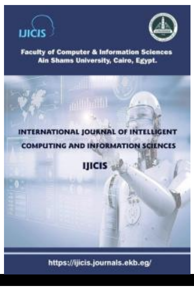
IJICIS, Vol.23, No.3, 31-58
DOI: 10.21608/ijicis.2023.200737.1261

International Journal of Intelligent Computing and Information Sciences

https://ijicis.journals.ekb.eg/



SOFTWARE DEFECT PREDICTION APPROACHES REVISITED

Khaled S. Shebl *
Information Systems,
Faculty of Computer and Information Sciences, Ain Shams University,
Cairo, Egypt
khaled.said@cis.asu.edu.eg

Yasmine M. Afify
Information Systems,
Faculty of Computer and Information Sciences, Ain Shams University,
Cairo, Egypt
yasmine.afify@cis.asu.edu.eg

Nagwa Badr
Information Systems,
Faculty of Computer and Information Sciences, Ain Shams University,
Cairo, Egypt
nagwabadr@cis.asu.edu.eg

Received 2023-03-17; Revised 2023-08-03; Accepted 2023-08-14

Abstract: *A crucial field in software development and testing is Software Defect Prediction (SDP) because the quality, dependability, efficiency, and cost of the software are all improved by forecasting software defects at an earlier stage. Many existing models predict defects to facilitate software testing process for testers. A comprehensive review of these models from different perspectives is crucial to help new researchers enter this field and learn about its latest developments. Algorithms, method types, datasets, and tools were the only perspectives discussed in the current literature. A comprehensive study that takes into account a wide spectrum of viewpoints hasn't yet been published. Examining the development and advancement of SDP-related studies is the goal of this literature review. It provides a comprehensive and updated state-of-the-art that satisfies all stated criteria. Out of 591 papers retrieved from 6 reputable databases, 73 papers were eligible for analysis. This review addresses relevant research questions regarding techniques & method types, data details, tools, code syntax, semantics, structural and domain information. Motivation to conduct this comprehensive review is to equip the readers with the necessary information and keep them informed about the software defect prediction domain.*

Keywords: *Software Defect Prediction, Software testing, Solve class imbalance issue, Feature selection, Machine learning.*

1. Introduction

With the tremendous technological development, applications usage has become inevitable. Unfortunately, the larger number of applications, more issues have been detected. Predicting these issues at an early stage is important because not all application fields can tolerate the presence of issues in their live applications. For example, the presence of issues in applications related to the medical field may lead to deaths and the presence of issues in applications related to the space field may lead to a national disaster.

Programmers employ a wide range of tools during the software development process, such as task management, bug tracking, and version control systems. For their usage, there are various open-source

***Corresponding Author**: Khaled S. Shebl

Information Systems Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt

Email address: khaled.said@cis.asu.edu.eg

and commercial software solutions. In addition, several online services are developed to satisfy these requirements [1].

One of the main difficulties in programming language research and software development is prediction of defects, a crucial step in raising the quality and dependability of software. Finding flawed source code with great accuracy is a significant issue in this domain. A lot of methods have been offered throughout the years to the challenging problem of creating defect prediction models. [2].

Early reviews of Software Defect Prediction in the literature were constrained to debates on SDP from a dataset perspective, an algorithmic perspective, or a method type perspective. To the authors' knowledge, a thorough study that addresses several perspectives has not yet been published. In this paper, an in-depth review is proposed with comprehensive comparison on cornerstone perspectives, which include: recency, techniques and method types, data details, tools, code syntax, semantics, structural and domain information. This paper includes a recent and extensive review of the studies associated with SDP that combines all criteria mentioned above to serve as guidance for researchers toward this field of study. Systematically examining 73 papers, this review addresses the following research questions:

- Q1: Is SDP a hot research field?
- Q2: What is the most widespread learning field, algorithms, metrics, and method types in SDP?
- Q3: What are the most common data sources, datasets distributions and used languages in SDP?
- Q4: What are the dominant tools used in SDP field?
- Q5: How popular are code syntax, semantics, structural and domain information?

The structure of this review is as follows. Section 2 shows the major differences between our paper and other review papers. Section 3 explains the review methodology and inclusion/exclusion criteria. Section 4 presents details on the included papers. Section 5 elaborates on answers to the motivating research questions. Finally, section 6 concludes the review.

2. Related Work

This section covers other recent review papers in the Software Defect Prediction field.

The authors of [3] compared the effectiveness of data mining, machine learning, and deep learning techniques in order to predict both cross-project and within-project defects. They looked at the many kinds of empirical comparison and validation measures. They emphasized the application of instance filtering and attribute selection during dataset pre-processing to get better outcomes. They investigated the most widely used datasets and comparison standards for SDP. They chose 68 primary research papers, summarizing their characteristics based on datasets, methods, and performance metrics. They evaluated the effectiveness of models for predicting software defects using data mining, machine learning, and deep learning techniques.

The authors of [4] discussed and compared several research studies and systems that use Logistic Regression. They identified and categorized measuring techniques, including metrics, features, parameters, classifiers, accuracy, and data sets. Additionally, to show how effective their approach is, their obstacles, risks, and limitations were listed. Next, this review-based study made distinctions between several existing systems based on six fundamental measurement criteria and statistically analyzed them.

It also illustrated the difficulties associated with current methods, highlighting the need for an autonomous and successful defect-based prediction system built on a foundation of classifiers.

A systematic literature review was conducted by the authors of [5] to assess the use of machine learning for predicting mobile application defects. In order to address nine questions, they identified 47 publications from scientific databases that concentrated on mobile defect prediction models and evaluated them in a variety of ways. Publications that did not present any empirical findings and were not directly relevant to the creation of the mobile defect prediction model were removed. They found problems and gaps in the literature and proposed fixes. According to datasets, platforms, machine learning algorithms, machine learning types, validation approaches, evaluation metrics, software metrics, the best deep learning algorithms and machine learning, gaps, and challenges, the chosen publication was categorized, and the associated findings were presented.

In order to fully understand current SDP-related methodologies using DL, the authors of [6] aimed to synthesize literature on SDP using DL, including measurements, models, techniques, datasets, and achievements. They also compared the performance of deep learning models with that of machine learning models in classifying software defects. They offered a sample of 63 primary studies conducted between 2010 and 2021 that met the standards for reporting rigor (quality score >= 4.5) as determined by the quality evaluation criteria. They provided a full SLR (quantitative and qualitative synthesis) of the state-of-the-art in SDP by combining data from those 63 rigorous trials with DL. They published a meta-analysis that contrasted the performance of DL and ML in SDP, split down by research and dataset, for a sample of 19 primary studies that met the requirements for the MA quality evaluation. They used both fixed and random effects analyses, as well as presented differences between papers, to confirm the validity of the meta-analysis.

A thorough literature review of Just-In-Time Software Defect Prediction (JIT-SDP) was released by the authors of [7]. Their objective was to provide a thorough overview of the most recent developments in JIT-SDP, covering modeling approaches, data preparation and data sources, dependent and independent variables, performance evaluation. They conducted a meta-analysis of earlier studies and offered a systematic review of 67 JIT-SDP investigations. They also described best practices for each stage of the JIT-SDP process.

As shown in Table 1, related review papers only focused on a few perspectives over a long-time span (about 10 years), while our work considered more perspectives on a large number of relevant research works over a short time span (only 5 years from 2018 to 2022) with the aim of focusing on and discussing the most recent trends. In our work, the included perspectives are learning field, algorithms, performance measures, method types, data sources, datasets, programming languages used in datasets, tools, code syntax information, code semantics information, code structural information, and domain information.

3. Methodology

The methodology used is explained in this section. Criteria for inclusion/exclusion, the procedure and the obtained results are presented in the following subsections.

Table 1 Related work overview

#	Review papers	Number of included papers	Time span
1.	[3]	68	2010–2021
2.	[4]	22	2012-2020
3.	[5]	47	2010-2020
4.	[6]	19	2010-2021
5.	[7]	67	2000-2021
6.	Our paper	73	2018-2022

3.1. Procedure

Software defect prediction is the foundation for this comprehensive review. The key term "Software Defect Prediction" was used in a thorough search of six pertinent databases: ACM Digital Library, Springer, ScienceDirect, IEEE Xplore, MDPI, and Wiley. These databases were used due to their reputation. Database filters were utilized to screen the papers.

The retrieved papers were assessed for adherence to the inclusion requirements. First, all titles and abstracts of the publications were reviewed to check if they complied with these requirements. Papers that passed the first round of screening were downloaded and stored according to the name of the source database. Afterwards, we re-checked all research papers through the body and conclusion again to make sure they meet this review's criteria. Keeping the research questions in mind, the related papers was thoroughly researched to extract the necessary information from the qualified papers, namely: 1) publication year; 2) paper purpose: solve class imbalance issue, feature selection, propose a new approach, or edit an existing approach in software defect prediction ; 3) algorithms and methods; 4) field of learning: machine learning, deep learning or data mining; 5) datasets used in training and testing; 6) dataset source, such as: PROMISE or NASA; 7) programming languages used in datasets; 8) method types include: Just-in-Time Software Defect Prediction, Within-Project Defect Prediction, and Cross-Project Defect Prediction.; 9) evaluation measure, such as: Recall, F-measure, Accuracy, Precision, or AUC; 10) tools; 11) approach used to address the unbalanced dataset issue if the paper's goal is to address the problem of class imbalance, such as: over-sampling or under-sampling; 12) was source code used in the paper? Yes or No; 13) was code semantics information used in the paper? Yes or No; 14) Was structural information used in the paper? Yes or No.

3.2. Criteria for inclusion and exclusion

The following inclusion criteria were used to find relevant studies: 1) any publication, except systematic reviews, that discusses how to address class imbalance issue, feature selection, propose a new approach, or edit an existing approach in software defect prediction; 2) published between January 2018 and April 2022, considering the tremendous speed of change in the area of SDP over the previous five years, even when compared to the rate of change since the invention of the Internet; 3) papers from journals, conferences, or workshops. 4) written in English with no regard to location; 5) not duplicated. Outline of the outcome of the inclusion methodology is presented in Table 2.

Table 2 Summary of selected papers from databases

#	Database	Number of retrieved papers	Number of included papers	Number of excluded papers
1.	IEEE Xplore	226	22	204
2.	Springer	81	18	63
3.	ScienceDirect	218	11	207
4.	ACM Digital Library	23	9	14
5.	MDPI	26	7	19
6.	Wiley	17	6	11
	Total	591	73	518

3.3. Results

There are 592 papers in the search results; however, after the first round of assessment, only 319 were downloaded. 73 papers met the inclusion criteria for analysis after the second round of filtration. The process flow diagram for the systematic review is shown in Figure. 1.

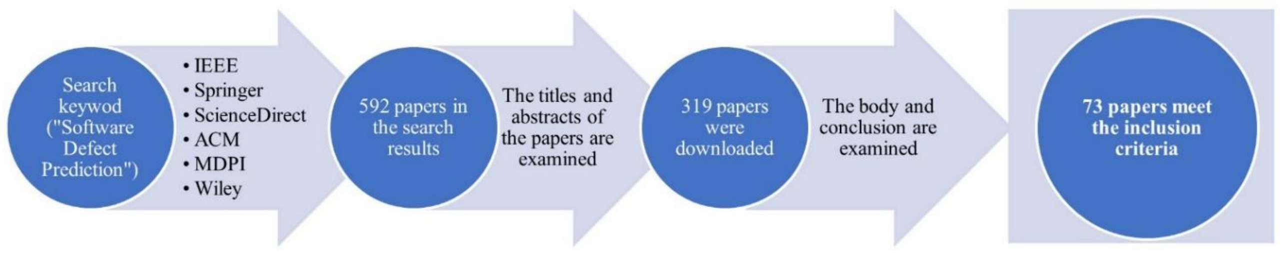

Search keywod ("Software Defect Prediction")
• IEEE
• Springer
• ScienceDirect
• ACM
• MDPI
• Wiley
592 papers in the search results
The titles and abstracts of the papers are examined
319 papers were downloaded
The body and conclusion are examined
73 papers meet the inclusion criteria

Figure.1: Systematic review process flow

4. Literature Review

In this section, details are presented on the 73 papers included in this review. Reviewed research papers are categorized according to the purpose of the paper into 3 types: 1) solve class imbalance issue; 2) feature selection; 3) propose a new approach or edit an existing approach in software defect prediction. It was noticed that the words "prediction" and "detection" were used interchangeably in the review papers. The following subsections discuss the papers under each category. Statistics for the three goals are shown in Table 3.

Table 3 Literature classified according to its purpose

Purpose	Solve class imbalance issue	Feature selection	Edit an existing approach, or propose a new approach
Research papers	[8]–[28]	[29]–[46]	[26], [28], [37], [40], [45]–[80]
Number of papers	21	18	40
Distribution ratio	29%	25%	55%

4.1. Solve Class Imbalance Issue

Datasets for SDP are frequently quite unbalanced, which makes it challenging for classifiers to detect defective occurrences. Numerous strategies have recently been put up to deal with this issue. The methods of Over- sampling and Under- sampling are the most popular methods to resolve the problem of class inequality and increase prediction performance. Over-sampling techniques add new synthetic instances to the minority class or duplicate existing ones, whereas under-sampling techniques eliminate or combine instances in the majority class.

There are 21 research papers that discuss the class imbalance issue, 13 out of 21 use over-sampling and 6 out of 21 use under-sampling technique.

When using a dataset for the software prediction study that focuses on the minority subset, to maximize performance and advantages, the authors of [8] proposed a new model by combining the heterogeneous stacking ensemble with SMOTE technique. To enhance defect prediction, the authors of [9] proposed a new model that incorporates ensemble imbalance learning and class overlap minimization using the AdaBoost mechanism, multiple under-sampling, and the neighbor cleaning technique. In order to produce more fictitious examples of the flawed classes, the authors of [10] developed an innovative hybrid oversampling ensemble technique. The technique combines Fuzzy Based Feature Instance Recovery, Majority Weighted Minority Oversampling, and random oversampling to produce an ensemble classifier. The authors of [11] introduced MAHAKIL, a brand-new and effective synthetic oversampling method depending on datasets of software failures and inheritance chromosomal theory.

To resolve the concerns of class overlap and inequality, the authors of [12] suggested an a more effective K-Means clustering cleaning method (IKMCCA). To reduce processing time and increase forecast accuracy, the authors of [13] suggested model concentrate on balancing the class of datasets. It is made up of two methods: correlation feature selection method and revised under-sampling methodology.

The authors of [14] suggested a hybrid multi objective cuckoo search under-sampled SDP method to handle the class imbalance concern and parameter selection of Support Vector Machine. By taking into account the distribution features of the datasets, a new method for reducing class imbalance approach was developed to balance the imbalanced dataset’s defect and non-defect instances. [15]. The authors of [16] suggested a novel approach to quantify the legitimacy of synthetic samples depending on their distribution by giving each synthetic sample a credit component. Additionally, they suggested a weight update approach to direct base classifiers' attention towards real examples and highly credible fake examples. To overcome the problem of class imbalance, the authors of [17] used SMOTE based on neural networks where SMOTE and neural networks were combined, the neural networks' and SMOTE's hyperparameters are all randomly modified via random search.

The creators of [18] presented a method that uses hybrid sampling, which mixes over-sampling and under-sampling techniques. Over-sampling for minority classes employs k-means clustering of samples and SMOTE generation of synthetic data based on the clustering outcome's safe zones. The possibility that each sample will be misclassified, and its instance hardness value are obtained for the majority class using the logistic regression classifier in the under-sampling method. The samples that have instance hardness values below a certain level are then eliminated from the databases. The authors of [19] proposed the Complexity based Over-sampling Technique (COSTE), which creates synthetic instances by combining pairs of flawed instances with similar complexity. COSTE boosts data diversity, keeps prediction models' ability to identify flaws, and considers the variable testing effort required in various instances. To address the unequal distributions in the SDP, the authors of [20] introduced a mechanism for generating samples for the minority class from a high-dimensional sampling space using random over-sampling. Two restrictions are applied to this mechanism to provide a reliable method for creating new synthetic samples, which involves narrowing the range of random oversampling and differentiating the majority-class samples in key areas.

For efficient defect prediction using SVMs, a unique filtering technique (FILTER) was suggested in [21]. The authors of [22] presented a new over-sampling method that generate synthetic samples using a genetic algorithm. The concept of weighted complexity was suggested by the authors of [23]. Each sample's weighted complexity is determined by taking into account the weights of its many condition variables. Based on the weighted complexity, they proposed a new under-sampling technique called WCP-UnderSampler and used it to forecast software defects. Neighborhood based Under-Sampling (N-US) algorithm is a novel solution that was suggested in [24] to deal with the issue of class imbalance. The authors of [25] conducted this study to look into the connection between the chosen instances' distance and the effectiveness of SMOTE-based approaches. Based on self-organizing data mining, the authors of [26] suggested a new technique for SDP. This technique can show that software metrics and defects are related causally. In [27], they proposed a classification model that blends the ensemble learning technique XGBoost with SMOTE-Tomek sampling. For Cross-Project Defect Prediction, the authors of [28] introduced a new algorithm called (TSboostDF) based on transfer leaning that takes into account class imbalance and the transfer of knowledge.

After analyzing recent research papers dealing with the issue of class imbalance, it is clear that the majority of researchers use the Synthetic Minority Oversampling Technique (SMOTE) to solve this problem.

4.2. Feature Selection

In order to avoid overfitting, increase accuracy, and save training time, feature selection is the procedure of deleting undesired and unnecessary properties from high-dimensional datasets. There are 18 research papers discussing feature selection.

By combining wrapper and filter techniques, the authors of [31] suggested two brand-new hybrid software defect prediction models to pinpoint the important metrics. A feature grouping-based feature selection technique based on the hybrid Wrapper and Filter framework was proposed by the authors of [29]. The authors of [30] proposed a new nonlinear manifold detection model in order to reduce the dimensions of high dimensional datasets while increasing prediction accuracy and software quality. The authors [32] proposed a novel Nonlinear Manifold Detection Model to find the best attributes, eliminating any unnecessary, duplicate, and undesirable attributes in the process.

Local tangent space alignment SVM (LTSA-SVM) technology was used by the creators of [33] to offer a novel model. K-nearest neighbor, support vector machine, and naive bayes classifiers, as well as the firefly algorithm, were utilized in [34] to categorize the features that were chosen. The authors of [35] suggested a framework that builds the prediction model using the Naive Bayes classification method and PCA for dimensionality reduction. The authors of [36] suggested a hybrid preprocessing strategy in which feature selection is followed by iterative partitioning filtering.

The authors of [37] built the improved metaheuristic search based on feature selection method called (EMWS) by combining a distinct complementing simulated annealing approach with the recently developed whale optimization algorithm, which may successfully choose fewer but closely related representative features. In order to overcome the high dimensionality and filter rank selection issues in SDP, a new adaptive rank aggregation-based ensemble multi-filter feature selection (AREMFFS) technique was presented in [38]. By utilizing the island BMFO (IsBMFO) paradigm, an effective binary form of MFO (BMFO) was suggested in [39]. In [40], the research of SDP model based on LASSO-SVM was the main objective, and the issue of the majority of SDP models' subpar prediction accuracy was raised. For managing the dataset's dimensionality issue and enhancing classification performance, the authors of [41] presented a feature extraction technique called FLDA.

The authors of [42] trained a model for just-in-time defect prediction based on random forest classification to increase the prediction effectiveness of just-in-time defect prediction technology. In [43], the authors proposed a feature selection and transfer learning-based method for SDP with metric compensation. An SDP methodology based on heterogeneous feature selection and nested-stacking was suggested by the authors of [44]. A unique SDP method based on PCA and RVFL was proposed by the authors of [45]. The authors of [46] suggested a hybrid classifier that uses a bug tracker application to track and collect defect metrics while gaining semantic knowledge of the traits of the errors by employing an ontology of software flaws.

After analyzing the recent research papers dealing with feature selection, we found that most researchers use Chi-square, Information Gain (IG), Principal Component Analysis (PCA) and LASSO method to choose the important features.

4.3. Edit an Existing Approach or Propose a New Approach

There are 40 research papers which propose a new approach or edit an existing approach to predict software defects.

To improve software defect prediction, a novel technique dubbed node2defect employs the recently announced network embedding method node2vec to automatically learn to encapsulate dependency network structure into low-dimensional vector spaces was proposed by the authors of [47]. In order to distinguish between software entities that are flawed and those that are not, the creators of [48] created a novel supervised classification technique called HyGRAR. This technique incorporates gradual relational association rule mining and ANN. On the basis of atomic class association rule mining (ACAR), the authors of [49] proposed a unique supervised method for SDP. In [50], the authors intended to offer a comprehensive and successful solution for both CSDP and WSDP issues. The semi supervised dictionary learning method was introduced, and a cost-sensitive kernelized semi supervised dictionary learning (CKSDL) strategy was suggested. Deep learning, a potent representation-learning technique, was suggested by the authors of [51] as a way to automatically learn applications semantic representations from code modifications and the files source code. A unique defect prediction model called CAP-CNN (Convolutional Neural Network for Comments Augmented Programs), which is a deep learning model, was proposed in [52]. It automatically embeds code comments in producing semantic features from the source code for SDP.

In [53], authors discussed their practical implementation of a novel deep learning tree-based defect detection model. This model was created using a Long Short Term Memory network with a tree-structure, which perfectly correlates to the Abstract Syntax Tree (AST) representation of source code. A system for determining whether a program module includes bugs was proposed by the creators of [54] and is based on Deep Belief Network Prediction Model (DBNPM). The PROMISE Source Code (PSC) dataset was created by the authors of [55] to expand the CNN research's initial dataset, which they called the Simplified PROMISE Source Code dataset. Then, they suggested an enhanced CNN method for WPDP. The authors of [56] presented a unique weighted naive Bayes strategy for SDP based on information diffusion. The authors of [57] presented a method called Defect Prediction through Attention Mechanism (DP-AM) to fully utilize the semantics and static metrics of applications. The authors of [58] introduced Seml, a unique framework for defect prediction that combines word embedding and deep learning techniques.

Density-percentile-average (DPA), a metric that was suggested by the authors of [59], was utilized as the goal of model optimization on the training set. Then, they developed models using logistic regression and calculated the logistic regression coefficients using the differential evolution algorithm. Without the need of feature-extraction tools, the creators of [60] suggested an end-to-end framework that could directly obtain prediction results for programs. In order to achieve this, they initially extracted visual attributes like image featured from programs using the self-attention method, applied it to visualize programs as images, used transfer learning to lessen the disparity in sample distributions between projects, and then fed the image into a pre-trained, deep learning model for defect prediction. In [61], the authors presented a novel method for predicting the quantity of software flaws using deep learning techniques. They began by doing data normalization and log transformation on a publicly accessible dataset. They next did data modelling. Third, they used a deep neural network-based model that was specifically created to receive the modelled data and forecast the quantity of flaws. The SDP techniques operated at units, like a class or module, which made it more difficult for developers to pinpoint the issue.

The authors of [62] proposed a novel method called Statement-Level Software Defect Prediction Using Deep-Learning Model (SLDeep) to address this problem. Using a long short-term memory (LSTM) network, the authors of [64] were able to automatically extract the semantic and contextual elements from the source code. The Convolution Neural Network (CNN) was employed in [66] to identify the software components that contain defects. A collection of software modules from five specific datasets were employed in this study using static code metrics. SDP model based on program slice and deep learning was proposed by the creators of [67]. They used the Gated Recurrent Unit (GRU) to create features and retrieved the program slice based on the system dependence graph. The authors of [68] proposed a brand-new technique called ALTRA that makes use of both active learning and TrAdaBoost.

A unique method for SDP of unlabeled datasets using modified objective cluster analysis (OCA) was proposed in [63]. In [69], Domain adaptation (DA) is implemented using kernel twin SVMs (KTSVMs) to fit the distributions of training data for various applications. Additionally, KTSVMs with DA function (also known as DA-KTSVMs) are employed as the CPDP model. The authors of [70] introduced a new strategy for drift detection and adaptation based on paired learner that adjusts concepts dynamically by updating one of the pair's learners. A new paradigm for defect prediction based on hierarchical neural networks (DP-HNN) was proposed by the creators of [71]. The enormous file-level AST was divided into many subtrees according to certain AST nodes important to the SDP job, utilizing the AST's hierarchical structure.

In order to forecast software problems, the developers of [72] suggested an approach that incorporated deep learning techniques and word embedding. They would map tokens using an abstract syntax tree derived from the source code. To better reduce the class overlap issue without modifying hyperparameters in SDP, The class overlap cleaning based on the radius technique was presented by the authors of [73].

To enhance defect prediction performance on CPDP, the authors of [74] set out to jointly learn statement level trees (SLT) and reduce data distribution differences using maximum mean discrepancy (MMD). A novel code graph model called Augmented-CPG was put forth in [75]. A defect region candidate extraction strategy specific to the defect type was suggested based on this representation. Key information can be embedded into end-to-end learning software modules' code semantics using the SDP technique that the authors of [76] presented. This method was based on the Transformer model and was entirely dependent on self-attention mechanisms.

The authors of [77] aimed to develop a correlation based neural network model for detecting defects in software applications. Using modelled data, a novel correlation-based modified long short-term memory (CM-LSTM) was introduced to estimate the software defects in software modules. A solution to eliminate the unfavorable confounding effects of size metric was put forth by the creators of [78]. A spotted hyena optimizer classifier used by the creators of [65] to anticipate problems in cross-projects. For the purpose of finding the most effective classification rules, confidence and support were used as a multipurpose fitness function. A fresh methodology was proposed by the authors of [79] based on a comprehensive feature set that includes software metrics for both products and processes that were gathered from software system commits and their history. A form of the deep temporal convolutional network based on the layers of hierarchical attention was also incorporated into the algorithm in order to carry out the defect prediction. Semantic feature based on BERT and bidirectional long short-term memory, which captured the code semantics, were used in the SDP model that authors of [80] provided to forecast software defects.

Notably, six papers [26], [28], [37], [40], [45], [46] combine more than one category, such as (solve class imbalance issue & propose a new approach or edit an existing approach) or (feature selection & propose a new approach or edit an existing approach), already presented in previous categories.

After analyzing the recent research papers that propose a new approach or edit an existing approach, we found that the most used algorithms are Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), Graph Neural Network (GNN), Artificial Neural Network (ANN), Recurrent Neural Networks (RNN), Back Propagation (BP) neural network, Logistic Regression, Random Forrest, Naïve Bayes and A-priori.

5. Discussion

In this section, we elaborate on our insights and findings in software defect prediction research field. The following subsections discuss different perspectives to answer the previously mentioned research questions. Subsection 5.1 answers Q1: Is SDP a hot research field? Subsection 5.2 answers Q2: What is the most widespread learning field, algorithms, metrics, and method types in SDP? Subsection 5.3 answers Q3: What are the most common data sources, datasets distributions and used languages in SDP? Subsection 5.4 answers Q4: What are the dominant tools used in SDP field? Subsection 5.5 answers Q5: How popular are Code Syntax, Semantics, Structural and Domain Information?

5.1. Research Field Recency

The distribution of research papers according to the publication year is shown in Figure. 2. Notably, SDP is a very hot research topic. 18 papers were published in 2021. During only 4 months in 2022. (From January 2022 to April 2022), 16 research papers have been published, showing a clear rise in scholarly attention.

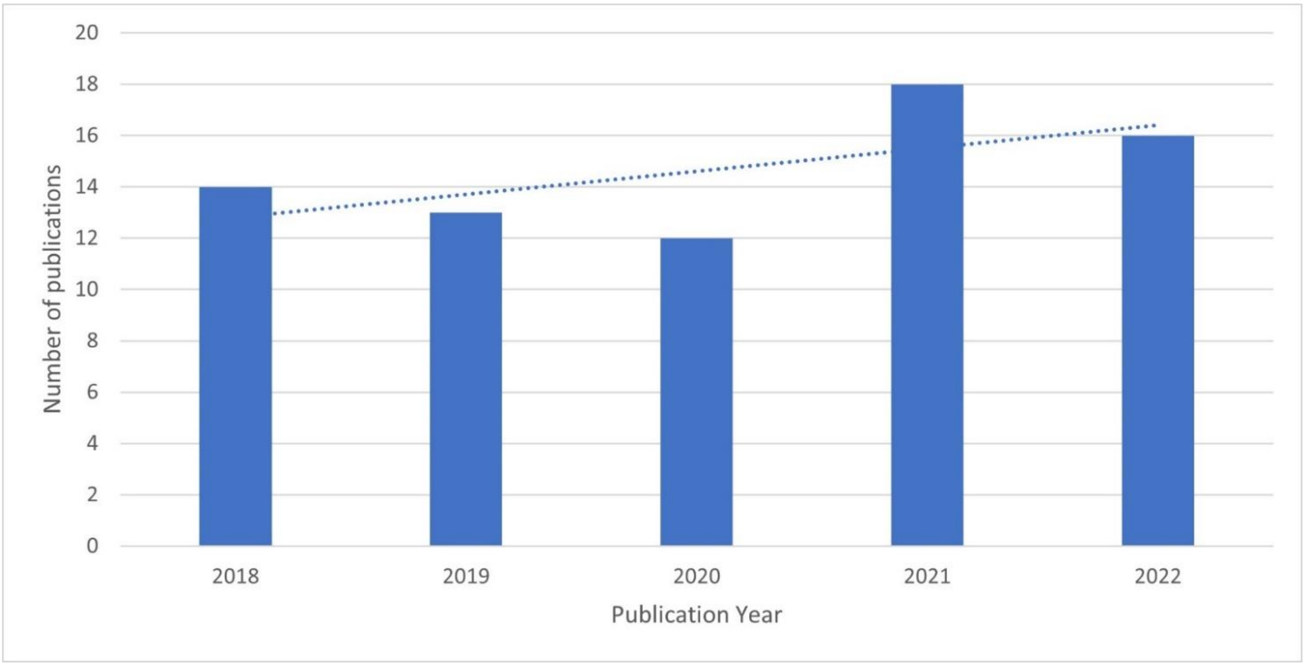


Figure. 2: Distribution of reviewed papers according to publication year

5.2. Techniques, Performance Metrics and Method Types

Techniques, performance metrics and types of methods are covered in this section to help the researchers learn the most common areas of learning, algorithms, performance measures, along with statistics about method types. The details will be discussed in the following two subsections.

5.2.1. Techniques and Performance Metrics

Identifying the algorithms used is crucial for any researcher interested in the field of predicting software flaws. Realizing the most common and least common types would help him in deciding on conducting research on which type. Our review includes research papers in different areas of learning such as machine learning, deep learning, and data mining. After analyzing the included papers, we noticed that 64% of the papers fall into the field of machine learning, 31% of the papers fall into the field of deep learning, and 5% of the papers fall into the field of data mining. Therefore, we encourage researchers to conduct more research in the field of deep learning, especially that work in the field of deep learning has better performance metrics values compared to that in the field of machine learning and data mining. Moreover, we encourage researchers to pay more attention to the field of data mining because the number of research works in this field is very few compared to the other fields.

The use of performance measures is a must to evaluate the work and determine whether the research paper has good or bad results compared to benchmark approaches. The performance measures used vary according to the field of research. After conducting a comprehensive analysis on the included papers, we found that five measures dominate the evaluation of predicting software defects, namely: F-measure, AUC, Recall, Accuracy and Precision. Notably, 67% of papers used F-measure, 49% of papers used AUC, 44% of papers used Recall, 41% of papers used Accuracy, and 36% of papers used Precision.

Details regarding the field of learning, algorithms and performance metrics used in reviewed papers are shown in Table 4.

Table 4 Techniques and performance metrics details

Paper	Field of learning	Algorithms and techniques	Performance metrics
[79]	Deep Learning	Temporal Convolutional Networks (TCN network)	Accuracy, F-measure
[51]		Deep Belief Network (DBN), ASTs, bag-of-words	Precision, Recall, F-measure
[61]		Deep Neural Network-based model	Mean Square Error (MSE), Squared correlation coefficient
[55]		Improved CNN model, ASTs, Word Embedding, Skip-gram model	F-measure, G-measure, and MCC
[71]		Hierarchical Neural Network (HNN), Bidirectional LSTM (Bi-LSTM) algorithm, ASTs, Word Embedding	MCC, AUC
[57]		Defect Prediction through Attention Mechanism (DP-AM), ASTs, Recurrent Neural Network, self-attention mechanism, global attention mechanism, Bi-LSTM, Word Embedding	F-measure
[53]		Tree-structured LSTM network (Tree-LSTM), ASTs	F-measure, Precision, Recall, AUC
[60]		Deep Transfer Learning (DTL) model based on AlexNet network with self-attention mechanism and transfer learning	F-measure
[62]		Long Short-Term Memory (LSTM), ASTs	Accuracy, Recall, Precision, F-Measure
[67]		System Dependence Graph (SDG), leverage Gated Recurrent Unit (GRU), ASTs, Word2vec, CBOW	Precision, Recall, F-measure
[52]		CAP-CNN (Convolutional Neural Network for Comments Augmented Programs), Word2vec model	F-measure
[58]		ASTs, Word Embedding, a Continuous Bag of Words (CBOW) model, LSTM, Word2vec2, CBOW	Precision, Recall, F-measure
[80]		Bidirectional Long Short-Term Memory (BiLSTM), Data augmentation technique, BERT-based semantic feature, Attention mechanism, ASTs	Precision, Recall, F-measure
[74]		Statement semantic learning and maximum mean discrepancy, Word Embedding, Word2Vec, Bi-GRU model, Stochastic Gradient Descent (SGD) algorithm, ASTs.	AUC
[54]		Deep Belief Network, BP algorithm	Accuracy, Precision, Recall, F-measure
[64]		Long Short-Term Memory (LSTM), ASTs, Word Embedding technique	Precision, Recall, F-measure
[77]		Correlation-based modified LSTM	Precision, Accuracy, True Positive Rate (TPR), True Negative Rate (TNR), False Positive Rate (FPR), F-measure, AUC-ROC
[72]	Deep Learning	LSTM, ASTs, Word Embedding, Continuous Bag of Words (CBOW), Word2Vec	Accuracy, Precision, Recall, F-measure
[75]		Augmented Code Property Graphs (Augmented-CPGs), Word2Vec, Graph Neural Network (GNN), ASTs, Word Embedding, Word2Vec	Accuracy, Recall, Precision, F-measure, AUC

[76]		Logistic Regression, CNN, Transformer model and Softmax Neural Network, ASTs, word embedding	F-measure
[66]		CNN	Accuracy
[46]		Ontology, GRU model	Precision, Recall, Accuracy, false negative rate (FNR), F-measure
[37]	Deep Learning + Machine Learning	Whale Optimization Algorithm (WOA), Kernel Extreme Learning Machine (KELM) for new approach to improve performance using Deep Learnin, simulated annealing (SA) for feature selection using Machine Learning, Convolutional Neural Network (CNN)	F-measure, MCC, G-measure, AUC
[59]	Machine Learning	A new method called DEJIT based on differential evolution algorithm, Logistic Regression	Popt, Accuracy
[13]		Modified Under-Sampling method (MUS) and a Correlation Feature Selection (CFS) method, Z-score method, Tomek link (TLink) technique, Greedy Forward Selection (GFS) algorithm.	F-measure, Processing time (s)
[42]		LIME (Local Interpretable Modelagnostic Explanations) Interpretability technique. SP (Sub-modular Pick) LIME (SP-LIME).	Accuracy, Recall, F-measure, AUC
[18]		K-Means, SMOTE, Logistic Regression	Accuracy
[16]		Anovel Credibility-based Imbalance Boosting (CIB) method	AUC, F-measure, Adjusted F-measure (AGF), MCC
[56]		Information Diffusion Model (IDM), Weighted Naïve Bayes (WNB)	F-measure
[22]		Genetic Algorithm (GA)	Accuracy, Recall, False Alarm Rate
[45]		Random Vector Functional Link (RVFL), Principal Component Analysis (PCA), Randomized Neural Networks	AUC–ROC, F-measure
[12]		Improved K-Means Clustering Cleaning (IKMCCA)	balance, Recall, AUC
[50]		Semi-supervised dictionary learning technique, sensitive kernelized Semi-supervised dictionary learning (CKSDL) approach	Recall, Precision, F-measure, AUC
[44]		Nested-Stacking, heterogeneous feature selection., CatBoost, LightGBM, AdaBoost, Random Forest, MLP, Gradient Boosting Decision Tree, Meta-classifier of Logistic Regression.	Precision, Recall, F-measure, AUC
[69]		Improved Quantum Particle Awarm Optimization algorithm (IQPSO), DA function, kernel twin support vector machines (KTSVMs)	F-measure, AUC
[63]		Modified Objective Cluster Analysis (OCA) algorithm, Z-score	Precision, Recall, F-measure, AUC
[14]		Hybrid Multi- Objective cuckoo search Under Sampled SDP model based on SVM (HMOCS-US-SVM), K-Means, Neighborhood Clearance, and NearMiss-2	Probability of Detection, False Positive Rate, G-mean
[20]		Radius Synthetic Minority Over- sampling Technique (RSMOTE)	Recall, Precision, F-measure
[15]		An innovative Class Imbalance Reduction (CIR) algorithm for creating new samples is based on determining the centroid of all attributes of minority-class samples.	Accuracy, Precision, Recall, F-Measure, Specificity, G-mean
[68]		Active Learning, Tradaboost, Weighted variant of Support Vector Machine (WeightSVM), Burak filter	AUC, F-measure
[11]		MAHAKIL, a novel and efficient synthetic oversampling approach based on the chromosomal theory of inheritance	Precision, Recall, F-measure

[43]	Machine Learning	Pearson feature selection method, transfer learning technique, a metric compensation method called peUpMeCom consists of two main phase (1) Pearson coefficients (2) Upgrade metric compensation (upMeCom)	AUC, F-measure
[28]		Transfer-leaning algorithm (TSboostDF), Bernoulli Sampling based on Weight (BSW), cumulative method	F-measure, G-mean, MCC, balance
[23]		WCP-UnderSampler based on weighted complexity	Accuracy, F-measure, AUC
[29]		FCBF-based grouping algorithm, PSO algorithm	AUC
[47]		Node2Vec	F-measure, AUC
[8]		SMOTE technique with the heterogeneous stacking model composed by three base classifiers (Naïve Bayes, Multilayer Perception, J48) and a meta classifier (Boosting or Bagging).	TP Rate, FP Rate, Precision, Recall, F-measure, MCC, ROC Area, PRC Area
[30]		Nonlinear Manifold Detection Techniques (ISOMAP (Isometric Feature Mapping), LLE (Locally Linear Embedding), Diffusion Maps, Laplacian Eigen Maps, NPE (Neighborhood Preserving Embedding), SPE (Stochastic Proximity Embedding), LPP (Linearity Preserving Projection), L-ISOMAP (Landmark ISOMAP))	Accuracy Percentage, F-measure, AUC
[19]		Complexity-based OverSampling TEchnique (COSTE)	AUC, balance, probability of detection (pd), probability of false alarms(pf), Norm (Popt), Accuracy
[33]		Local Tangent Space Alignment SVM (LTSA-SVM) algorithm	Accuracy, Precision, Recall, F-measure
[70]		Random Forrest, Paired Learners of Naïve Bayes.	Accuracy, ROC, AUC
[38]		A new Adaptive Rank aggregation-based Ensemble Multi Filter Feature Selection (AREMFFS). Three filter FS methods: chi-square (CS), Relief (REF), information gain (IG).	Accuracy, AUC, F-measure
[39]		Swarm intelligence algorithm. Island-Based Moth Flame Optimization (IsMFO) Algorithm	Recall, Recall, G-mean
[21]		Support Vector Machines (SVMs), a novel Filtering technique (FILTER)	ROC, AUC, Accuracy, F-measure
[34]		Firefly algorithm	F-measure, Recall, Precision, Accuracy
[24]		A new Neighborhood based Under- Sampling (N-US) technique.	ROC, AUC, Accuracy
[40]		LASSO–SVM (Use LASSO method and Support vector machine (SVM))	Accuracy, Precision, Recall, F-measure
[9]		AdaBoost mechanism, Ensemble Random Under- Sampling (ERUS), Neighbor Cleaning Learning (NCL)	G-mean, Probability of False alarm (PF), Probability of Detection (pd)
[35]		Principal Component Analysis (PCA) and Naïve Bayes classification algorithm	Accuracy
[41]		Linear Discriminant Analysis by Fisher (FLDA) feature extraction technique	Recall, AUC
[73]	Machine Learning	Radius-based class Overlap Cleaning Technique (ROCT)	AUC, balance, Probability of detection (pd), probability of false alarm (pf)
[17]		Synthetic Minority Over-sampling Technique (SMOTE), Neural Network (NN)	Recall, balance
[31]		Wrapper and filter techniques, Support Vector Machines (SVM), Artificial Neural Network (ANN), Maximum Relevance (MR) filter approach	Accuracy, AUC
[36]		The Maximum Likelihood Logistic Regression (MLLR)	AUC, Accuracy, Precision, Recall, F-measure

[32]		Nonlinear Manifold Detection (NMD), ISOMAP (Isometric Feature Mapping), LLE (Locally Linear Embedding), FASTMVU, D Maps (Diffusion Maps), NPE (Neighborhood Preserving Embedding), SPE (Stochastic Proximity Embedding). Correlation based Feature selection subset evaluator (CFs) along with Chi-Squared attribute evaluator (Chi-Squared).	Accuracy, MAE, RMSE, AUC
[10]		Random Over-Sampling (ROS), Majority Weighted Minority (MWM), Fuzzy-Based Feature and Instance Recovery using Information decomposition (FIDos), Random Forest (RF)	Accuracy, True Positive Rate (TPR), False Negative Rate (FNR), AUC, F-measure
[25]		Improved SMOTUNED (that is a SMOTE-based technique)	AUC, balance, probability of detection (pd), Probability of false alarm (PF)
[78]		Causally Removing Negative Confound Effects Method (CRNCEM)	Precision, Recall, F-measure
[27]		SMOTE-Tomek sampling and ensemble learning algorithm (XGBoost)	Accuracy, F-Measure, AUC
[48]	Machine Learning + Data Mining	HyGRAR is a non-linear hybrid model that combines relational association rule mining and artificial neural networks	AUC
[26]	Data Mining	Self-Organizing Data Mining (SODM) algorithm, SMOTE algorithm	Precision, probability of detection (pd), probability of false alarm (PF), F-measure, AUC
[65]		Spotted Hyena Optimizer (SHO) algorithm	F-measure, Sensitivity, Recall, Precision, Specificity, Accuracy
[49]		Atomic Class-Association Rule Mining (ACAR), A-priori	AUC

5.2.2. Method Types

The type of method indicates the relation between the target project that we want to predict its defects and the source projects found in the datasets. Three method types exist:

- Cross-Project Defect Prediction (CPDP): is a procedure that uses data from other projects to forecast defects in a target project.

- Within-Project Defect Prediction (WPDP): is a procedure that uses information from the same project to forecast defects in a target project.

- Just-In-Time Defect Prediction (JITDP): is the process of predicting defects on a commit level.

As shown in Figure 3, method types are divided according to the field of learning. Noticeably, the number of research papers that use cross-project defect prediction are similar to the number of research papers that use within-project defect prediction across all types of learning, whether machine learning, deep learning or data mining. On the other hand, the just-in-time defect prediction method always scores in few numbers in all types of learning. Just-in-time defect prediction has not received its eligible scholarly attention.

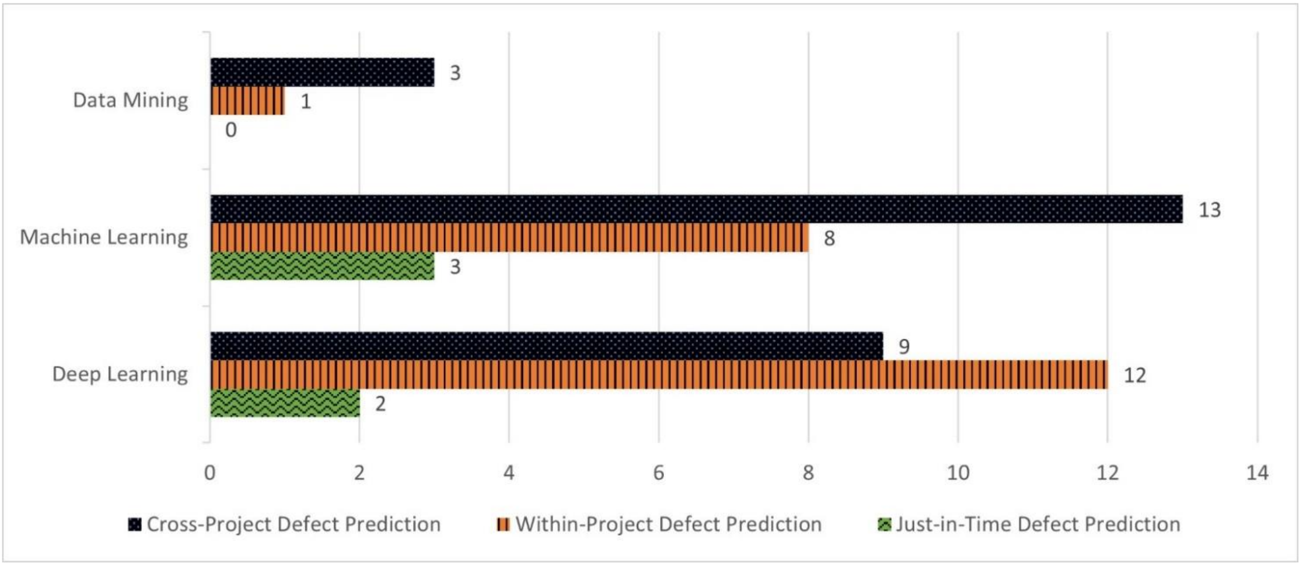


Figure. 3: Distribution of method types according to field of learning

5.3. Data Details

Data is one of the crucial components of any research field that uses machine learning, deep learning, or data mining. The data mainly affects the model that is trained on this data. The greater the size and accuracy of the data, the higher the efficiency of the model. In the following three subsections, we discuss the data criterion from three perspectives, namely: data source, dataset, and programming language.

5.3.1. Dataset Source

There are many data sources used in predicting software defects. Knowing the data source that is used in training and testing the model is important because the more data source is used, the more reliable it is. After careful examination of the included papers, we found that 24 data sources exist. The four most frequently used data sources are PROMISE, NASA, ReLink and AEEEM sorted from most to least popular. Specifically, PROMISE has been used 48 times; NASA has been used 24 times; ReLink has been used 9 times; AEEEM has been used 8 times. Details on these four sources along with the rest of the data sources are shown in Figure. 4.

Since there are four data sources that are the most famous and have proven good results in predicting software defects, we highly encourage researchers to use them for credible results.

5.3.2. Datasets

Researchers are looking for popular datasets so that they can train and test their models to prove their efficiency and to compare their models with other researchers. By studying the datasets used in the included research papers, we found that there is a large number of datasets. Moreover, each researcher uses more than one dataset in his research. We summarized all datasets used in different research papers in Table 5. The table includes four columns, the first column includes the name of the dataset, and the second column includes references of the research papers that used this dataset. The third column includes the number of research papers that used this dataset, and the fourth column shows the proportion of research papers that made use of this dataset.

Table 5 helps new researchers to identify the most used datasets in training, testing, and comparing the existing models, thus reducing the time required to select the datasets they will use in their research.

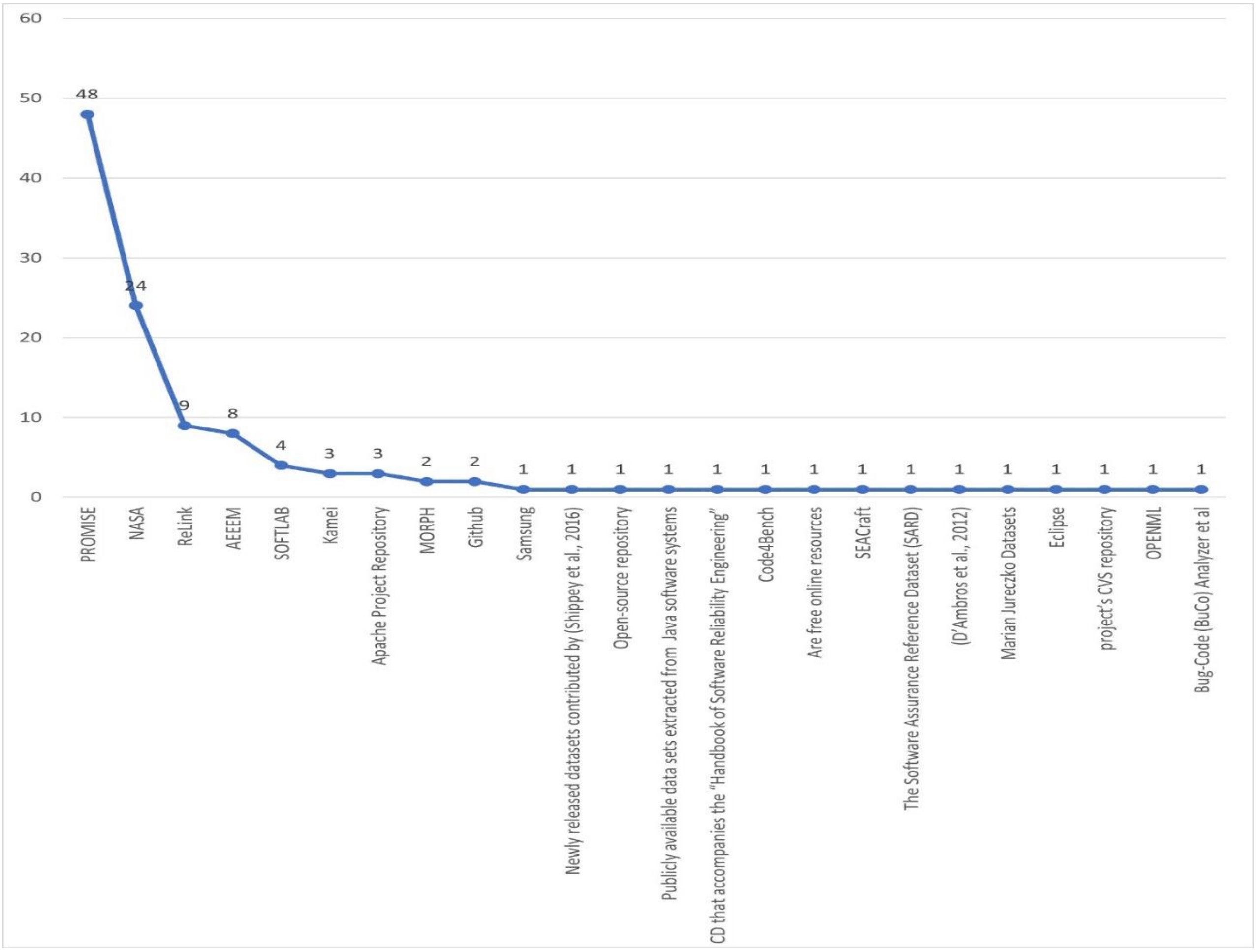


Figure. 4: Distribution of datasets sources

Table 5 Details of datasets

Dataset	Papers	Number of papers	Distribution ratio
Camel	[11]–[13], [15], [16], [18], [19], [22], [23], [25], [26], [37]–[39], [44], [45], [47], [51]–[53], [57], [58], [60], [64], [67]–[69], [71]–[74], [80]	32	44%
Xalan	[12], [15], [16], [18]–[20], [23], [25], [26], [28], [29], [37]–[39], [44], [47], [50]–[53], [56]–[58], [60], [64], [67]–[69], [71], [73], [74], [80]	32	44%
Ant	[11]–[13], [15], [16], [18], [19], [22], [23], [25], [26], [28], [36]–[39], [44], [45], [47], [48], [51], [53], [60], [63], [67], [68], [71], [73], [74], [80]	30	41%
PC	[9], [10], [12]–[14], [16]–[18], [21], [24], [26], [27], [29], [31], [33]–[36], [38]–[41], [43], [49], [50], [54], [56], [66], [73], [78]	30	41%

Lucene (LC)	[12], [15], [20], [25], [26], [32], [37], [39], [43]–[45], [47], [50]–[53], [56]–[58], [60], [64], [67]–[69], [71], [73], [74], [76], [80]	29	40%
Jedit	[11], [15], [16], [19], [22], [23], [25], [26], [28], [37]–[39], [44], [47], [48], [51], [53], [57], [58], [60], [64], [67], [68], [71], [73], [74], [80]	27	38%
KC	[9], [10], [13], [14], [16]–[18], [21], [24], [26], [27], [29], [31], [33]–[35], [38], [39], [41], [43], [49], [54], [61], [65], [66], [70], [78]	27	37%
Xerces	[11], [12], [15], [16], [18], [19], [22], [23], [25], [26], [28], [37], [39], [44], [50]–[53], [57], [58], [60], [64], [67], [71], [74], [80]	26	36%
CM	[9], [10], [12], [14], [16], [18], [21], [24], [26], [27], [31], [33]–[36], [38], [39], [41], [43], [49], [50], [54], [56], [66], [73], [78]	26	36%
Poi	[12], [15], [19], [25], [26], [37], [39], [44], [45], [47], [51]–[53], [56]–[58], [60], [64], [67]–[69], [71], [73], [74], [76], [80]	26	36%
Synapse	[11], [15], [19], [20], [22], [23], [25], [26], [28], [37], [44], [47], [51]–[53], [57], [58], [60], [63], [67], [71], [73], [74], [76], [80]	25	34%
Ivy	[11], [15], [19], [20], [22], [23], [25], [26], [37], [39], [44], [45], [47], [51]–[53], [56], [60], [67], [68], [71], [73], [80]	23	32%
Log4j	[11], [15], [16], [18]–[20], [22], [25], [26], [39], [44], [51]–[53], [58], [60], [63], [64], [67], [68], [71], [73], [74]	23	32%
MW	[9], [10], [12], [14], [16], [20], [26], [27], [33], [35], [38], [39], [49], [50], [54], [56], [66], [73], [78]	19	26%
Velocity	[12], [15], [18]–[20], [23], [25], [26], [29], [38], [44], [47], [56], [71], [73], [74], [80]	17	23%
JM	[10], [16]–[18], [21], [24], [27], [31], [33], [39], [43], [49], [54], [65], [70], [77]	16	22%
Tomact	[11], [12], [15], [16], [22], [28], [36], [38], [39], [47], [48], [50], [68]	13	18%
Eclipse JDT core (JDT)	[12], [13], [36]–[38], [43], [50], [51], [69], [73]	10	14%
MC	[13], [16], [26], [27], [33], [39], [49], [54], [66], [78]	10	14%
PDE	[12], [18], [36]–[38], [43], [50], [69], [73]	9	12%
OpenIntents Safe	[12], [30], [37], [38], [43], [50], [63], [69], [73]	9	12%
ZXing	[12], [30], [37], [38], [43], [50], [63], [69], [73]	9	12%
Apache	[12], [30], [37], [38], [43], [50], [63], [69], [73]	9	12%
Equinox Framework (EQ)	[12], [18], [37], [38], [43], [50], [69], [73]	8	11%
AR	[10], [12], [26], [31], [32], [48], [49], [73]	8	11%
Mylyn (ML)	[12], [18], [37], [38], [43], [50], [69], [73]	8	11%
PostgreSQL (POS)	[13], [42], [44], [51], [59], [69]	6	8%
Redaktor	[11], [12], [15], [20], [22], [38]	6	8%
Prop	[16], [37], [45], [68]–[70]	6	8%
Columba (COL)	[13], [42], [44], [59], [69]	5	7%
Mozilla (MOZ)	[13], [42], [44], [59], [69]	5	7%
Arc	[12], [15], [20], [22], [28]	5	7%
Bugzilla (BUG)	[42], [44], [59], [69]	4	5%
Eclipse Platform (PLA)	[42], [44], [59], [69]	4	5%
Eclipse JDT	[42], [44], [59]	3	4%
Xerces-init	[15], [25], [73]	3	4%
forrest	[26], [63]	2	3%

berek	[32], [63]	2	3%
sklebagd	[26], [63]	2	3%
pbeans2	[11], [22]	2	3%
systemdata	[11], [22]	2	3%
Skarbonka	[12]	1	1%
Open-source projects contributed by Samsung	[53]	1	1%
DrJava	[47]	1	1%
Genoviz	[47]	1	1%
Jmri	[47]	1	1%
Jmol	[47]	1	1%
Jppf	[47]	1	1%
Eclipse34_debug	[30]	1	1%
Eclipse	[38]	1	1%
MIS	[61]	1	1%
119,989 C/C++ programs within Code4Bench	[62]	1	1%
Average	[13]	1	1%
e-learning	[63]	1	1%
termo	[63]	1	1%
syzbkafucha	[63]	1	1%
SWT	[38]	1	1%
Simplified PROMISE Source Code (SPSC)	[55]	1	1%
Apache Luence	[73]	1	1%
Juliet Test Suite for Java in SARD	[75]	1	1%
Linux kernel	[51]	1	1%
Xorg	[51]	1	1%
Jackrabbit	[51]	1	1%
BroadleafCommerce	[58]	1	1%
elasticsearch	[58]	1	1%
hazelcast	[58]	1	1%
netty	[58]	1	1%
orientdb	[58]	1	1%
ZooKeeper	[79]	1	1%
Xerces2 Java	[79]	1	1%
JFreeChart	[79]	1	1%
Jackson Data Format	[79]	1	1%
Jackson Core	[79]	1	1%
Commons Imaging	[79]	1	1%

pdftranslator	[26]	1	1%
wspornaganiepi	[26]	1	1%

5.3.3. Use of Programming Languages

Knowing the programming languages used in datasets is important in the field of predicting software defects. Every person wants to use a dataset that includes specific programming languages, whether he is interested in it or has experience with it. Figure. 5 shows the 3 programming languages that were used the most in the datasets. First, Java, which is used in 35 research papers. Second, C, which is used in 17 research papers. Third, C++, which is used in 9 research papers. These three languages along with the rest of the statistics are in Figure. 5.

By analyzing Figure 5, we notice an important finding. Very popular programming languages exist, but they are not used in datasets or are rarely used, such as Python, C#, R and JavaScript. Therefore, we highlight the need for researchers to build datasets that use these important programming languages that are widespread in the software development market.

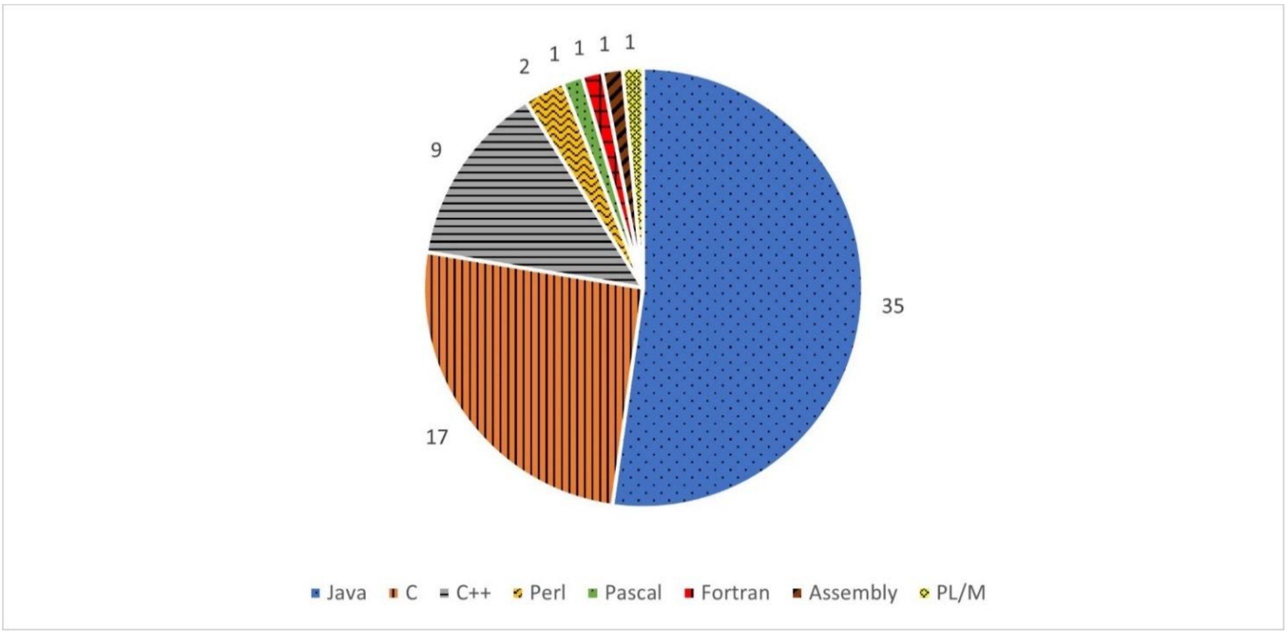


Figure. 5: Distribution of programming languages

5.4. Tools

This section discusses the tools used to build, test, and compare models, preprocess data, or display the results of performance measures. After summarizing all the tools used in the included papers, we found that the four most used tools are as follows Python scripts/libraries, Weka, MATLAB, and Scikit Learn Tool. 15% of the papers use Python scripts/libraries, 15% of papers use Weka, 11% of papers use MATLAB, and 7% of papers use the Scikit Learn tool. Figure 6 shows the number of research papers that use each tool.

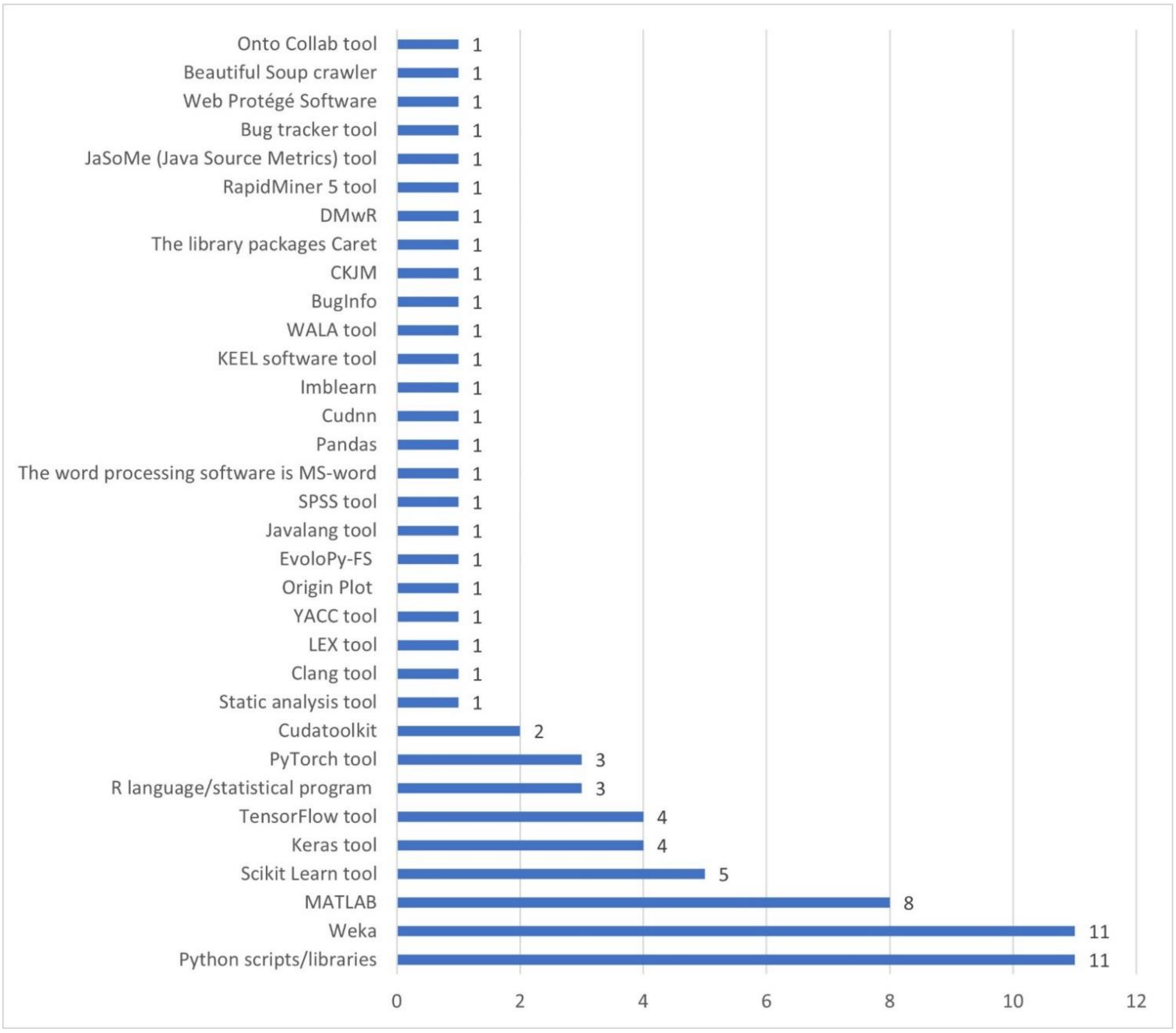


Figure. 6: Distribution of used tools

5.5. Code Syntax, Semantics, Structural and Domain Information

Prediction of software defects remains a difficult problem. There have been numerous attempts to create efficient techniques for forecasting future software flaws. Prior research on defect prediction was based on code metrics. Unfortunately, code metrics are incapable of representing the implicit features of defective modules' syntactic, semantic, structural, and domain information. When compared to earlier methods based on explicit features (code metrics), the application of these implicit features yields superior outcomes.

This section describes the number of papers that use code syntax, semantics, structural and domain information. There are 19 papers that use source code semantics information, 18 papers that use source code structural information, 17 papers that use source code structural syntax information and only 3 papers that use domain information. Results are shown in Figure. 7.

Although the domain information greatly affects the accuracy, it has been used very little. The reason behind that is the training dataset contains data in only one field (such as the commercial, medical, or space field) which increases accuracy. On the other hand, the use of a dataset that includes more than one field negatively affects the accuracy of the model.

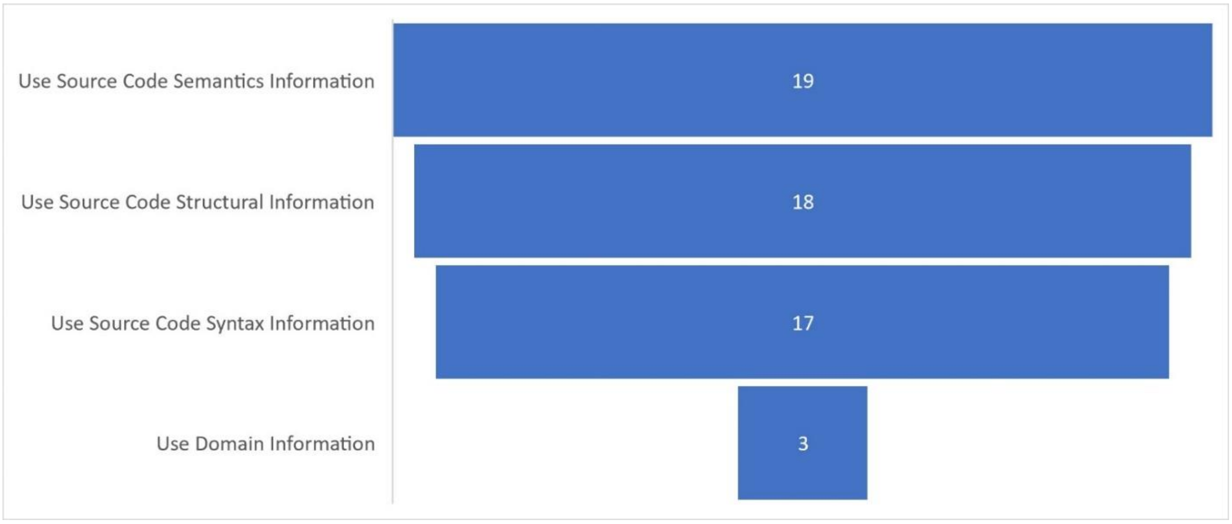


Figure. 7: Summary of code syntax, semantics, structural and domain information

6. Conclusion

This literature review aim is to determine the state of the software defect prediction research by thoroughly examining the developments in published works over the past five years. We provide valuable insights for both practitioners and researchers on the learning field, algorithms, performance measures, method types, data sources, datasets, programming languages used in datasets, tools, code syntax information, code semantics information, code structural information and domain information. We believe this literature review provides the research community with clarity, understanding and motivation for further advancement.

Major findings include that machine learning is extensively used compared to deep learning despite its superior results. Notably, data mining was used very little compared to other fields of learning. We discovered that the most widely employed performance metrics are AUC, F-measure, Recall, Accuracy and Precision. It turns out that the just-in-time defect prediction is the least used type compared to the others. We concluded that the most widely used data sources are PROMISE, NASA, ReLink and AEEEM and summarized all datasets that were used in the included papers. It was clear that Java, C, and C++ were the most often discovered programming languages in the datasets. On the other hand, we noticed that there are very popular programming languages such as Python, C#, R and JavaScript that were rarely used despite their importance and spread. We concluded that the most used tools are Python scripts/libraries, Weka, MATLAB and Scikit Learn tool. We found that domain information was not much used despite its importance.

Suggested future directions are as follows. Researchers are advised to direct their efforts to the fields of deep learning and data mining. Secondly, employ the well-known performance metrics mentioned previously for credible assessment. Thirdly, pay more attention to the just-in-time defect prediction due to the scarcity of papers that use it. Fourthly, create datasets that include neglected -although popular- programming languages such as Python, C#, R and JavaScript. Finally, use domain information because it didn't receive its eligible attention.

References

[1] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth, and T. Gyimóthy, "An automatically created novel bug dataset and its validation in bug prediction," *Journal of Systems and Software,* vol. 169, p. 110691, 2020.

[2] S. Engineering Standards Committee of the IEEE Computer Society, "IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), IEEE Standard Classification for Software Anomalies," 2010.

[3] I. Batool and T. A. Khan, "Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review," *Computers and Electrical Engineering*, vol. 100, May 2022, doi: 10.1016/j.compeleceng.2022.107886.

[4] J. Goyal and R. Ranjan Sinha, "Software Defect-Based Prediction Using Logistic Regression: Review and Challenges," 2022, pp. 233–248. doi: 10.1007/978-981-16-4641-6_20.

[5] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Machine Learning-Based Software Defect Prediction for Mobile Applications: A Systematic Literature Review," *Sensors,* vol. 22, no. 7. MDPI, Apr. 01, 2022. doi: 10.3390/s22072551.

[6] Z. M. Zain, S. Sakri, and N. H. A. Ismail, "Application of Deep Learning in Software Defect Prediction: Systematic Literature Review and Meta-analysis," *Information and Software Technology,* vol. 158. Elsevier B.V., Jun. 01, 2023. doi: 10.1016/j.infsof.2023.107175.

[7] Y. Zhao, K. Damevski, and H. Chen, "A Systematic Survey of Just-In-Time Software Defect Prediction," 2022. [Online]. Available: https://doi.org/xx.xxxx/xxyyzza.aabbcde

[8] S. A. El-Shorbagy, W. M. El-Gammal, and W. M. Abdelmoez, "Using SMOTE and heterogeneous stacking in ensemble learning for software defect prediction," in *ACM International Conference Proceeding Series,* Association for Computing Machinery, May 2018, pp. 44–47. doi: 10.1145/3220267.3220286.

[9] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction," *Software Quality Journal*, vol. 26, no. 1, pp. 97–125, Mar. 2018, doi: 10.1007/s11219-016-9342-6.

[10] S. Huda *et al.*, "An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction," *IEEE Access*, vol. 6, pp. 24184–24195, Mar. 2018, doi: 10.1109/ACCESS.2018.2817572.

[11] K. E. Bennin, J. Keung, P. Phannachitta, A. Monden, and S. Mensah, "MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction," *IEEE Transactions on Software Engineering,* vol. 44, no. 6, pp. 534–550, Jun. 2018, doi: 10.1109/TSE.2017.2731766.

[12] L. Gong, S. Jiang, R. Wang, and L. Jiang, "Empirical Evaluation of the Impact of Class Overlap on Software Defect Prediction," 2019.

[13] P. Lingden, A. Alsadoon, P. W. C. Prasad, O. H. Alsadoon, R. S. Ali, and V. T. Q. Nguyen, "A novel modified undersampling (MUS) technique for software defect prediction," *Comput Intell*, vol. 35, no. 4, pp. 1003–1021, Nov. 2019, doi: 10.1111/coin.12229.

[14] X. Cai *et al.*, "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," *Concurr Comput*, vol. 32, no. 5, Mar. 2020, doi: 10.1002/cpe.5478.

[15] K. K. Bejjanki, J. Gyani, and N. Gugulothu, “Class imbalance reduction (CIR): A novel approach to software defect prediction in the presence of class imbalance,” *Symmetry (Basel)*, vol. 12, no. 3, Mar. 2020, doi: 10.3390/sym12030407.

[16] H. Tong, S. Wang, and G. Li, “Credibility based imbalance boosting method for software defect proneness prediction,” *Applied Sciences (Switzerland)*, vol. 10, no. 22, pp. 1–29, Nov. 2020, doi: 10.3390/app10228059.

[17] R. B. Bahaweres, F. Agustian, I. Hermadi, A. I. Suroso, and Y. Arkeman, “Software defect prediction using neural network based smote,” in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Institute of Advanced Engineering and Science, Oct. 2020, pp. 71–76. doi: 10.23919/EECSI50503.2020.9251874.

[18] X. Du, H. Yue, and H. Dong, “Software Defect Prediction Method based on Hybrid Sampling,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, May 2021. doi: 10.1145/3474198.3478215.

[19] S. Feng *et al.*, “COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction,” *Inf Softw Technol*, vol. 129, Jan. 2021, doi: 10.1016/j.infsof.2020.106432.

[20] S. Guo, J. Dong, H. Li, and J. Wang, “Software defect prediction with imbalanced distribution by radius-synthetic minority over-sampling technique,” *Journal of Software: Evolution and Process*, vol. 33, no. 7, Jul. 2021, doi: 10.1002/smr.2362.

[21] S. Goyal, “Effective software defect prediction using support vector machines (SVMs),” *International Journal of System Assurance Engineering and Management*, vol. 13, no. 2, pp. 681–696, Apr. 2022, doi: 10.1007/s13198-021-01326-1.

[22] C. Arun and C. Lakshmi, “Genetic algorithm-based oversampling approach to prune the class imbalance issue in software defect prediction,” *Soft comput*, 2021, doi: 10.1007/s00500-021-06112-6.

[23] W. Wei, F. Jiang, X. Yu, and J. Du, “An Under-sampling Algorithm Based on Weighted Complexity and Its Application in Software Defect Prediction,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jan. 2022, pp. 38–44. doi: 10.1145/3520084.3520091.

[24] S. Goyal, “Handling Class-Imbalance with KNN (Neighbourhood) Under-Sampling for Software Defect Prediction,” *Artif Intell Rev*, vol. 55, no. 3, pp. 2023–2064, Mar. 2022, doi: 10.1007/s10462-021-10044-w.

[25] S. Feng, J. Keung, P. Zhang, Y. Xiao, and M. Zhang, “The impact of the distance metric and measure on SMOTE-based techniques in software defect prediction,” *Inf Softw Technol*, vol. 142, Feb. 2022, doi: 10.1016/j.infsof.2021.106742.

[26] Q. Zhang and J. Ren, “Software-defect prediction within and across projects based on improved self-organizing data mining,” *Journal of Supercomputing*, vol. 78, no. 5, pp. 6147–6173, Apr. 2022, doi: 10.1007/s11227-021-04113-8.

[27] H. Yang and M. Li, “Software Defect Prediction Based on SMOTE-Tomek and XGBoost,” in *Communications in Computer and Information Science*, Springer Science and Business Media Deutschland GmbH, 2022, pp. 12–31. doi: 10.1007/978-981-19-1253-5_2.

[28] S. Tang, S. Huang, C. Zheng, E. Liu, C. Zong, and Y. Ding, “A Novel Cross-Project Software Defect Prediction Algorithm Based on Transfer Learning,” 2022.

[29] Y. Du, L. Zhang, J. Shi, J. Tang, and Y. Yin, “Feature-grouping-based two steps feature selection algorithm in software defect prediction,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jun. 2018, pp. 173–178. doi: 10.1145/3239576.3239607.

[30] S. Ghosh, A. Rana, and V. Kansal, "A Nonlinear Manifold Detection based Model for Software Defect Prediction," in *Procedia Computer Science*, Elsevier B.V., 2018, pp. 581–594. doi: 10.1016/j.procs.2018.05.012.

[31] S. Huda *et al.*, "A Framework for Software Defect Prediction and Metric Selection," *IEEE Access*, vol. 6, pp. 2844–2858, Dec. 2017, doi: 10.1109/ACCESS.2017.2785445.

[32] S. Ghosh, A. Rana, and V. Kansal, "A Novel Model Based on Nonlinear Manifold Detection for Software Defect Prediction," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, IEEE, 2018, pp. 140–145.

[33] H. Wei, C. Hu, S. Chen, Y. Xue, and Q. Zhang, "Establishing a software defect prediction model via effective dimension reduction," *Inf Sci (N Y)*, vol. 477, pp. 399–409, Mar. 2019, doi: 10.1016/j.ins.2018.10.056.

[34] M. Anbu and G. S. Anandha Mala, "Feature selection using firefly algorithm in software defect prediction," *Cluster Comput*, vol. 22, pp. 10925–10934, Sep. 2019, doi: 10.1007/s10586-017-1235-3.

[35] N. Dhamayanthi and B. Lavanya, "Software defect prediction using principal component analysis and naïve bayes algorithm," in *Lecture Notes on Data Engineering and Communications Technologies*, Springer Science and Business Media Deutschland GmbH, 2019, pp. 241–248. doi: 10.1007/978-981-13-6459-4_24.

[36] K. Bashir, T. Ali, M. Yahaya, and A. Saad Hussein, "A Hybrid Data Preprocessing Technique based on Maximum Likelihood Logistic Regression with Filtering for Enhancing Software Defect Prediction," 2019.

[37] K. Zhu, S. Ying, N. Zhang, and D. Zhu, "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network," *Journal of Systems and Software*, vol. 180, Oct. 2021, doi: 10.1016/j.jss.2021.111026.

[38] A. O. Balogun *et al.*, "An adaptive rank aggregation-based ensemble multi-filter feature selection method in software defect prediction," *Entropy*, vol. 23, no. 10, Oct. 2021, doi: 10.3390/e23101274.

[39] R. A. Khurma, H. Alsawalqah, I. Aljarah, M. A. Elaziz, and R. Damaševičius, "An enhanced evolutionary software defect prediction method using island moth flame optimization," *Mathematics*, vol. 9, no. 15, Aug. 2021, doi: 10.3390/math9151722.

[40] K. Wang, L. Liu, C. Yuan, and Z. Wang, "Software defect prediction model based on LASSO–SVM," *Neural Comput Appl*, 2020, doi: 10.1007/s00521-020-04960-1.

[41] R. B. Bahaweres, E. D. H. Jana, I. Hermadi, A. I. Suroso, and Y. Arkeman, "Handling High-Dimensionality on Software Defect Prediction with FLDA," in *Proceedings of 2nd 2021 International Conference on Smart Cities, Automation and Intelligent Computing Systems, ICON-SONICS 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 76–81. doi: 10.1109/ICON-SONICS53103.2021.9616999.

[42] W. Zheng, T. Shen, X. Chen, and P. Deng, "Interpretability application of the Just-in-Time software defect prediction model," *Journal of Systems and Software*, vol. 188, Jun. 2022, doi: 10.1016/j.jss.2022.111245.

[43] J. Chen, X. Wang, S. Cai, J. Xu, J. Chen, and H. Chen, "A software defect prediction method with metric compensation based on feature selection and transfer learning," *Frontiers of Information Technology and Electronic Engineering*, vol. 23, no. 5, pp. 715–731, May 2022, doi: 10.1631/FITEE.2100468.

[44] L. Chen, C. Wang, and S. Song, "Software defect prediction based on nested-stacking and heterogeneous feature selection," *Complex & Intelligent Systems*, Aug. 2022, doi: 10.1007/s40747-022-00676-y.

[45] R. Malhotra, D. Aggarwal, and P. Garg, "Application of Random Vector Functional Link Network for Software Defect Prediction," 2022, pp. 127–143. doi: 10.1007/978-981-16-3097-2_11.

[46] N. Manoj and G. Deepak, "SDPO: An Approach Towards Software Defect Prediction Using Ontology Driven Intelligence," in *Lecture Notes in Electrical Engineering*, Springer Science and Business Media Deutschland GmbH, 2022, pp. 164–172. doi: 10.1007/978-981-19-1677-9_15.

[47] Y. Qu *et al.*, "Node2defect: Using network embedding to improve software defect prediction," in *ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, Association for Computing Machinery, Inc, Sep. 2018, pp. 844–849. doi: 10.1145/3238147.3240469.

[48] D. L. Miholca, G. Czibula, and I. G. Czibula, "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks," *Inf Sci (N Y)*, vol. 441, pp. 152–170, May 2018, doi: 10.1016/j.ins.2018.02.027.

[49] Y. Shao, B. Liu, S. Wang, and G. Li, "A novel software defect prediction based on atomic class-association rule mining," *Expert Syst Appl*, vol. 114, pp. 237–254, Dec. 2018, doi: 10.1016/j.eswa.2018.07.042.

[50] F. Wu *et al.*, "Cross-Project and Within-Project Semisupervised Software Defect Prediction: A Unified Approach," *IEEE Trans Reliab*, vol. 67, no. 2, pp. 581–597, Jun. 2018, doi: 10.1109/TR.2018.2804922.

[51] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep Semantic Feature Learning for Software Defect Prediction," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1267–1293, Dec. 2020, doi: 10.1109/TSE.2018.2877612.

[52] X. Huo, Y. Yang, M. Li, and D. C. Zhan, "Learning Semantic Features for Software Defect Prediction by Code Comments Embedding," in *Proceedings - IEEE International Conference on Data Mining, ICDM*, Institute of Electrical and Electronics Engineers Inc., Dec. 2018, pp. 1049–1054. doi: 10.1109/ICDM.2018.00133.

[53] H. K. Dam *et al.*, "Lessons learned from using a deep tree-based model for software defect prediction in practice," in *IEEE International Working Conference on Mining Software Repositories*, IEEE Computer Society, May 2019, pp. 46–57. doi: 10.1109/MSR.2019.00017.

[54] H. Wei, C. Shan, C. Hu, Y. Zhang, and X. Yu, "Software defect prediction via deep belief network," *Chinese Journal of Electronics*, vol. 28, no. 5, pp. 925–932, Sep. 2019, doi: 10.1049/cje.2019.06.012.

[55] C. Pan, M. Lu, B. Xu, and H. Gao, "An improved CNN model for within-project software defect prediction," *Applied Sciences (Switzerland)*, vol. 9, no. 10, May 2019, doi: 10.3390/app9102138.

[56] H. Ji, S. Huang, Y. Wu, Z. Hui, and C. Zheng, "A new weighted naive Bayes method based on information diffusion for software defect prediction," *Software Quality Journal*, vol. 27, no. 3, pp. 923–968, Sep. 2019, doi: 10.1007/s11219-018-9436-4.

[57] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Deep Semantic Feature Learning with Embedded Static Metrics for Software Defect Prediction," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, IEEE Computer Society, Dec. 2019, pp. 244–251. doi: 10.1109/APSEC48747.2019.00041.

[58] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A Semantic LSTM Model for Software Defect Prediction," *IEEE Access*, vol. 7, pp. 83812–83824, 2019, doi: 10.1109/ACCESS.2019.2925313.

[59] X. Yang, H. Yu, G. Fan, and K. Yang, "A differential evolution-based approach for effort-Aware just-in-Time software defect prediction," in *RL+SE and PL 2020 - Proceedings of the 1st ACM SIGSOFT International Workshop on Representation Learning for Software Engineering and Program Languages, Co-located with ESEC/FSE 2020*, Association for Computing Machinery, Inc, Nov. 2020, pp. 13–16. doi: 10.1145/3416506.3423577.

[60] J. Chen *et al.*, "Software visualization and deep transfer learning for effective software defect prediction," in *Proceedings - International Conference on Software Engineering*, IEEE Computer Society, Jun. 2020, pp. 578–589. doi: 10.1145/3377811.3380389.

[61] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, Apr. 2020, doi: 10.1016/j.neucom.2019.11.067.

[62] A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, "SLDeep: Statement-level software defect prediction using deep-learning model on static code features," *Expert Syst Appl*, vol. 147, Jun. 2020, doi: 10.1016/j.eswa.2019.113156.

[63] J. Ren and Q. Zhang, "A novel software defect prediction approach using modified objective cluster analysis," *Concurr Comput*, vol. 33, no. 9, May 2021, doi: 10.1002/cpe.6112.

[64] J. Deng, L. Lu, and S. Qiu, "Software defect prediction via LSTM," *IET Software*, vol. 14, no. 4, pp. 443–450, Aug. 2020, doi: 10.1049/iet-sen.2019.0149.

[65] M. A. Elsabagh, M. S. Farhan, and M. G. Gafar, "Cross-projects software defect prediction using spotted hyena optimizer algorithm," *SN Appl Sci*, vol. 2, no. 4, Apr. 2022, doi: 10.1007/s42452-020-2320-4.

[66] K. Wongpheng and P. Visutsak, "Software Defect Prediction using Convolutional Neural Network," 2020.

[67] J. Tian and Y. Tian, "A model based on program slice and deep learning for software defect prediction," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, IEEE, 2020, pp. 1–6.

[68] Z. Yuan, X. Chen, Z. Cui, and Y. Mu, "ALTRA: Cross-Project Software Defect Prediction via Active Learning and Tradaboost," *IEEE Access*, vol. 8, pp. 30037–30049, 2020, doi: 10.1109/ACCESS.2020.2972644.

[69] C. Jin, "Cross-project software defect prediction based on domain adaptation learning and optimization," *Expert Syst Appl*, vol. 171, Jun. 2021, doi: 10.1016/j.eswa.2021.114637.

[70] A. K. Gangwar, S. Kumar, and A. Mishra, "A paired learner-based approach for concept drift detection and adaptation in software defect prediction," *Applied Sciences (Switzerland)*, vol. 11, no. 14, Jul. 2021, doi: 10.3390/app11146663.

[71] H. Yu, X. Sun, Z. Zhou, and G. Fan, "A novel software defect prediction method based on hierarchical neural network," in *Proceedings - 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC 2021*, Institute of Electrical and Electronics Engineers Inc., Jul. 2021, pp. 366–375. doi: 10.1109/COMPSAC51774.2021.00059.

[72] R. B. Bahaweres, D. Jumral, I. Hermadi, A. I. Suroso, and Y. Arkeman, "Hybrid Software Defect Prediction Based on LSTM (Long Short Term Memory) and Word Embedding," in *Proceedings of 2nd 2021 International Conference on Smart Cities, Automation and Intelligent Computing Systems, ICON-SONICS 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 70–75. doi: 10.1109/ICON-SONICS53103.2021.9617182.

[73] S. Feng, J. Keung, J. Liu, Y. Xiao, X. Yu, and M. Zhang, "ROCT: Radius-based class overlap cleaning technique to alleviate the class overlap problem in software defect prediction," in *Proceedings - 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC 2021*, Institute of Electrical and Electronics Engineers Inc., Jul. 2021, pp. 228–237. doi: 10.1109/COMPSAC51774.2021.00041.

[74] W. Liu, Y. Zhu, X. Chen, Q. Gu, X. Wang, and S. Gu, "S2LMMD: Cross-Project Software Defect Prediction via Statement Semantic Learning and Maximum Mean Discrepancy," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, IEEE Computer Society, 2021, pp. 369–379. doi: 10.1109/APSEC53868.2021.00044.

[75] J. Xu, J. Ai, and T. Shi, “Software Defect Prediction for Specific Defect Types based on Augmented Code Graph Representation,” in *Proceedings - 2021 8th International Conference on Dependable Systems and Their Applications, DSA 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 669–678. doi: 10.1109/DSA52907.2021.00097.

[76] W. Zheng, L. Tan, and C. Liu, “Software Defect Prediction Method Based on Transformer Model,” in *2021 IEEE International Conference on Artificial Intelligence and Computer Applications, ICAICA 2021*, Institute of Electrical and Electronics Engineers Inc., Jun. 2021, pp. 670–674. doi: 10.1109/ICAICA52286.2021.9498179.

[77] S. K. Pemmada, H. S. Behera, J. Nayak, and B. Naik, “Correlation-based modified long short-term memory network approach for software defect prediction,” *Evolving Systems*, 2022, doi: 10.1007/s12530-022-09423-7.

[78] C. Li, Y. Yuan, and J. Yang, “Causally Remove Negative Confound Effects of Size Metric for Software Defect Prediction,” *Applied Sciences (Switzerland)*, vol. 12, no. 3, Feb. 2022, doi: 10.3390/app12031387.

[79] P. Ardimento, L. Aversano, M. L. Bernardi, M. Cimitile, and M. Iammarino, “Just-in-time software defect prediction using deep temporal convolutional networks,” *Neural Comput Appl*, vol. 34, no. 5, pp. 3981–4001, Mar. 2022, doi: 10.1007/s00521-021-06659-3.

[80] M. N. Uddin, B. Li, Z. Ali, P. Kefalas, I. Khan, and I. Zada, “Software defect prediction employing BiLSTM and BERT-based semantic feature,” *Soft comput*, Aug. 2022, doi: 10.1007/s00500-022-06830-5.