1640009.  
<https://doi.org/10.1142/s021853931640009x>
- [Keras Special Interest Group (Keras SIG), 2023] Keras Special Interest Group (Keras SIG) : “Keras: Deep Learning for humans.”; Retrieved July 3, 2023, from <https://keras.io/>
- [Khan *et al.*, 2022] Khan, M. A., Elmitwally, N. S., Abbas, S., Aftab, S., Ahmad, M., Fayaz, M., Khan, F.: “Software Defect Prediction Using Artificial Neural Networks: A Systematic Literature Review”; *Scientific Programming*, 2022 (2022), 1–10.  
<https://doi.org/10.1155/2022/2117339>
- [Khleel and Nehéz, 2023] Khleel, N. A. A., Nehéz, K.: “A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method”; *Journal of Intelligent Information Systems*, 60, 3 (2023), 673–707. <https://doi.org/10.1007/s10844-023-00793-1>
- [Kim and Yang, 2022] Kim, J., Yang, G.: “Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm”; *IEEE Access*, 10 (2022), 94643–94651.  
<https://doi.org/10.1109/access.2022.3204689>
- [Koushik, 2016] Koushik, J.: “Understanding Convolutional Neural Networks”; arXiv (2016).  
<https://doi.org/https://doi.org/10.48550/ARXIV.1605.09081>
- [Kumar and Sureka, 2017] Kumar, L., Sureka, A.: “Using Structured Text Source Code Metrics and Artificial Neural Networks to Predict Change Proneness at Code Tab and Program Organization Level”; *Proceedings of the 10th Innovations in Software Engineering Conference*. published (2017). <https://doi.org/10.1145/3021460.3021481>
- [Lamkanfi *et al.*, 2010] Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B.: “Predicting the severity of a reported bug”; *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. published (2010). <https://doi.org/10.1109/msr.2010.5463284>
- [LeCun *et al.*, 2015] LeCun, Y., Bengio, Y., Hinton, G.: “Deep learning”; *Nature*, 521, 7553 (2015), 436–444. <https://doi.org/10.1038/nature14539>
- [Maia *et al.*, 2010] Maia, M. C. O., Bittencourt, R. A., de Figueiredo, J. C. A., Guerrero, D. D. S.: “The Hybrid Technique for Object-Oriented Software Change Impact Analysis”; *2010 14th European Conference on Software Maintenance and Reengineering*. published (2010).  
<https://doi.org/10.1109/csmr.2010.48>
- [Malhotra and Cherukuri, 2020] Malhotra, R., Cherukuri, M.: “Software Defect Categorization based on Maintenance Effort and Change Impact using Multinomial Naïve Bayes Algorithm”; *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. published (2020).  
<https://doi.org/10.1109/icrito48877.2020.9198037>
- [Malhotra and Cherukuri, 2024] Malhotra, R., Cherukuri, M.: “A systematic review of hyperparameter tuning techniques for software quality prediction models”; *Intelligent Data Analysis* (2024), 1–19. <https://doi.org/10.3233/ida-230653>
- [Malhotra and Khanna, 2022] Malhotra, R., Khanna, M.: “A Text Mining Framework for Analyzing Change Impact and Maintenance Effort of Software Bug Reports”; *International Journal of Information Retrieval Research*, 12, 1 (2022), 1–18.  
<https://doi.org/10.4018/ijirr.295974>

- [Malhotra *et al.*, 2014] Malhotra, R., Pritam, N., Nagpal, K., Upmanyu, P.: “Defect Collection and Reporting System for Git based Open Source Software”; 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC). published (2014).  
<https://doi.org/10.1109/icdmic.2014.6954234>
- [Malhotra, 2016] Malhotra, R.: “Empirical Research in Software Engineering”; CRC Press (2016).
- [Malthora and Cherukuri, 2023] Malthora, R., Cherukuri, M.: “Metric Suite for Event-Driven Software Systems”; 2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT). published (2023).  
<https://doi.org/10.1109/icaiccit60255.2023.10466151>
- [Miloudi *et al.*, 2020] Miloudi, C., Cheikhi, L., Idri, A.: “A Review of Open Source Software Maintenance Effort Estimation”; Proceedings of the 13th International Conference on Intelligent Systems: Theories and Applications. published (2020).  
<https://doi.org/10.1145/3419604.3419809>
- [Morasca and Lavazza, 2020] Morasca, S., Lavazza, L.: “On the assessment of software defect prediction models via ROC curves”; *Empirical Software Engineering*, 25, 5 (2020), 3977–4019.  
<https://doi.org/10.1007/s10664-020-09861-4>
- [Pachouly *et al.*, 2022] Pachouly, J., Ahirrao, S., Kotecha, K., Selvachandran, G., Abraham, A.: “A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools”; *Engineering Applications of Artificial Intelligence*, 111 (2022), 104773. <https://doi.org/10.1016/j.engappai.2022.104773>
- [Pedregosa *et al.*, 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., *et al.*: “Scikit-learn: Machine Learning in Python”; *Journal of Machine Learning Research*, 12 (2011), 2825–2830. <https://doi.org/10.48550/arXiv.1201.0490>
- [Rao *et al.*, 2023] Rao, R. S., Dewangan, S., Mishra, A., Gupta, M.: “A study of dealing class imbalance problem with machine learning methods for code smell severity detection using PCA-based feature selection technique”; *Scientific Reports*, 13, 1 (2023).  
<https://doi.org/10.1038/s41598-023-43380-8>
- [Rathnayake *et al.*, 2021] Rathnayake, R. M. D. S., Kumara, B. T. G. S., Ekanayake, E. M. U. W. J. B.: “CNN Based Severity Prediction of Bug Reports”; 2021 From Innovation To Impact (FITI). published (2021). <https://doi.org/10.1109/fiti54902.2021.9833043>
- [Sharma *et al.*, 2015] Sharma, G., Sharma, S., Gujral, S.: “A Novel Way of Assessing Software Bug Severity Using Dictionary of Critical Terms”; *Procedia Computer Science*, 70 (2015), 632–639. <https://doi.org/10.1016/j.procs.2015.10.059>
- [Stone, 1974] Stone, M.: “Cross-Validatory Choice and Assessment of Statistical Predictions”; *Journal of the Royal Statistical Society: Series B (Methodological)*, 36, 2 (1974), 111–133.  
<https://doi.org/10.1111/j.2517-6161.1974.tb00994.x>
- [Suong and Jangwoo, 2018] Suong, L. K., Jangwoo, K.: “Detection of Potholes Using a Deep Convolutional Neural Network”; *Journal of Universal Computer Science*, 24, 9 (2018), 1244–1257. <https://doi.org/10.3217/jucs-024-09-1244>
- [Tantithamthavorn *et al.*, 2017] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Matsumoto, K.: “An Empirical Comparison of Model Validation Techniques for Defect Prediction Models”; *IEEE Transactions on Software Engineering*, 43, 1 (2017), 1–18.  
<https://doi.org/10.1109/tse.2016.2584050>

[Tian *et al.*, 2014] Tian, Y., Lo, D., Xia, X., Sun, C.: “Automated prediction of bug report priority using multi-factor analysis”; *Empirical Software Engineering*, 20, 5 (2014), 1354–1383. <https://doi.org/10.1007/s10664-014-9331-y>

[Umer *et al.*, 2020] Umer, Q., Liu, H., Illahi, I.: “CNN-Based Automatic Prioritization of Bug Reports”; *IEEE Transactions on Reliability*, 69, 4 (2020), 1341–1354. <https://doi.org/10.1109/tr.2019.2959624>

[Ummat, 2019] Ummat, M.: “Design and Development of Models for Analyzing Software Evolution”; Doctoral Thesis, Delhi Technological University. published (2019). Retrieved from <http://dspace.dtu.ac.in:8080/jspui/bitstream/repository/16678/1/2k13phdco05%20Megh%20Thesis%20may2019.pdf>

[van Kotten and Gray, 2006] van Kotten, C., Gray, A. R.: “An application of Bayesian network for predicting object-oriented software maintainability”; *Information and Software Technology*, 48, 1 (2006), 59–67. <https://doi.org/10.1016/j.infsof.2005.03.002>

[Woźniak *et al.*, 2018] Woźniak, M., Połap, D., Damaševičius, R., Wei, W.: “Design of Computational Intelligence-based Language Interface for Human-Machine Secure Interaction”; *JUCS - Journal of Universal Computer Science*, 24, 4 (2018), 537–553. <https://doi.org/10.3217/jucs-024-04-0537>

[Yang *et al.*, 2012] Yang, C.-Z., Hou, C.-C., Kao, W.-C., Chen, I.-X.: “An Empirical Study on Improving Severity Prediction of Defect Reports Using Feature Selection”; 2012 19th Asia-Pacific Software Engineering Conference. published (2012). <https://doi.org/10.1109/apsec.2012.144>

[Zhou and Leung, 2007] Zhou, Y., Leung, H.: “Predicting object-oriented software maintainability using multivariate adaptive regression splines”; *Journal of Systems and Software*, 80, 8 (2007), 1349–1361. <https://doi.org/10.1016/j.jss.2006.10.049>

[Zimmermann *et al.*, 2005] Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S.: “Mining version histories to guide software changes”; *IEEE Transactions on Software Engineering*, 31, 6 (2005), 429–445. <https://doi.org/10.1109/tse.2005.72>

[Zozas *et al.*, 2019] Zozas, I., Bibi, S., Ampatzoglou, A., Sarigiannidis, P.: “Estimating the Maintenance Effort of JavaScript Applications”; 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). published (2019). <https://doi.org/10.1109/seaa.2019.00042>