Received 18 October 2022, accepted 2 November 2022, date of publication 9 November 2022, date of current version 16 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3221184

TOPICAL REVIEW

Cross-Project Defect Prediction: A Literature Review

SOURABH PAL[1] AND ALBERTO SILLITTI[2]
[1]Faculty of Computer Science and Engineering, Innopolis University, 420500 Kazan, Russia
[2]Centre for Applied Software Engineering, 16121 Genova, Italy
Corresponding author: Sourabh Pal (s.pal@innopolis.university)

ABSTRACT **Background:** Software defect prediction models aim at identifying the potential faulty modules of a software project based on historical data collected from previous versions of the same project. Due to the lack of availability of software engineering data from the same project, the researchers proposed cross-project defect prediction (CPDP) models where the data collected from one or more projects are used to predict faults in other project. There are a number of approaches proposed with different levels of success and very limited repeatability. **Goals:** The purpose of this paper is to investigate the existing studies of cross-project models for defect prediction. It synthesizes the literature focusing on characteristics such as project type, software metrics, data preprocessing techniques, features selection approaches, classifiers, and performance measures used. **Method:** This paper follows the well-known Systematic Literature Review (SLR) approach proposed by Barbara Kitchenham in 2007. **Results:** Our finding shows that most of the article was published between 2015 and 2021. Moreover, the studies are mostly based on open-source datasets and the software metrics used to create the models are mainly product metrics. We also found out that most studies attempted to improve their models improving data preprocessing and feature selection approaches. Furthermore, logistic regression followed by naive bayes and random forest are the most adopted classifier techniques in such models. Finally, the f-measure followed by recall and AUC are the most preferred evaluation measure used to evaluate the performance of the models. **Conclusions:** This study provides an overview of the different approaches used to improve the CPDP models analyzing the different techniques used for data preprocessing, feature selection, and the selection of the classifiers. Moreover, we identified some aspects that need further investigation.

INDEX TERMS Software defect prediction, software fault prediction, cross-project defect prediction.

I. INTRODUCTION

Quality plays a major role in the software development life cycle with a number of different points of view that can be considered. For instance, many testing approaches focus on quality interpreted as whether a product meets the requirements; if there are mismatches, these are considered defects. The testing process aims at identifying and measuring software defects: the early identification and fixing of defects can improve the quality of the final product. There are a number of different approaches that can be used including the ones based on the analysis of the activities performed by developers (e.g., [1], [2]), assessment approaches (e.g. [3]), tools (e.g., [4], [5]), etc. Software defect prediction models support the software testing activities helping in the identification of the modules which are the most likely to fail. During the testing phase, those modules can be tested more rigorously to reduce the chance to fail. The early identification of such modules that require particular care in testing are considered as defect-prone modules in prediction models. Testing a large software systems is time-consuming and costly. Due to time and budget constraints, it is wise to identify the potential defect-prone modules in the early phases of development to focus the testing activities on those modules.

The CPDP models have gained significant attention in the last few years. Large software systems include numerous functionalities that interact with each other in complex ways. Therefore, manual testing of those functionalities requires a

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

relevant amount of effort. The idea of the defect prediction is based on the ability to build a model from existing data using supervised or unsupervised machine learning approaches and apply such model on a target project to help in the development and testing phases. The training data of the model could be from the same project (i.e., Within-Project) or different projects (i.e., Cross-Project). The purposes of the defect prediction models are the identification the faulty modules and the estimation of the number of defects, to allocate in an optimal way the available resources to improve the overall quality. With the evolution of artificial intelligence (machine learning in particular), it is possible to identify potential defect-prone modules automatically based on empirical data [6], [7]. Therefore, a fault prediction model can identify the modules that are the most likely to include defects without testing the entire code base to focus the manual testing effort on such modules. This kind of approaches can reduce the overall testing effort and, at the same time, make the effort spent more effective.

In our previous work, we classified the defect prediction models into 10 categories [8]. Most of the papers identified focus on within-project software defect prediction (SDP) models. Such models are developed using training data from the same project. Therefore, the source and target data of the model are homogeneous. However, it is not always possible to find data from the same project, because either they do not exist (for instance, in case of the first release of a product) or they were not collected and stored properly for future use. Due to the lack of availability of empirical data from the same project, the researchers look forward to similar projects to extract data to train the model (cross-project models).

Even if there is significant amount of research in software defect prediction models over last three decades, it fails to make it useful for the industry due to the difficulty of collecting and organizing defect-related software data [11]. CPDP models provide an alternative option which does not require much effort of collecting, storing, and organizing the data. Moreover, since most of software projects are now developed with agile approaches, it is not easy to use information coming from previous development approaches and existing models may not perform well [12]. In such circumstances, the CPDP is a convenient model to bring the defect prediction into practice because it does not rely on the previous data of the same project. It is possible to use similar and suitable open source data for train the cross-project models. Therefore, there is an increasing interest in CPDP research over last decade [11]. Although, there are few literature reviews available, none of then performed a detailed analysis of the approaches used to build the different models. Therefore, in this study, we performed a details analysis of CPDP models since 2009 [20]. The purpose of this Systematic Literature Review (SLR) is to provide a fine-grained analysis of applied software metrics, projects, modeling techniques, evaluation measures, and tools used in CPDP.

II. RESEARCH METHODOLOGY

A. GOALS OF THE RESEARCH

Due to the increasing level of software adoption in many companies, the increasing number of start-ups, and the overall digital transformation, there is an increasing number of new software projects that requires accurate SDP models to achieve high quality levels and increase the customer satisfaction in a very competitive environment.

Due to the lack of suitable data in such projects/companies, the only feasible approach is the implementation of models based on the different flavors of CPDP. The aim of this paper is to provide a reference to practitioners and researchers to understand better the complex world of CPDP and identify the best approach to use in the different contexts.

B. RESEARCH QUESTIONS (RQ)

In practice, we have identified the following research questions to investigate:

- **RQ1: How has the interest in CPDP models evolved over time?**
  This question aims at analyzing the interest of researchers towards the CPDP models based on the number of publications appeared every year from 2009 to 2021.
- **RQ2: Which projects are used in CPDP models?**
  This question investigates which kind of projects have been used in the experimentation of CPDP models: public (open source) projects or private ones.
- **RQ3: Which software metrics are used in CPDP models?**
  Different prediction models require different software metrics. However, some metrics are more popular than others for several reasons. This question investigates such popularity over the investigated timeframe.
- **RQ4: Which are the different techniques applied to improve the performance of CPDP models?**
  Software fault prediction models mainly include three steps: project selection, software metrics identification, and data analysis. This last step also includes three sub-steps: data preprocessing, feature selection, and classification. Project and metrics selection can impact deeply the performance of the prediction model. Moreover, the quality of data can improve the performance of classifiers to distinguish between faulty and non-faulty modules. Raw data quality can be improved through the process of preprocessing and feature selection. Therefore, this question explores how CPDP models changes over time in terms of project selection, software metrics identification, data preprocessing, feature selection, and classification.
- **RQ5: Which are the evaluation measures applied to CPDP models?**
  The evaluation measures provide a quantitative analysis of the the performance of a model. In this question,

we explore the different approaches used so far in CPDP models and which are the preferred ones.

- **RQ6: Which are the different tools used in CPDP models?**
  We discuss the tools used by researchers for software metrics extraction, statistical machine learning, data analysis, parsing source code, etc.

C. METHODOLOGY

We extracted all relevant research papers from the most popular digital libraries: IEEE Xplore, Springer Link, Elsevier Science Direct, ACM Digital Library, and Google Scholar. The number of articles found in each search category is shown in Table 1. We realized that many articles appeared with different queries, therefore we filtered out all the duplicates.

TABLE 1. Number of paper extracted in each search category.

Search Criteria	Number of Papers
Software Defect Prediction	1259
Software Quality Prediction	298
Software Fault Prediction	395
Software Defect Prediction, and Software Quality Prediction	209
Software Defect Prediction, and Software Fault Prediction	87
Software Quality Prediction, and Software Fault Prediction	123
Software Defect Prediction, Software Quality Prediction, and Software Fault Prediction	27
Total	2,398

We merged all the articles fetched removing duplicates and we identified 2,398 papers. Then, we followed the following steps to identify the relevant research papers related to CPDP models:

1) We shortlisted all the papers which contain the term *Cross-Project* in the title or abstract. We found 123 such papers.
2) We went through the abstract of all shortlisted papers to identify whether those papers are really about CPDP. We identified 88 papers.
3) Finally, we read all the papers and shortlisted 81. We removed 7 papers because two papers were related to data security in SDP and five papers were literature reviews we discuss in Section 3 of our study.

Table 2 summarizes our inclusion and exclusion criteria.

TABLE 2. Number of paper selected at each step.

Category	Number of Papers	Include or Exclude: Why?
CPDP	81	Include: These articles deal with CPDP models or experimentation on CPDP models.
CPDP focus on security	2	Exclude: These articles deal with data security issues in CPDP. We exclude these articles because our literature review is not related to security issues in the models.
CPDP literature reviews	5	Exclude: These articles deal with the literature review of CPDP. Therefore, we exclude them. However, Our literature review discusses existing literature reviews in the background section.
CPDP with other purposes	35	Exclude: These articles do not deal with CPDP models and their experimentation. These articles used CPDP as an example or for comparison purposes.
Others	2,275	Exclude: These articles do not belong to the CPDP category. It included within-project models, personalized models, and so on.
Total	2,398	

III. BACKGROUND

According to our findings, most of the available defect prediction papers and review articles focus on within-project defect prediction models [11]. We only extracted the literature review articles of CPDP. According to our findings, there are only five literature reviews on CPDP (all of them from 2017 to 2021). We analyzed the existing literature reviews to compare them with our objectives. This section discusses such aspects.

Lipika Goel et al. [21] performed a Systematic Literature Review on CPDP to summarize various methodologies of CPDP models, data sets used for experiments, software metrics, classifiers, and evaluation measures of CPDP. This review consists of 12 research papers from 2009 to 2016. The data distribution of source and target projects are different in CPDP because, source and target projects data sets collected from the different projects. The Transfer Learning approach widely used in CPDP to reduce data difference between source and target projects. It also defines that source and target project with similar software metrics is known as homogeneous transfer learning whereas source and target project with different software metrics is known as heterogeneous transfer learning. It found that most of the CPDP models evaluated the performance of their models in terms of false-positive rate (probability of false alarm), accuracy, precision, recall, f-measure, and area under the curve (AUC). It showed that product metrics such as Size Metrics, Halstead Metrics, McCabe Metrics, CK Metrics, and Object-Oriented Metrics, are widely used in CPDP. This literature also explored the most used classifiers in CPDP. It identified logistic regression, Naive Bayes, decision tree, random forest, support vector machine, and artificial neural network, are most used classifiers. It performed comparative studies on data set type, transfer learning applied, preprocessing technique, classifiers used, and f-measure. It shows that heterogeneous data sets perform better compare to homogeneous data set in terms of f-measure. Finally, it identified the existing research gaps in CPDP, namely standard training data selection, class imbalance problem, and heterogeneous cross-project fault prediction.

Herbold et al. [22] investigated the performance of CPDP. This study consists of 24 research papers from 2008 to 2015. It found out that Decision Tree classifier-based CPDP performs better compared to other classifiers. It identified that no CPDP approach achieves the desired performance with at least 0.75 recall, 0.75 precision, and 0.75 accuracy. It also investigated the impact on the performance of data filtering approaches. It compared JURECZKO and FILTERJURECZKO datasets to understand whether the performances of the model are affected. JURECZKO dataset [23] includes 48 open source products, 27 proprietary products, and 17 academic products with a total of 92 products (we discuss more such datasets in the RQ2). It found out that there is no significant difference between the two data sets in terms of AUC, f-measure, g-measure, MCC, and AUCEC. However, if the filtered datasets are small compared to the original ones, it can increase the performance up to 5%. Finally, it concluded that the CPDP models has not reached the required performance to apply them in practice.

Hosseini et al. [24] investigated software metrics, prediction models, data approaches, datasets, and the level of performance in CPDP. It also compares the performance of CPDP with WPDP to explore whether CPDP performance are comparable with WPDP. This review consists of 30 research papers on CPDP until 2015. It identified that the performance of a prediction model depends on the choice of software metrics. Therefore, it explores the different software metrics and their performance in CPDP. It found out that process, product, and object-oriented metrics are the most popular software metrics used. Moreover, the combination of object-oriented metrics and other code metrics shows good performance in precision, recall, f-measure, and AUC. It identified Naive Bayes and Logistic Regression as the preferred classifiers in CPDP where Nearest-neighbor, Support Vector Machine, and Decision Tree achieved the highest performance in terms of f-measure. This study also identified that recall, false positives, precision, f-measure, and AUC are the most frequently used measures to evaluate performance. Most of the older studies tried to improve the performance of CPDP reducing the data heterogeneity between source and target projects, while the most recent ones focus on the class imbalance problem, noise data, and feature selection. It also mentioned the preprocessing techniques used to address the data heterogeneity: normalization and log-filtering. It compares the performance between WPDP and CPDP showing that WPDP outperforms CPDP and CPDP challenges WPDP only in precision. Finally, it noticed that CPDP has not reached the required level of performance to be used in practice.

Zhou et al. [25] investigated how CPDP models evolved comparing their performance with WPDP models focusing on the performance of supervised models for both classifying and ranking defect-prone modules. It identifies two categories of prediction models based on module size. They found out that small module size models perform better compared to other CPDP models. Finally, it suggested that future CPDP studies should use the identified models as a baseline for comparison.

Khatri et al. [26] analyzed 34 article related to CPDP from 2008 to 2019. They explored the types of datasets, modeling techniques, types of software metrics, evaluation parameters, and statistical tests performed. They identified that studies used proprietary datasets (6%), open-source datasets (26%), and mixed datasets (a combination of both proprietary and open-source datasets) (68%). They showed that logistic regression and Naive Bayes are the most used classification techniques. Product metrics were used in 62% of the studies whereas process metrics in 38%. Moreover, the f-measure is the preferred evaluation parameter. Additionally, it was observed that 21% of the studies do not include any statistical test, while the most used is the Wilcoxon Signed-Rank test. Finally, they concluded that there is big margin to improve the performance of the CPDP models.

Our review is different from the ones already available in several aspects. We consider the timeframe 2009-2021 while the others do not consider recent studies appeared in 2020 and 2021 (they stop in 2015, 2016, 2017, and 2019). The previous research articles did not perform fine-grained analysis of project selection, data preprocessing, feature selection, and classification. Moreover, we consider several aspects considered separately in other papers:

- we discuss how the CPDP models evolved over time in a similar way as Seyedrebvar Hosseini et al. [24]
- we discuss the different kinds of projects used in a similar way as Steffen Herbold et al. [22]
- we discuss software metrics and evaluation measures in a similar way as Lipika Goel et al. [21] and Seyedrebvar Hosseini et al. [24] but our paper includes the recent literature with a more detailed analysis. Khatri et al. [26] consider only classifiers as models whereas our study will consider preprocessing, feature selection, and classifiers together with the model. In this paper, we analyze CPDP models considering different dimensions: project and software metrics selection, data preprocessing and feature selection, and classifiers.

IV. DISCUSSION

This section focuses on the answers to the research questions presented in Section 2 investigating the current status of CPDP models.

A. RQ1: HOW HAS THE INTEREST IN CPDP MODELS EVOLVED OVER TIME?

Our finding shows that the majority of CPDP studies has been published since 2015. Figure 1 shows the number of papers published each year from 2009 to 2021, also showing that the researcher's interest in the topic has increased over time. We can split the time span from 2009 to 2021 into two parts, namely 2009 to 2014 and 2015 to 2021. 84% of CPDP studies were published during the span of 2015 to 2021 whereas 16% of CPDP studies published during the span of 2009 to 2014.

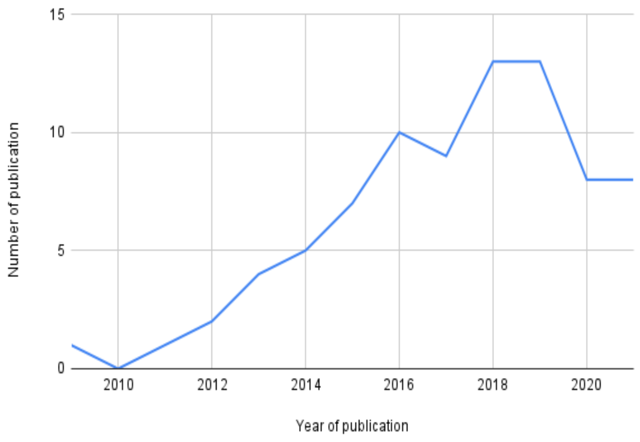


FIGURE 1. Number of papers published each year in CPDP.

B. RQ2: WHICH PROJECTS ARE USED IN CPDP MODELS?

In this section, we focus on the different software projects used in the various articles for the cross-project defect prediction model. Software projects follow the iterative approach which makes them agile in nature. Moreover, the empirical data from software projects used in the different articles are from open source data sets. These data sets are not up to date. For example, the software project data in the PROMISE repository are from 2004, 2005, and 2006. The researchers continuously use these old data sets for their experimentation. It diminishes the agile impact of software projects across the cross-project models in different research articles. Therefore, this article does not take into consideration the agility of the projects. According to our findings, researchers used the terms *dataset* and *repository* interchangeably in their papers, we do the same. Moreover, datasets and repositories consist of multiple project.

The recent trend of CPDP experimentation relies on open source, publicly available datasets. We analyzed 81 CPDP studies and we found that 78 CPDP studies are based on public datasets, 1 CPDP study is based on both public and proprietary datasets [20], and 2 studies do not mention the datasets used. 54 CPDP studies are based on the PROMISE repository. PROMISE (as described below) is a very popular software engineering database designed for research purposes.

Figure 2 shows the most used projects in all the CPDP studies. We choose the project's datasets which were used by more than 5 times. Lucene, followed by Ant and Xalan, are the most used projects. Moreover, we found eight open-source datasets (grouping together a variable number of projects), namely NASA MDP dataset, AEEEM dataset, NetGene dataset, SeaCraft repository, MORPH dataset, ReLink dataset, SOFTLAB dataset, and JURECZKO dataset, are used in CPDP studies:

1) NASA METRICS DATA PROGRAM (MDP) DATASET

[27] NASA MDP Dataset consists of method-level software metrics related to 13 NASA software projects written in Java, C, and C++. The software metrics available in all the projects are not the same. There are two projects (KC1 and KC2) that contain 21 software metrics, one project (PC2) contain 36 software metrics, five projects (MW1, PC1, PC3, PC4, and CM1) contain 37 software metrics, two projects (MC1 and PC5) contain 38 software metrics, and two projects (KC3 and MC2) contain 39 software metrics. We exclude the defective metrics to count the number of software metrics in all projects. These data are available in two sources, namely PROMISE and MDP. In this section, all information provided about the MDP dataset is taken from MDP.

2) AEEEM DATASET

Marco D'Ambros et al. [28] collected and shared the AEEEM dataset. It consists of five open source projects, namely Eclipse JDT Core (JDE), Eclipse PDE UI (PDE), Equinox Framework (EQ), Apache Lucene (LC), and Mylyn (ML). These projects are written in Java and the data provided are 110 software metrics including change metrics, CK metrics, object-oriented metrics, number of previous defect metrics, the complexity of code metrics, churn of CK metrics, churn of object-oriented metrics, the entropy of CK metrics, and entropy of object-oriented metrics. We exclude the defective metrics to count the number of software metrics in all projects.

3) NetGene DATASET

Kim Herzig et al. [29] collected and shared the NetGene dataset. It consists of four open source Java projects, namely Httpclient, Jackrabbit, Lucene, and Rhino. It includes different metrics, namely complexity metrics, dependency network metrics, and genealogy metrics. The genealogy metrics include EGO network metrics to capture direct neighbor dependency, GLOBAL Network Metrics.

4) SeaCraft REPOSITORY

[30] Software Engineering Artifacts Can Really Assist Future Tasks (SeaCraft) Repository combined software engineering data from the PROMISE repository and data from several other sources such as conferences/workshops. The data here has been used in at least one (or more) other datasets. This repository includes data from open source projects, namely Arc, Camel, Lucene, Log4j, Poi, Prop, Synapse, Velocity, and many more.

5) MORPH DATASET

[31] MORPH dataset consists of 14 open source projects and 11 student projects, all written in Java. The projects include Ant, Arc, Camel, Poi, Tomcat, Redaktor, Skarbonka, etc. The data consist of metrics based on CK metrics. The MORPH dataset is available in the PROMISE repository and MORPH.

6) ReLink DATASET

[32] This dataset consists of three projects, namely Apache, Safe, and Zxing. The data contain 26 software metrics similar for all three projects.

7) SOFTLAB DATASET

[33] This dataset consists of five projects donated by SOFTLAB. The data consist of 29 metrics with manually designed defect levels. These data are available in the PROMISE repository.

8) JURECZKO DATASET

[23] This dataset consists of 15 open source projects, namely Ant, Camel, Ckjm, Forrest, Ivy, JEdit, Log4j, Lucene, PBeans, POI, Synapse, Tomcat, Velocity, Xalan, and Xerces. These data are available in both in JURECZKO and PROMISE repositories. It also consists of 6 proprietary projects and 17 academic projects. Each project includes 20 metrics.

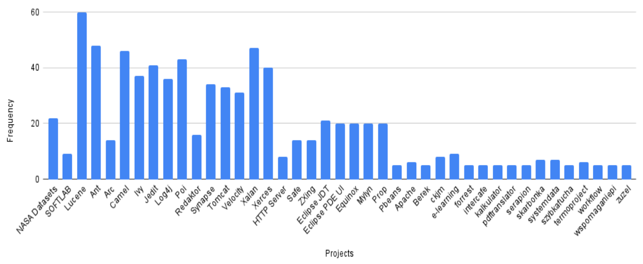


FIGURE 2. Projects used to train CPDP models.

The choice of a dataset is the first step to perform CPDP research. The performance of the model highly depends on the choice of the dataset. Hence, to overcome this, many studies select data from different sources to evaluate the stability of performance across the different datasets.

C. RQ3: WHICH SOFTWARE METRICS ARE USED IN CPDP MODELS?

Features in software fault prediction models can be categorized into two types: semantic features and hand-crafted features. The hand-crafted features are designed and extracted from the source code text (e.g., McCabe [34], CK [36]). Some researchers argue that hand-crafted features lack the semantics information of source code [37]. Therefore, some researchers turn towards to extraction of semantic features that are based on an abstract syntax tree (AST). The semantic feature is extracted based on transforming the source code into a token vector using ASTs [37]. In our findings, the application of semantic feature extraction techniques in the CPDP model is limited (about 6%) compared to the hand-crafted ones (about 94%). After the introduction of deep learning in software fault prediction, researchers tried to leverage semantic hidden features from source code based on the analysis of the abstract syntax trees.

The extraction of data from source code in the form of software metrics is the first stage of fault prediction. CPDP studies used open source datasets mentioned in RQ2, consists of software metrics (e.g., AEEEM datasets consists of 110 software metrics.). Researchers used the following three techniques to choose the most suitable software metrics among all available software metrics in a dataset:

1) **No selection:** researchers used all the available metrics in a dataset for analysis [38].
2) **Manual selection:** researchers choose the most suitable metrics manually based on their intuition and experience [39].
3) **Automated selection:** researchers applied an automatic feature selection approach to select the most suitable metrics among all available software metrics in a dataset [40].

We found that all CPDP studies used a combination of product and process metrics for their analysis. Table 3 shows the preferred product and process metrics used in CPDP studies. We found that the most used product metrics are:

- Chidamber & Kemerer metrics [36]
- Henderson Sellers Metrics [41]
- Martin Metrics [42]
- QMOOD Metrics [43]
- McCabe Metrics [34]
- Size Metrics [44]
- Halstead Metrics [45]
- Object-Oriented (OO) Metrics [46]

The process metrics are also considered as change metrics [47]. Therefore, we also categorized [48] the most used process metrics as follows:

- Number of Revisions [66]
- Commits [69]
- Modified Code [73]

It is also clear from Table 3 that product metrics are preferred over process metrics.

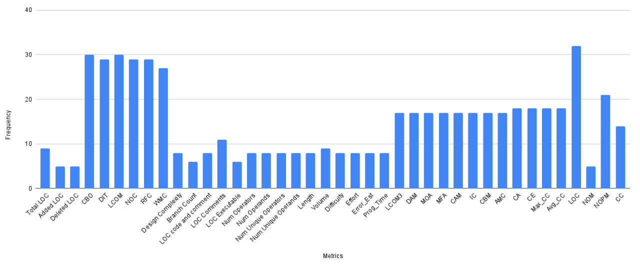


FIGURE 3. Metrics used in CPDP models.

Figure 3 presents the most used metrics in the CPDP studies, we included only the metrics used at least 5 times. LOC, followed by CBO, LCOM, DIT, NOC, RFC are the most used metrics.

TABLE 3. Types of Software Metrics in CPDP.

Category	Metric Type	Metrics
Product Metrics [35]	Chidamber & Kemerer metrics [36]	WMC, DIT, NOC, CBO, RFC, LCOM
	Henderson Sellers Metrics [41]	LCOM3
	Martin Metrics [42]	CA, CE
	QMOOD Metrics [43]	DAM, MOA, MFA, CAM
	Derived from Chidamber & Kemerer Metrics [49]	IC, CBM, AMC
	McCabe Metrics [50]	CC, Avg_CC, Max_CC etc.
	Size Metrics [51]	LOC, LOC Comments, LOC executables etc.
	Halstead Metrics [52]	Num Operators, Num Operands, Num Unique Operators, Num Unique Operands, Length, Volume, Difficulty, Effort, Error_Est, Prog_Time, Vocabulary
	OO Metris [46]	FanIn, FanOut, LOC, NOA, NOPA, NOPRA, NOAI, NOM, NOPM, NOPRM
Process Metrics or Change Metrics [47]	Number of Revisions [66]	Nrev, Nfix
	Commits [69]	Commits, Distinct Commiters
	Modified [73]	Added LOC, Deleted LOC, Changed LOC, Num of modified subsystems, Num of modified directories, Num of modified files

D. RQ4: WHICH ARE THE DIFFERENT TECHNIQUES APPLIED TO IMPROVE THE PERFORMANCE OF CPDP MODELS?

Software defect prediction is a binary classification problem [53]: the goal is to classify a piece of software as a defective or a non-defective one. The classification of software is done based on three different machine learning approaches:

1) **Supervised:** All training data used to build the prediction model are labeled as defective or a non-defective.
2) **Semi-supervised:** The model is built using a small number of labeled training data and a large number of unlabelled ones.
3) **Unsupervised:** They do not require any labeled training data to build the prediction model.

Some supervised learning approaches used are: TCA+ (Transfer Component Analysis) [54], Context-Aware Rank Transformations [55], Weighting and VAB [56], Hybrid Instance Selection Using Nearest-Neighbor (HISNN) [57], FeSCH (Feature Selection using Clusters of Hybrid-data) approach [58], and many more.

The used semi-supervised learning approaches are: Semi-Supervised Structured Dictionary Learning (SSDL) [59], Improved Subclass Discriminant Analysis (ISDA) with Semi-Supervised Transfer Component Analysis (SSTCA) [60], and many more.

The unsupervised learning approaches used are: CLA with CLAMI [31], Cluster Ensembles and Labeling (CEL) [61], and many more.

We found out that most of the studies use supervised learning approaches.

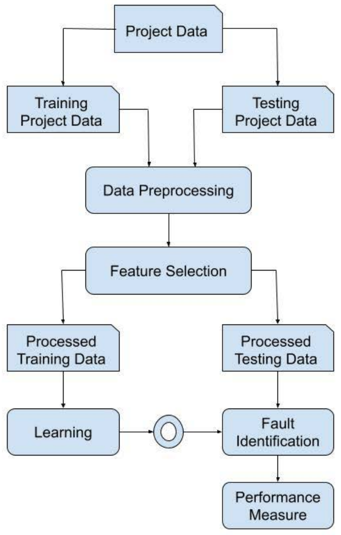


FIGURE 4. Menzies et al.'s Framework (built based on Menzies et al., 2007).

We use Menzies et al.'s framework [62] (Figure 4) of fault prediction to analyze changes at each stage of the framework to improve the CPDP model. The framework includes four stages: project selection, data preprocessing, feature selection, and classification. Our study analyzes the changes in CPDP models over the years in all these stages:

1) **Project selection:** In CPDP, projects consist of two types, namely training projects, and testing projects. The datasets from both the training projects and testing projects are extracted in form of software metrics. The datasets extracted from the training projects are utilized to train the fault prediction model. The appropriate training projects identification to predict faults in a specific target project is also challenging in the cross-project fault prediction paradigm. Thomas Zimmermann et al. [20] evaluated the performance of CPDP on 622 different combinations of source and target projects. This research considered a source project as a strong predictor on a target project when precision, recall, and accuracy are over 0.75. It shows that only about 3% of combinations of source and target projects achieved the required performance. For instance, Firefox is a strong predictor for Internet Explorer but the opposite is not true. Therefore, we analyze the approaches to the identification of suitable source projects as predictors for a specific target project.
2) **Data Preprocessing:** The raw data collected may consist of irrelevant data, noisy data, etc. Therefore, it is required to clean the data through preprocessing to achieve the best possible performance of the

model. Data preprocessing is a data mining technique used to transform the raw data into a meaningful format improving the performance of classifiers. In our study, data preprocessing consists of three components: data normalization, data imbalance, and data filtering. We discuss more about those components in the following section. Tim Menzies et al. [63] identified data preprocessing as more important than the choice of the classifier for the overall performance of the model. Therefore, we analyze the application of data preprocessing approaches over the years.

3) **Feature Selection:** It is a process to reduce the number of features to reduce computational cost and to improve the performance of the model. It reduces data discrepancy by reducing unstable and redundant features. Moreover, in CPDP, it is not always possible to collect the same set of software metrics for both source and target projects. Therefore, it is possible to use feature selection [64] to extract a similar set of metrics from both source and target projects. We analyze the application of different feature selection approaches in CPDP over the years.
4) **Classification:** The defect prediction is a binary classification of software artifacts [65] (faulty and non-faulty).There are many papers in CPDP focusing on improving the performance of the overall model by selecting and improving appropriate classifiers. Therefore, we analyze the application of different classifiers over the years in CPDP.

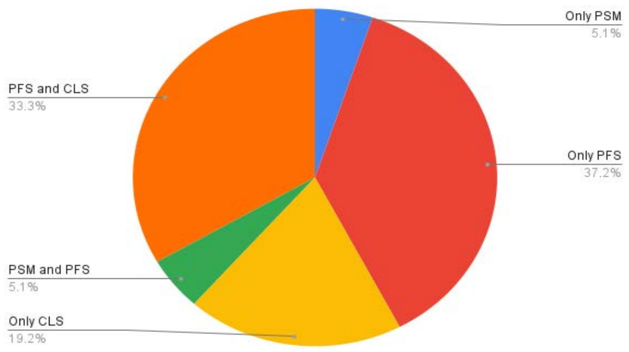


FIGURE 5. Papers focusing on different stages of the Menzies et al.'s framework.

In the following sub-sections, we present the different approaches of CPDP model considering the four stages: project selection, data preprocessing, feature selection, and classification stages. Therefore, we identify how the CPDP models develop in each stage. Moreover, we also present the evolution of the classifier over the base line classifier. Figure 5 shows the percentage of papers involved in the different stages of the framework. We represent those stages as projects and metrics selection (Only PSM), data preprocessing and feature selection (Only PFS), and classification (Only CLS). We noticed that the CPDP studies are based on: Only PSM, Only PFS, Only CLS, the combination of PFS and CLS, and combination of PSM and PFS. Most of the studies tried to improve performance based on the only PFS (37%) followed by combined PFS and CLS (33%).

1) PROJECTS AND METRICS SELECTION

In CPDP models, datasets used in the training phase are likely to be different that the ones available in the target project and the selection of the metrics also impacts the performance of the model. Raimund Moser et al. [66] analyzed the Eclipse project and they found out that process metrics were a better choice than code metrics. Moreover, due to data divergence among source and target projects, it is difficult to achieve reasonable performances. Thomas Zimmermann et al. [20] evaluated the performance of CPDP models on 622 different combinations of source and target projects. Only 3.4% combination of source and target project actually provide a sufficient level of performance to be useful in practice. Therefore, we analyzed the studies focusing to improve the performance of the prediction model based on projects and software metrics selection. We identified eight studies and Table 4 provides a short overview. All the studies are based on supervised learning.

Zimmermann et al. [20] experimented with large-scale Cross-Project Defect Prediction Models on 12 real-world applications with 622 combinations of source and target projects. They investigated whether any source project-based training model can predict fault to any target project. They used both proprietary projects (Microsoft's Internet Explorer (IE), Direct-X, Internet Information Services (IIS), Windows Clustering, Windows Printing, Windows File System, Windows Kernel, and SQL Server 2005) and open source ones (Apache's Derby and Tomcat, Firefox, and Eclipse). Metrics used in this study are Added LOC/Total LOC, Deleted LOC/Total LOC, Modified LOC/Total LOC, Pre-release bugs/Total LOC, (Added + Modified + Deleted LOC) / (Commits + 1) and Cyclomatic complexity/Total LOC. They considered a project as a strong defect prediction for another target project when precision, recall, and accuracy are above 0.75. Only 21 (3.4%) combinations out of 622 met such levels.

Rahman et al. [67] investigated whether the performance of CPDP depends on size-based metrics. The projects used in this study were from the Apache git repository (Axis2, CXF, Camel, Cayenne, Derby, Lucene, OpenEJB, Wicket, and XercesJ). They extracted the defects information about the modules from the related Jira issue tracking system. The metrics used in this research are Commits, Active Developers, Added LOC, Deleted LOC, Changed LOC, NEW Features Added, Improvements, Log SLOC, and many more. The results showed that size-based metrics perform worse.

Zhang et al. [55] proposed a universal defect prediction model instead of separate models for individual projects. The universal defect prediction model is a combination of within-project and cross-project defect prediction models. The proposed model is based on context-aware rank transformations to cluster projects based on similarity with rank transformation to manage diverse datasets distribution. It uses code metrics at file level, class level, and method level. The chosen code metrics are LOC, Comment Lines, Number of Statements, Number of Functions, Weighted Methods per Class, Depth of Inheritance Tree, Response For a Class, Number of Immediate Sub-classes, Coupling Between Objects, Lack of Cohesion in Methods, Number of instance variables, Number of instance methods, and many more. They found out that the universal defect prediction model performs better using code metrics, process metrics, and contexts together compared to using only code metrics or, code metrics and process metrics. Moreover, the within-project model performs better than the universal model but the universal model achieve reasonable results in the cross-project model.

Aarti et al. [68] investigated defect prediction models based on cross-projects and mixed-project (using both class level and method level metrics) model. The mixed-project model is the combination of both within-project and cross-project models. The datasets collected from the PROMISE repository in this study are NASA, SOFTLAB, and JURECZKO. They concluded that cross-project prediction models with common features for method-level prediction models achieve significantly better performances.

Wen et al. [70] conducted an empirical study on feature and project selection and proposed an approach based on TCA+ transfer learning. Moreover, they proposed the MZTCA+ (median_zscore TCA+) approach for cross-project models. This study used mean_log, std_log, median_log, median_zscore, and TDS for source project selection. The study uses the JURECZKO dataset with the following metrics: Weighted methods per class, Depth of Inheritance Tree, Number of Children, Coupling between objects, Response for a Class, Lack of cohesion in methods, Lack of cohesion in methods, Number of Public Methods, Data Access Metric, and many more. They concluded that the proposed combination of source project selection approach and TCA+ perform better compared to other approaches such as only TCA+.

Liu et al. [71] proposed a two stages cross-project prediction model. They select the two most suitable source projects at the first stage. After that, they apply TCA+ on both project separately to train the model. The selection of the projects is based on values of f1-score and cost-effectiveness. The JURECZKO dataset was used in this study. The metrics included are Number of methods in the class, Depth of inheritance tree, Number of children, Coupling between objects, Response for a class, Lack of cohesion in methods, and many more. The concluded that the proposed approach outperforms other models such as TCA+, TDS, LT, and Dycom in terms of F1-score and cost-effectiveness and it solves the instability problem of TCA+.

Agrawal et al. [39] analyzed the feasibility of training and test datasets for cross-project defect prediction. This research collected datasets from 13 open source projects from Sourceforge. The metrics extracted for this analysis are Average of McCabe's cyclomatic complexity, Coupling between object, Number of Children, Number of instance methods, Number of instance variables, Response for a Class, Number of Public Methods, Lines of Code, Maximum of McCabe's cyclomatic complexity, Depth of Inheritance Tree, Lack of cohesion in methods, and Weighted methods per class. They concluded that defects from large software systems can not be a good predictor for small software systems and datasets with the large differences in the number of classes can not be used in cross-project defect prediction.

Asano et al. [72] attempted to identify suitable training projects to improve the performance of the model. They used the Bandit Algorithm to identify the most suitable project. Moreover, they also compared the performance of the proposed approach with other baseline approaches. They concluded that the proposed approach was not able to improve the performance significantly in terms of AUC, and f1-score.

TABLE 4. Project and metrics selection.

Techniques	Type	Results
Experiment whether all Projects can use CPDP (2009) [20]	Supervised Learning	It showed only 3.4% source and target project combination can achieve over 75% performance rate.
Experiment Performance on Product Metrics (2012) [67]	Supervised Learning	It concluded that Product Metrics (Size of Code) perform worse in cost-sensitive CPDP.
Context-aware rank transformations for Universal Defect Prediction (combine within and cross project) model (2014) [55]	Supervised Learning	It perform well using code metrics, process metrics and contexts. This model perform well using external data compare to internal data.
Mixed-Project (Using Both Class and Method Level Metrics) CPDP (2017) [68]	Supervised Learning	It concluded that performance was well using common feature of all projects.
Source Project Selection (2019) [70]	Supervised Learning	It concluded that median_zscore is the best approach for source project selection.
F1-score and PofB20 based Source Project Selection (2019) [71]	Supervised Learning	It applied this techniques to two TCA+ approach CPDP.
Feasibility of Source and Target Datasets (2019) [39]	Supervised Learning	It concluded that Large Software system can't be a good predictor for small software system.
Bandit Algorithm (BA) to identify a suitable external project (2021) [72]	Supervised Learning	The BA-based project selection approach did not perform consistently well.

We found a significantly smaller number of studies focusing on the software project and metrics section-based approaches to improve the performance of the prediction model. According to our knowledge, Zimmermann et al. [20] first find out the necessity of identification of suitable source projects for any specific target project. Thereafter, we identified five studies that tried to choose suitable source projects which were identified by Thomas Zimmermann et al. According to our knowledge, we found one study (i.e., Aarti et al. [68]) that experimented on file level, class level, and method level software metrics and measure their performance accordingly.

2) DATA PREPROCESSING AND FEATURE SELECTION

The quality of data is important to achieve good performances in the prediction model. Moreover, it is important also to identify the features that contribute to achieve such performances avoiding to include in the models features that provide no or little contribution but make the model more complex. Additionally, not all metrics are suitable defect predictors. The identification of the most suitable metrics (either automatically or manually) has become one of the dominant aspects of CPDP.

a: DATA PREPROCESSING APPROACHES

The performance of a CPDP model depends on the quality of the data. Additionally, source and target data are collected from different projects making them different. Therefore, preprocessing becomes really important to achieve good performances. We divided the preprocessing approaches into three different sub-categories, namely data normalization, data imbalance, and data filtering.

The datasets in CPDP are complex and heterogeneous. Moreover, each software metric is calculated in different ways with different bounds and scales. Transfer learning approaches are also proposed by various researchers to reduce heterogeneity among the source and target data. We also consider transfer learning approaches in the normalization of data.

The approaches applied to normalize the data are:

- log transformation [57], [70], [74], [75], [76], [77], [78], [79], [80]
- Z-Score Transformation [54], [56], [60], [70], [71], [75], [77], [78], [81], [82], [83]
- mean-median-standard deviation [54], [56], [70], [71], [82], [84]
- context-aware-rank transformations [55]
- multi-collinearity [85]
- min-max approach [56]
- inter-quartile-range approach [56]
- data-discretization for skewed data [64]
- credibility factor based data re-weighting [86]
- rank transformation [78]
- box-cox transformation [78]
- multiple-components weights with kernel mean matching algorithm [87]
- vectorization based on Abstract Syntax Tree [88], [89]
- balanced distribution adaptation [90]
- discriminant subspace alignment [83]
- TrAdaBoost method based weight measure [91]

Log transformation, z-score transformation, and mean-median-standard deviation are the most used approaches. When dealing with the transfer learning approach, researchers try to reduce the divergence between source and target projects using the transfer component analysis [54]. It consists of a combination of log transformation, z-score transformation, and mean-standard deviation transformation. Therefore, we analyze those transformations separately. Additionally, the imbalanced data also impact the performance of a prediction model. Faulty and non-faulty modules in the training datasets are not equally distributed, affecting the performance of a model. Different approaches are applied to address the data imbalance problem:

- data weighting [76], [84]
- undersampling [64] (e.g., Tomek links [81])
- re-sampling [85]
- value aware boosting [56]
- multi-objective classifier [77]
- semi-supervised transfer component analysis [60]
- over-samples (e.g., SMOTE [81], [87], [92])
- SMOTE-PENN [79]
- class distribution estimation with synthetic minority [93])
- stratification embedded in nearest neighbor [82]
- improved K-Means clustering cleaning [80]
- ada-boost technique [40]

We found out that over-sampling is the most used approach to manage the data imbalance problem. Moreover, data filtering approach is widely applied to choose a similar set of data from source project and to remove noisy data from the training data to increase the performance of the model. The subset of training data selected is based on the specific target project. Different approaches are applied for data filtering:

- nearest-neighbor [64], [70], [74], [84], [94], [95], [96] (e.g., using Minimum Hamming Distance [57], using genetic instance selection [97], using Euclidean Distance [98], using fuzzy-rough instance selection [92])
- EM-clustering [70], [84]
- k-means [94]
- instance clustering [31]
- metric violation scores [31]
- burak-filter [76], [91], [96]
- density-based spatial clustering [99]
- mahalanobis distance [57]
- similarity weight computation [81] (e.g., dataset characteristic vector [78])
- patterns by ordered projections [100]
- Bayesian formula based filtering [101]
- harmony search algorithm [77]
- hierarchical select based filter [23]
- global filter [96]
- local filter [96]
- peters filter [96]
- kawata filter [96]
- He filter [96]
- HeBurak filter [96]
- HePeters filter [96]
- mean_log-std_log-median_log-median_zscore [70]
- improved K-Means clustering cleaning approach [80]
- iForest machine learning method [38]

Nearest-neighbor is the most used filtering approach in CPDP models. CPDP studies attempt to improve the

TABLE 5. Frequencies of different preprocessing approaches.

Techniques	Frequency	References
Data Normalization	11	[20], [54], [55], [71], [75], [78], [83], [86], [90], [102], [103]
Data Imbalance	2	[93], [104]
Data Filtering	10	[23], [38], [94], [96], [97], [99], [100], [101], [105]
Feature Selection	9	[37], [58], [106], [107], [108], [109], [110], [111], [112], [113]
Data Normalization, and Data Imbalance	5	[56], [79], [82], [85], [87]
Data Normalization, and Feature Selection	4	[88], [89], [114], [115]
Data Normalization, and Data Filtering	4	[57], [74], [78], [91]
Data Imbalance, and Data Filtering	1	[92]
Data Filtering, and Feature Selection	3	[31], [95], [98]
Data Imbalance, and Feature Selection	1	[40]
Data Normalization, Data Imbalance, and Data Filtering	5	[76], [77], [80], [81], [84]
Data Normalization, Data Imbalance, and Feature Selection	2	[60], [116]
Data Normalization, Data Filtering, and Feature Selection	1	[70]
Data Normalization, Data Imbalance, Data Filtering, and Feature Selection	1	[64]

performance of the model based on the combination of different approaches to preprocessing. Table 5 shows the different approaches applied in different studies. We found out that most of the studies are based on normalization, filtering, and their combinations. Just a few studies focus on solving the data imbalance problem. Additionally, Figure 6 shows the frequency of different approaches of data preprocessing over the years. In this figure, we consider all approaches separately. For instance, if any study applied both normalization and filtering approaches, we consider normalization and filtering separately in the figure.

b: FEATURE SELECTION APPROACHES

The approaches applied for feature selection are:

- feature subset selection with Bagging [64]
- subset of relevant features [114]
- metric violation scores [31]
- deep belief network based on abstract syntax tree [106], [111]
- correlation-based feature selection for feature subset selection [98], [109]
- improved subclass discriminant analysis [60]
- information flow algorithm [95]
- feature selection using clusters of hybrid-data approach [58]
- top-k feature subset based on number of occurrences of different metrics [107]
- geodesic flow kernel feature selection [108]
- similarity measure [109]
- correlation [70], [109]
- gain ratio [70], [109]
- reliefF [70]
- InfoGain [70]
- OneR [70]
- Symmetrical Uncertainty [70]
- tree-based-embedding convolutional neural network with transferable hybrid feature learning [88]
- bi-directional long short-term memory based automatic feature selection from vectorized token sequences [89]
- source code image generation based on source code visualization to extract feature [110]
- chi-square feature selection technique [40]
- Spearman's Rank Correlation [40]
- adaptive distributed convolutional neural network based abstract syntax trees [37]

Figure 6 shows the frequency of feature selection approaches applied over the years to improve the performance of the prediction model. Most of the studies used open source datasets and selected common software metrics for both source and target projects manually, instead of using an automatic feature selection method.

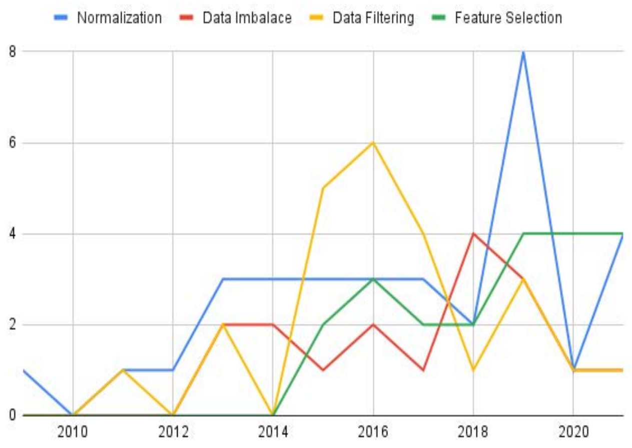


FIGURE 6. Frequency of approaches of data preprocessing and feature selection.

Most of the CPDP models tried to improve the performance of the model by using data preprocessing, including project selection and feature selection approaches. Figure 7 shows the frequency of the combination of preprocessing and feature selection approaches from 2009 to 2021. According to our findings, 2019 was the year with the highest number of publications. Figure 8 shows that most of the studies on preprocessing and feature selection approaches are based supervised learning. Figure 9 shows the percentage of supervised, semi-supervised, and unsupervised learning applied over the years in preprocessing and feature selection.

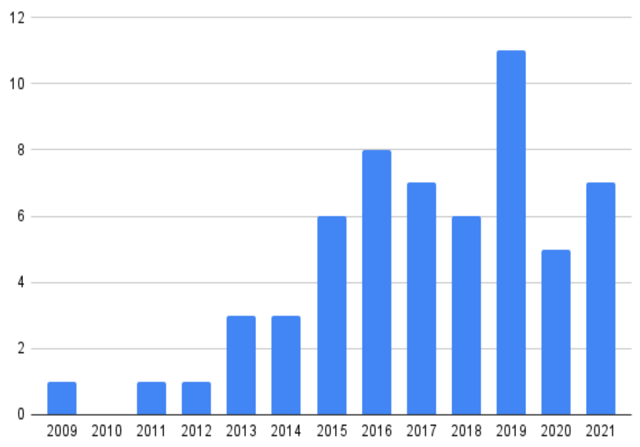


FIGURE 7. Frequency of data preprocessing and feature selection papers.

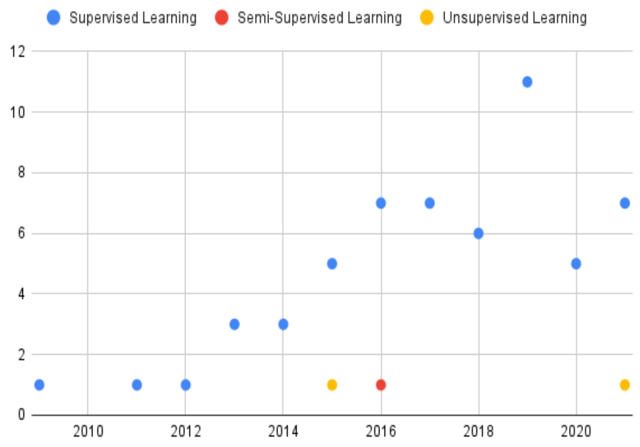


FIGURE 8. Frequency of data preprocessing and feature selection techniques over time.

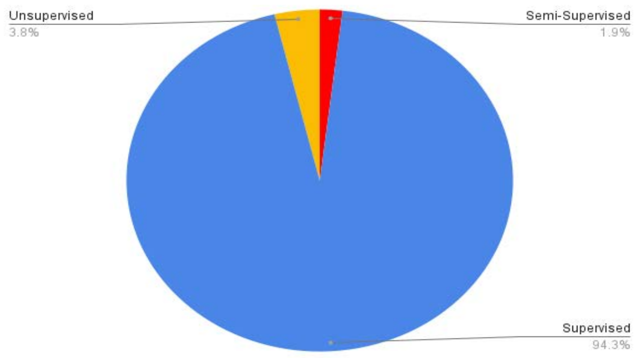


FIGURE 9. Distribution of data preprocessing and feature selection techniques.

3) CLASSIFIERS

The classification is the last step of the prediction model. Over the years, researchers attempted to improve the performance of the CPDP model also by improving the performance of the classifiers. We have identified the following 48 classifiers applied in different studies:

1) Ada Boost (AB)
2) Alternating Decision Tree (ADTree)
3) Artificial Neural Network (ANN)
4) Bayesian Networks (BN)
5) Boost (BT)
6) Boosting-Support Vector Machine (B-SVM)
7) Classification and Regression Tree (CART)
8) Coordinate Ascent (CA)
9) Correlation Metric Selection based Correlation Slignment (CMSCA)
10) Cost-Sensitive Kernelized Semi-Supervised Dictionary Learning (CKSDL)
11) Decision Table (DTa)
12) Decision Tree J48 (DT)
13) Deep Adaptation Networks (DAN)
14) Diffused Bayes Classifier (DBC)
15) Ensemble Learning (EL)
16) Ensemble learning classifier Gradient Boosting (ELGB)
17) Extra Tree (ET)
18) Genetic Algorithm with Ensemble Learning (GAEL)
19) Gradient Boost (GB)
20) Heterogeneous Ensemble (HE)
21) Kernelized Semi-Supervised Dictionary Learning (KSDL)
22) K-Nearest Neighbors (KNN)
23) Kstar(K*) (KS)
24) LambdaMART (LM)
25) ListNet (LN)
26) Logistic Model Tree (LMT)
27) Logistic Regression (LR)
28) Logistic Regression using Genetic Algorithm (LRGA)
29) Multi-Objective Decision Tree (MODT)
30) Multi-Objective Logistic Regression (MOLR)
31) Multi-Objective Naive Bayes (MONB)
32) Multi-Objective Naive Bayes with Nearest Neighbors (MONNB)
33) Multilayer Perceptron (MLP)
34) Naive Bayes (NB)
35) oneR (OR)
36) Radial Basis Function Network (RBF)
37) Random Forest (RF)
38) RankNet (RN)
39) Ridge (RE)
40) Semi-Supervised Structured Dictionary Learning (SSDL)
41) Semi-Supervised Dictionary Learning (SDL)
42) Spectral Clustering (SC)
43) Spotted Hyena Optimizer (SHO)
44) Support Vector Machine (SVM)
45) Support Vector Machine (SVM) with RBF kernel (RBF-SVM)
46) Token Based Classifier (TBC)
47) Transfer Naive Bayes (TNB)
48) Value Aware Boosting with Support Vector Machine (VAB-SVM)

We found out that Logistic Regression (LR), Naive Bayes (NB), and Random Forest (RF) are the most used classifiers, as displayed in Figure 10. Figure 11 shows that LR, NB, and RF are applied in 21%, 16.1%, and 10.2% of the studies respectively. There are several classifiers that were experimented with only once. Figure 11 shows the percentage of the classifiers used.

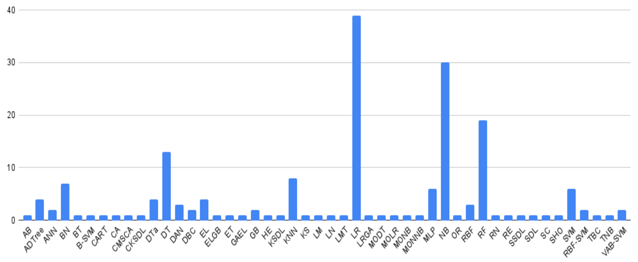


FIGURE 10. Frequency of classifiers usage.

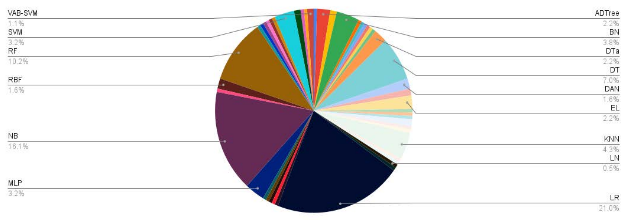


FIGURE 11. Percentage of Applied Classifiers.

The classifier is an important component of a CPDP model but not all the studies tried to improve them to achieve better overall predictions. In our study, we grouped the classification-based studies into three categories:

1) **Comparison among classifiers**
In this category, the studies compared the prediction performance of different classifiers to identify the most suitable classifiers for a given dataset.
Abunadi et al. [117] conducted experiments on open source datasets, namely Drupal, Moodle, and PHPMyAdmin. They attempted to identify suitable classifiers among five popular classifiers: naive bayes, logistic regression, support vector machine, decision tree J48, and random forest. They showed that in the Drupal dataset, the decision tree J48 performs better in terms of precision, and random forest performs better in terms of recall, and f-measure. In the Moodle dataset, decision tree J48, and random forest perform similarly in terms of precision but random forest performs better in terms of recall, and f-measure. In the PHPMyAdmin datasets, random forest is the best performing classifier in terms of precision, recall, and f-measure. It concludes that decision tree J48, and random forest are the most suitable classifiers compared to naive bayes, logistic regression, and support vector machine.
Kaur et al. [100] attempted to understand whether it is possible to predict defects in real-world software projects, using academic projects as training data. They collected academic projects as training data and open source real-world projects as test data from the PROMISE repository. They attempted to identify the most suitable classifier among naive bayes, bayes network, random forest, logistic regression, Kstar, and the heterogeneous ensemble of these five classifiers. They concluded that bayes network is the best performing classifier followed by the heterogeneous ensemble in terms of mean AUC.
Sohan et al. [104] attempted to measure the performance of imbalanced data. They used the SeaCraft repository. They attempted to identify the most suitable classifier among decision tree, random forest, extra tree, ada boost, gradient boost, nave bayes, nearest neighbors, and artificial neural network. They found that gradient boost is the best performing classifier in case of imbalance training data and imbalance test data in terms of precision, recall, f1-score, accuracy, and AUC. Extra tree and gradient boost are the best performing classifier in case of imbalanced training data and balance test data in terms of precision. Ada boost is the best performing classifier in case of imbalanced training data and balance test data in terms of recall, f1-score, and AUC. The decision tree is the best performing classifier in case of imbalanced training data and balance test data in terms of accuracy. Gradient boost is the best performing classifier in case of balance training data and imbalance test data in terms of precision, recall, f1-score, accuracy, and AUC. Random forest is the best performing classifier in the case of balance training data and balance test data in terms of precision. Ada boost is the best performing classifier in case of balance training data and balance test data in terms of recall, f1-score, accuracy, and AUC.
Li et al. [118] attempted to compare the performance of the combination of different popular transfer learning approaches and classification techniques. The used the AEEEM, ReLink, and JURECZKO datasets. They used transfer learning methods for this experiment, namely Bruak filter, DS, DSBF, TCA, DBSCAN filter, Universal, DTB, and Peter filter. They used the following classifiers: k-nearest neighbor, boost, classification and regression tree, random forest, support vector machine, multi-layer perceptron, ridge, and naive Bayes. They concluded that the combination of the DTB transfer learning approach with random forest classifier performs the best.

2) **Performance of proposed approach in different classifiers**
Different studies proposed different approaches over the years to improve the performance of CPDP by improving the data preprocessing, feature selection, and project selection techniques. These studies

experimented their approaches on multiple classifiers to identify the most suitable ones.

Herbold et al. [84] investigated the appropriate training data selection based on distance measurements (i.e., EM-clustering and Nearest Neighbor algorithm). Moreover, they analyzed the performance of selected training data on different classifiers: logistic regression, naive bayes, bayesian networks, SVM with RBF kernel, C4.5 decision trees, random forest, and multilayer perceptron. They used the PROMISE repository. They concluded that the performance of classifiers depend on the performance of data preprocessing. In this experiment they identified SVM with RBF kernel as the best combination. He et al. [64] investigated whether it is possible to use open source projects to predict defect on proprietary ones. They proposed a novel techniques to reduce data distribution among source and target data. They applied undersampling to reduce data imbalance, data normalization to scale data, data discretization, and nearest-neighbor filter for preprocessing, and applied feature subset selection for feature selection. They used the PROMISE repository. Moreover, they analyzed the performance of proposed techniques on different classifiers, namely random forest, naive bayes, and logistic regression, to identify the most suitable one. They concluded that naive bayes is the most suitable classifier for their approach.

Qing et al. [94] proposed a feature based transfer learning approach to reduce data divergence between source and target projects. They applied the kernel based principal component analysis for transfer learning to reduce data divergence. In addition, they experimented on feature selection approaches, namely chi-square, Info Gain, Forward Selector, and backward elimination method. They used the PROMISE repository. Moreover, they analyzed the performance of the proposed techniques on different classifiers: naive bayes, decision tree J48, and oneR. They identified the Info Gain feature selection approach with decision tree J48 as the best performers in terms of f-measure and AUC.

Nam et al. [31] proposed a novel approach called CLA (Clustering instances and LAbeling instances in clusters) and CLAMI (Clustering instances, LAbeling instances in clusters, Metric selection, and Instance selection) to predict defect based on unlabeled datasets. This is the first study which focused on unsupervised learning. This study used clustering instances in preprocessing and metric violation score based instance and feature selection. It used the NetGene and ReLink datasets. Moreover, it analyze the performance of proposed techniques on different classifiers: logistic regression, bayesian network, J48 decision tree, logistic model tree, naive bayesian, random forest, and support vector machine. Furthermore, it identified that CLAMI model with logistic model tree classifier outperforms CLAMI model with other classifiers in terms of AUC. Moreover, CLAMI model with bayesian network, logistic regression, and random forest performs equally and better than other classifiers in terms of f-measure.

Amasaki et al. [76] experimented the effects of data simplification on CPDP. They compared the performance of the combination of four classifiers, namely logistic regression, random forests, naive bayes and SVM with RBF kernel, with two data selection approaches: burak-filter and cross-project selection. They used the PROMISE repository. They compared the performance of combination of burak-filter with four classifiers, combination of burak-filter and data simplification with four classifiers, combination of cross-project selection with four classifiers, and combination of cross-project selection and data simplification with four classifiers, to measure the most suitable classifiers for data simplification approach. They concluded that the combination of data simplification and cross-project selection with logistic regression outperforms other combinations of classifiers in terms of f-measure and AUC.

Kawata et al. [99] proposed a novel data filtering approach based on DBSCAN (Density-Based Spatial Clustering) to reduce data divergence between source and target projects. They used the PROMISE repository. They compared the performance of the DBSCAN data filtering approach with four different classifiers: logistic regression, random forests, naive bayes, and k-nearest neighbors (with $k = 1$). They concluded that the DBSCAN filter perform better with logistic regression and KNN classifiers but it performs worse with random forests and naive bayes classifiers.

Yu et al. [98] investigated the performance of feature selection and instance selection approaches. They used the PROMISE repository. This study used nearest neighbor for instance selection and correlation-based feature selection approach. It also used four classifiers (k-nearest neighbors, logistic regression, multiLayer perceptron, and naive bayes) to compare the performance feature selection, instance selection and a combination of both. It identified that prediction model based on combination of feature selection with four classifiers perform better compared to model based on only classifiers and a combination of instance selection with four classifiers in terms of AUC. Moreover, the prediction model based on combination of feature selection with four classifiers, model based on combination of feature selection followed by instance selection with four classifiers, and model based on combination of instance selection followed by feature selection with four classifiers, perform almost equally in terms of AUC. All the four classifiers perform almost equally in terms of AUC.

Li et al. [23] compare the previously proposed data filtering approaches and proposed a novel data filtering approach. They used the PROMISE repository. This study proposed a novel hierarchical selection-based filter to select the most suitable instances. Moreover, it compared the performance hierarchical selection-based filter with two different classifiers: naive bayes and support vector machine. It found out that the proposed approach performs better with naive bayes compared to support vector machine.

Hosseini et al. [119] measured the performance of search-based datasets selection and the genetic instance selection approach. They used the PROMISE repository. They compared the performance of search-based instance selection and genetic instance selection approach with two classifiers: naive bayes and J48 decision tree. They concluded that search-based instance selection with naive bayes classifier and genetic instance selection with naive bayes classifier perform better compared to search-based instance selection with J48 decision tree classifier and genetic instance selection with J48 decision tree classifier.

Limsettho et al. [93] proposed a novel approach to reduce class imbalance and data distribution difference. They used the PROMISE repository. They applied the proposed approach called Class Distribution Estimation with Synthetic Minority Oversampling Technique (CDE-SMOTE) on different classifiers (J48 decision tree, random forest, naive bayes, logistic regression, k-nearest neighbors, vote ensemble (J48 decision tree and naive bayes), and vote ensemble (J48 decision tree, naive bayes and k-nearest neighbors)) to identify the most suitable classifier for the proposed approach. They concluded that naive bayes and vote ensemble (J48 decision tree, naive bayes and k-nearest neighbors) are the most suitable classifiers.

Yu et al. [109] explored the effectiveness of feature subset selection and feature ranking approach. They used NASA and PROMISE datasets. They compared the performance of the feature subset selection approach and the feature ranking approach using k-nearest neighbors and naive bayes classifiers. They found out that the feature subset selection approach perform better with naive bayes classifier compared to k-nearest neighbor classifier.

Qiu et al. [120] proposed a novel distribution adaptation approach called joint distribution matching (JDM) to reduce data divergence between source and target projects. They used AEEEM and PROMISE datasets. They compared the performance of JDM with logistic regression classifier, kernel mean matching (KMM), semi-supervised TCA (SSTCA), transfer learning approach (TCA), joint distribution adaptation (JDA), k-nearest neighbours filter, data gravitation, and only logistic regression classifier in terms of f-measure and AUC. Additionally, they compared the performance of JDM with random forest classifier, with kernel mean matching (KMM), semi-supervised TCA (SSTCA), transfer learning approach (TCA), joint distribution adaptation (JDA), k-nearest neighbours filter, data gravitation, and only random forest classifier in terms of f-measure and AUC. The proposed JDM approach yields better performance compare to other approaches in both logistic regression and random forest classifier. This study suggested to apply any well known classifiers with JDM to achieve better performance.

Cui et al. [38] proposed the isolation forest (iForest) filter to reduce dissimilarity of data distribution between source and target projects. They used the PROMISE repository. They compared the performance of proposed iForest filter with burak filter and peter filter. Moreover, they measured the performance of these three filters with different classifiers: naive bayes, decision tree, logistic regression, k-nearest neighbor, and random forest. They concluded that iForest filter with random forest classifier and iForest filter with naive bayes classifier perform better in terms of AUC and f-measure. Moreover, they identified burak filter with naive bayes classifier perform better in terms of g-measure, g-mean, and balance.

Vashisht et al. [40] explored the impact of heterogeneous software metrics in source and target projects. They used AEEEM, ReLink, and SOFTLAB datasets. They measured the performance of three classifiers: gradient boosting method, naive bayes, and logistic regression. They found out that gradient boosting method is the most suitable classifier in terms of accuracy, f-measure and AUC.

3) Evolution over Baseline Classifiers

We consider conventional machine learning classifiers as baseline classifiers to show the evolution. The conventional machine learning classifier is the simplest version of any classification model. For instance, we consider naive bayes classifier as a baseline classifier and we consider diffused naive bayes classifier as an evolution over the baseline one. Figure 12 shows the evolution of classifiers over baseline ones. The right arrow ($\rightarrow$) in the figure represents the development of the baseline classifier to the higher level classifier. However, not all baseline classifiers evolved in CPDP models. The following classifiers did not develop over time in CPDP:

- Classification and Regression Tree
- Coordinate Ascent
- Extra Tree
- K-Nearest Neighbors
- Kstar(K*)
- LambdaMART
- ListNet
- Logistic Model Tree
- oneR
- Random Forest
- RankNet

- Ridge
- Spectral Clustering
- Spotted Hyena Optimizer
- Token Based Classifier

The baseline classifiers which evolved over time in CPDP are:

- Artificial Neural Network [68]
- Boosting [121]
- Bayesian Networks [84]
- Decision Table [122]
- Decision Tree [84]
- Ensemble Learning [123]
- Logistic Regression [84]
- Naive Bayes [74]
- Radial Basis Function Network [122]
- Semi-Supervised Learning [60]
- Support Vector Machine [56]

We found out that:

- Artificial Neural Network evolved to Multi-layer Perceptron [84] and Deep Adaptation Networks [110]
- Boosting evolved into Ada Boost [104] and Gradient Boost [104]
- Decision Tree evolved into C4.5 Decision Trees [84], Alternating Decision Tree [122] and J48 Decision Trees [94]
- Ensemble Learning evolved into Ensemble Clustering with Labeling [61], Ensemble Learning (Heterogeneous) [100], Ensemble Learning (AdaBoost with Genetic Algorithm) [123], Ensemble Learning on Weight Vote [82]
- Logistic Regression evolved into Logistic Regression (Simple) with one predictor variable [75] Logistic Regression (Multivariate) with Genetic Algorithm [10]
- Naive Bayes evolved into Naive Bayes (Transfer) [56], Bagging or Bootstrap Aggregating (Naive Bayes) [114], Boosting (Naive Bayes) [114], Naive Bayes (Diffused Bayes) [95] and Naive Bayes with Effort-Aware Measure [124]
- Semi-Supervised Learning evolved into Semi-Supervised Structured Dictionary Learning [59], Semi-Supervised Dictionary Learning [125], Kernelized Semi-supervised Dictionary Learning [125] and Cost-Sensitive Kernelized Semi-supervised Dictionary Learning [125]
- Support Vector Machine evolved into Support Vector Machine with Boosting [56], Support Vector Machine with Radial Basis Function kernel [84], Support Vector Machine with Value Aware Boosting [56] and Weighted Support Vector Machine [91]
- J48 Decision Trees evolved into Bagging or Bootstrap Aggregating (J48 Decision Tree) [114] and Boosting (J48 Decision Tree) [114]

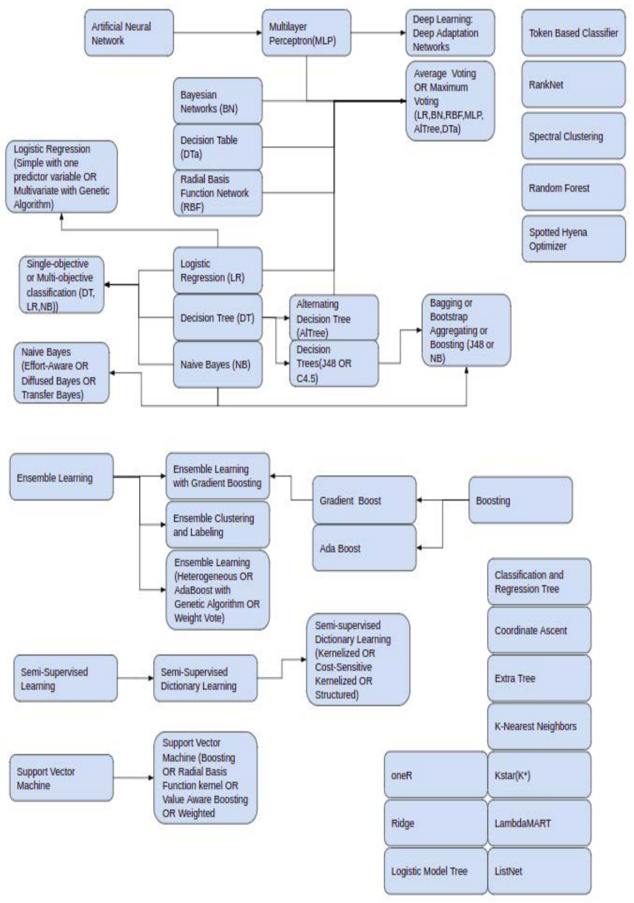

Artificial Neural Network
Multilayer Perceptron(MLP)
Deep Learning: Deep Adaptation Networks
Token Based Classifier
Bayesian Networks (BN)
Average Voting OR Maximum Voting (LR,BN,RBF,MLP, AlTree,DTa)
RankNet
Logistic Regression (Simple with one predictor variable OR Multivariate with Genetic Algorithm)
Decision Table (DTa)
Spectral Clustering
Radial Basis Function Network (RBF)
Random Forest
Spotted Hyena Optimizer
Single-objective or Multi-objective classification (DT, LR,NB))
Logistic Regression (LR)
Decision Tree (DT)
Alternating Decision Tree (AlTree)
Bagging or Bootstrap Aggregating or Boosting (J48 or NB)
Naive Bayes (Effort-Aware OR Diffused Bayes OR Transfer Bayes)
Naive Bayes (NB)
Decision Trees(J48 OR C4.5)
Ensemble Learning
Ensemble Learning with Gradient Boosting
Gradient Boost
Boosting
Ensemble Clustering and Labeling
Ada Boost
Ensemble Learning (Heterogeneous OR AdaBoost with Genetic Algorithm OR Weight Vote)
Classification and Regression Tree
Coordinate Ascent
Semi-supervised Dictionary Learning (Kernelized OR Cost-Sensitive Kernelized OR Structured)
Semi-Supervised Learning
Semi-Supervised Dictionary Learning
Extra Tree
K-Nearest Neighbors
Support Vector Machine
Support Vector Machine (Boosting OR Radial Basis Function kernel OR Value Aware Boosting OR Weighted
oneR
Kstar(K*)
Ridge
LambdaMART
Logistic Model Tree
ListNet

FIGURE 12. Evolution of baseline classifiers.

Moreover, we noticed that many studies applied more than one baseline classifiers to create a new one:

- the combination of Ensemble Learning and Gradient Boosting evolved intto Ensemble Learning with Gradient Boosting [126]
- the combination of Logistic Regression, Bayesian Network, Radial Basis Function Network, Multi-Layer Perceptron, Alternating Decision Trees and Decision Table evolved into Average Voting [121]
- the combination of Logistic Regression, Bayesian Network, Radial Basis Function Network, Multi-Layer Perceptron, Alternating Decision Trees and Decision Table evolved into Maximum Voting [121]
- the combination of Decision Tree, Logistic Regression and Naive Bayes evolved into single-objective classifier [77]
- the combination of Multi-Objective Decision Tree, Multi-Objective Logistic Regression, Multi-Objective Naive Bayes and Multi-Objective Naive Bayes with Nearest Neighbors evolved into Multi-objective classifier [77]

As shown in Figure 9, most of the classification studies are based on supervised learning.

E. RQ5: WHICH ARE THE EVALUATION MEASURES APPLIED TO CPDP MODELS?

Evaluation is an important stage to understand the performance of a prediction model. The preferred evaluation measures used are:

- precision [67]
- recall [84]
- accuracy [127]
- probability of false positive [64]
- true negative rate [119]
- balance [93]
- ROC-AUC [75]
- F-Measure [70]
- G-Measure [55]
- H-Measure [56]
- G-Mean [78]
- Win/Draw/Loss [58]
- false negative rate [119]
- Matthews correlation coefficient [119]

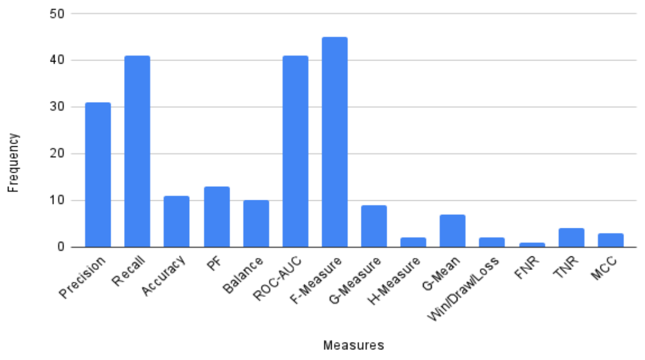


FIGURE 13. Evaluation measures frequency.

Figure 13 shows that precision, recall, ROC-AUC, and F-measure are the preferred evaluation measures. Most of the performance measure are organized using a confusion matrix. Although the researchers choose various evaluation measures to measure the performance of the model, none of the articles explained the purpose of choosing the specific evaluation measure in the context of the CPDP.

F. RQ6: WHICH ARE THE DIFFERENT TOOLS USED IN CPDP MODELS?

Nearly half of the studies did not mention the tools used for their experimentations (Figure 14). Considering only the papers mentioning tools, WEKA is the preferred one by far. Figure 15 shows that WEKA is the most used tool followed by Understand and MATLAB. We identified the following tools used:

- WEKA [128]: It is a machine learning tool that can be used for data preprocessing, feature selection, classification, and validation.
- Understand [129]: It is a static code analysis tool that can generate metrics from source code.

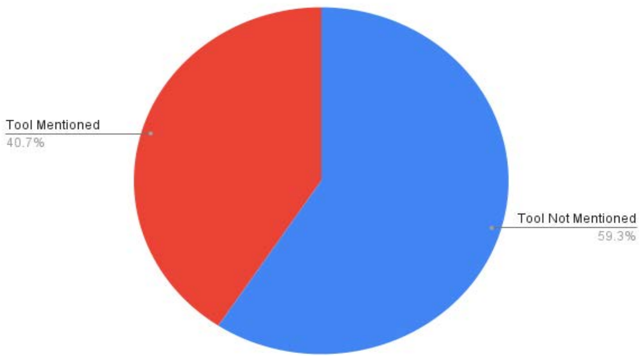


FIGURE 14. Tools mentioned and not mentioned in papers.

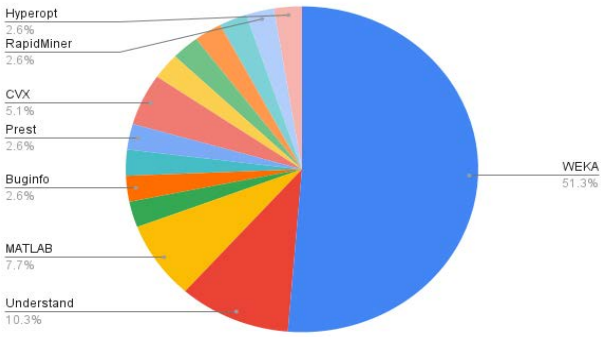


FIGURE 15. Frequency of used tools.

- MATLAB: It has automatic machine learning features like feature selection, classification, and many more. It mainly used for classification purposes.
- Ckjm [130]: The ckjm tool calculates six CK software metrics: WMC, DIT, NOC, CBO, RFC, and LCOM.
- Buginfo: It can identify bugs based on logs information of version control system. This tool has not released officially yet [131].
- MINE [132]: Maximal Information-based Nonparametric Exploration is used to calculate relevance formulas.
- Prest [133]: It extracts source code metrics and call graphs.
- CVX [134]: It is a tool for mathematical optimization.
- Javalang [135]: It is used to parse the Java source code.
- Keras: It is a deep learning tool.
- CrossPare [136]: This tool includes of many state-of-the-art cross-project fault prediction models.
- WALA [137]: The WALA libraries provide static source code analysis capabilities for Java bytecode and related languages and for JavaScript.
- RapidMiner [138]: It is a data science tool that provides an integrated platform for machine learning, test mining, data preparation, deep learning, and prediction analytics.
- Hyperopt [139]: It is a python library for optimizing hyperparameters of machine learning algorithms.

V. RESEARCH GAPS

There is a significant amount of work done in CPDP. Still, the following existing challenges need to be taken into consideration to bring CPDP models into practice:

1) It has been found that most of the CPDP studies use supervised learning. Semi-supervised learning and unsupervised learning have not been explored deeply. There is no clear evidence of which is the best performing approach and in which conditions.
2) The early identification of appropriate projects to train the model can be a game-changer for the overall performances. There is a very limited number of papers dealing with that but they are not sufficient to provide proper guidelines for projects selection.
3) It is not clear which metrics perform better in cross-project settings. Most of the CPDP models are built with the assumption that selected metrics are good for CPDP. Some studies compared the performance between process and product metrics. However, it is required to perform a fine-grained analysis at metric level.
4) The concept of information content may be useful to examine the hidden relationship between projects and features. There are few works based on the concept of information content but they did not investigate the hidden relationship between the projects and between the features.
5) It is clear that almost all the studies tried to improve the performance of CPDP by improving projects selection techniques, automatic feature selection techniques, and proper classifiers. Still, they could not go beyond 70% performance. This level needs to be improved to bring CPDP models into practice.
6) The application of deep learning provides interesting results in many different domains. However, we found just a few recent studies based on deep learning. This area could be explored more.
7) Rule-based soft computing model (e.g., Fuzzy Logic) may be useful in CPDP. Cross-project decision making is a very complex process and rule-based decision-making may give interesting results. This area is not explored yet.
8) None of the studies integrated the reason for the faults (root cause analysis) in their models. The identification of the reason for the fault could provide benefits in practical use of such models.

VI. CONCLUSION

Researchers have worked actively in the area of SDP for the last three decades. Since the last decade, researchers focused more on CPDP. The early-stage identification of faults can reduce risks and costs of production by reducing maintenance costs. In a growing start-up ecosystem and demand for complex systems (e.g., cyber-physical systems, distributed systems, cloud services, etc.), CPDP models can reduce drastically the dependence on within-project datasets that are not always available. In this paper, we analyzed the CPDP appraoches focusing on projects selection, metrics selection, data preprocessing, feature selection, and choice of classification. This investigation can help researchers and practitioners to identify proper approaches to apply in their contexts and improve the state-of-the-art. Finally, we identified some open questions and areas that need more research to improve the performances of CPDP models.

REFERENCES

[1] I. D. Coman and A. Sillitti, "An empirical exporatory study on inferring developpers' activities from low-level data," in *Proc. 19th Int. Conf. Softw. Eng. Knowl. Eng. (SEKE)*, 2007, pp. 15–18.

[2] I. Fronza, A. Janes, A. Sillitti, G. Succi, and S. Trebeschi, "Cooperation wordle using pre-attentive processing techniques," in *Proc. 6th Int. Workshop Cooperat. Hum. Aspects Softw. Eng. (CHASE)*, May 2013, pp. 57–64.

[3] R. Moser, W. Pedrycz, A. Sillitti, and G. Succi, "A model to identify refactoring effort during maintenance by mining source code repositories," in *Proc. 9th Int. Conf. Product Focused Softw. Process Improvement (PROFES)*, 2008, pp. 360–370.

[4] V. Lenarduzzi, A. Sillitti, and D. Taibi, "Analyzing forty years of software maintenance models," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 146–148.

[5] V. Lenarduzzi, A. Sillitti, and D. Taibi, "A survey on code analysis tools for software maintenance prediction," in *Proc. Int. Conf. Softw. Eng. Defence Appl. (SEDA)*, 2018, pp. 165–175.

[6] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Comput.*, vol. 22, no. S1, pp. 77–88, Jan. 2019.

[7] S. Pal, "Generative adversarial network-based cross-project fault prediction," 2021, *arXiv:2105.07207*.

[8] S. Pal and A. Sillitti, "A classification of software defect prediction models," in *Proc. Int. Conf. 'Nonlinearity, Inf. Robot.' (NIR)*, Aug. 2021, pp. 1–6.

[9] T. M. Khoshgoftaar and N. Seliya, "Fault prediction modeling for software quality estimation: Comparing commonly used techniques," *Empirical Softw. Eng.*, vol. 8, no. 3, pp. 255–283, 2003.

[10] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *Proc. IEEE 6th Int. Conf. Softw. Test., Verification Validation*, Mar. 2013, pp. 252–261.

[11] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Eng.*, vol. 14, no. 5, pp. 540–578, Oct. 2009.

[12] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 316–329, May 2007.

[13] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2013, pp. 279–289.

[14] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Programmer-based fault prediction," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng. (PROMISE)*, 2010, pp. 1–10.

[15] P. Singh, "Knowledge transfer software fault learning," in *Proc. Int. Conf. Inventive Comput. Informat. (ICICI)*, Nov. 2017, pp. 1109–1114.

[16] Z. Xu, S. Li, Y. Tang, X. Luo, T. Zhang, J. Liu, and J. Xu, "Cross version defect prediction with representative data via sparse subset selection," in *Proc. 26th Conf. Program Comprehension*, May 2018, pp. 132–13211.

[17] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2013.

[18] B. Caglayan, A. Tosun, A. Miranskyy, A. Bener, and N. Ruffolo, "Usage of multiple prediction models based on defect categories," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng. (PROMISE)*, 2010, pp. 1–9.

[19] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proc. 13th Int. Conf. Softw. Eng. (ICSE)*, 2008, pp. 531–540.

[20] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2009, pp. 91–100.

[21] L. Goel, D. Damodaran, S. K. Khatri, and M. Sharma, "A literature review on cross project defect prediction," in *Proc. 4th IEEE Uttar Pradesh Sect. Int. Conf. Electr., Comput. Electron. (UPCON)*, Oct. 2017, pp. 680–685.

[22] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 811–833, Sep. 2018.

[23] Y. Li, Z. Huang, Y. Wang, and B. Fang, "Evaluating data filter on cross-project defect prediction: Comparison and improvements," *IEEE Access*, vol. 5, pp. 25646–25656, 2017.

[24] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 2, pp. 111–147, Feb. 2019.

[25] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, and B. Xu, "How far we have progressed in the journey? An examination of cross-project defect prediction," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 1, pp. 1–51, Jun. 2018.

[26] Y. Khatri and S. K. Singh, "Cross project defect prediction: A comprehensive survey with its SWOT analysis," *Innov. Syst. Softw. Eng.*, vol. 18, no. 2, pp. 263–281, Jun. 2022.

[27] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "NASA MDP software defects data sets," Figshare Collection, 2018, doi: 10.6084/m9.figshare.c.4054940.v1.

[28] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proc. 7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, May 2010, pp. 31–41.

[29] K. Herzig, S. Just, A. Rau, and A. Zeller, "Predicting defects using change genealogies," in *Proc. IEEE 24th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2013, pp. 118–127.

[30] T. Menzies, R. Krishna, and D. Pryor, "The SEACRAFT repository of empirical software engineering data," 2017. [Online]. Available: https://zenodo.org/communities/seacraft

[31] J. Nam and S. Kim, "CLAMI: Defect prediction on unlabeled datasets (T)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 452–463.

[32] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng.*, 2011, pp. 15–25.

[33] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, May 2008.

[34] T. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[35] S. Purao and V. Vaishnavi, "Product metrics for object-oriented systems," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 191–221, Jun. 2003, doi: 10.1145/857076.857090.

[36] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[37] L. Sheng, L. Lu, and J. Lin, "An adversarial discriminative convolutional neural network for cross-project defect prediction," *IEEE Access*, vol. 8, pp. 55241–55253, 2020.

[38] C. Cui, B. Liu, and S. Wang, "Isolation forest filter to simplify training data for cross-project defect prediction," in *Proc. Prognostics Syst. Health Manage. Conf. (PHM-Qingdao)*, Oct. 2019, pp. 1–6.

[39] A. Agrawal and R. Malhotra, "Cross project defect prediction for open source software," *Int. J. Inf. Technol.*, vol. 14, no. 1, pp. 587–601, Feb. 2022.

[40] R. Vashisht and S. A. M. Rizvi, "Feature extraction to heterogeneous cross project defect prediction," in *Proc. 8th Int. Conf. Rel., InFocom Technol. Optim. (Trends Future Directions) (ICRITO)*, Jun. 2020, pp. 1221–1225.

[41] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.

[42] R. Martin, "OO design quality metrics," *Anal. Dependencies*, vol. 12, no. 1, pp. 151–170, 1994.

[43] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, Jan. 2002.

[44] V. R. Basili and R. W. Reiter, Jr., "Evaluating automatable measures of software development," in *Proc. Workshop Quant. Softw. Models*, 1979, pp. 107–116.

[45] C. T. Bailey and W. L. Dingee, "A software study using Halstead metrics," in *Proc. ACM Workshop/Symp. Meas. Eval. Softw. Qual.*, 1981, pp. 189–197.

[46] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Softw. Eng.*, vol. 17, nos. 4–5, pp. 531–577, Aug. 2012.

[47] G. R. Choudhary, S. Kumar, K. Kumar, A. Mishra, and C. Catal, "Empirical analysis of change metrics for software fault prediction," *Comput. Electr. Eng.*, vol. 67, pp. 15–24, Apr. 2018.

[48] M. Jureczko and L. Madeyski, "A review of process metrics in defect prediction studies," *Metody Informatyki Stosowanej*, vol. 5, pp. 133–145, May 2011.

[49] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Proc. 6th Int. Softw. Metrics Symp.*, 1999, pp. 242–249.

[50] B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love, "Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics," *IEEE Trans. Softw. Eng.*, vols. SE–5, no. 2, pp. 96–104, Mar. 1979.

[51] I. Herraiz, G. Robles, J. M. Gonzalez-Barahona, A. Capiluppi, and J. F. Ramil, "Comparison between SLOCs and number of files as size metrics for software evolution analysis," in *Proc. Conf. Softw. Maintenance Reeng. (CSMR)*, 2006, p. 8.

[52] M. H. Halstead, *Elements of Software Science*. New York, NY, USA: Elsevier, 1977.

[53] E. A. Felix and S. P. Lee, "Predicting the number of defects in a new software version," *PLoS ONE*, vol. 15, no. 3, Mar. 2020, Art. no. e0229131.

[54] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 382–391.

[55] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proc. 11th Work. Conf. Mining Softw. Repositories (MSR)*, 2014, pp. 182–191.

[56] D. Ryu, O. Choi, and J. Baik, "Improving prediction robustness of VAB-SVM for cross-project defect prediction," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng.*, Dec. 2014, pp. 994–999.

[57] D. Ryu, J.-I. Jang, and J. Baik, "A hybrid instance selection using nearest-neighbor for cross-project defect prediction," *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 969–980, Sep. 2015.

[58] C. Ni, W. Liu, Q. Gu, X. Chen, and D. Chen, "FeSCH: A feature selection method using clusters of hybrid-data for cross-project defect prediction," in *Proc. IEEE 41st Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Jul. 2017, pp. 51–56.

[59] F. Wu, X.-Y. Jing, X. Dong, J. Cao, M. Xu, H. Zhang, S. Ying, and B. Xu, "Cross-project and within-project semi-supervised software defect prediction problems study using a unified solution," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 195–197.

[60] X.-Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Trans. Softw. Eng.*, vol. 43, no. 4, pp. 321–339, Apr. 2017.

[61] Y. Yang, J. Yang, and H. Qian, "Defect prediction by using cluster ensembles," in *Proc. 10th Int. Conf. Adv. Comput. Intell. (ICACI)*, Mar. 2018, pp. 631–636.

[62] R. S. Wahono, "A systematic literature review of software defect prediction," *J. Softw. Eng.*, vol. 1, no. 1, pp. 1–16, 2015.

[63] A. Agrawal and T. Menzies, "Is 'better data' better than 'better data miners'?" in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng. (ICSE)*, May/Jun. 2018, pp. 1050–1061.

[64] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Oct. 2013, pp. 45–54.

[65] S. Kumar and S. S. Rathore, *Software Fault Prediction: A Road Map*. Singapore: Springer, 2018.

[66] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. 13th Int. Conf. Softw. Eng. (ICSE)*, 2008, pp. 181–190.

[67] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the 'imprecision' of cross-project defect prediction," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, 2012, pp. 1–11.

[68] G. Sikka and R. Dhir, "An investigation on the effect of cross project data for prediction accuracy," *Int. J. Syst. Assurance Eng. Manage.*, vol. 8, no. 2, pp. 352–377, Jun. 2017.

[69] K. Muthukumaran, A. Choudhary, and N. L. B. Murthy, "Mining GitHub for novel change metrics to predict buggy files in software systems," in *Proc. Int. Conf. Comput. Intell. Netw.*, Jan. 2015, pp. 15–20.

[70] W. Wen, B. Zhang, X. Gu, and X. Ju, "An empirical study on combining source selection and transfer learning for cross-project defect prediction," in *Proc. IEEE 1st Int. Workshop Intell. Bug Fixing (IBF)*, Feb. 2019, pp. 29–38.

[71] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Inf. Softw. Technol.*, vol. 107, pp. 125–136, Mar. 2019.

[72] T. Asano, M. Tsunoda, K. Toda, A. Tahir, K. E. Bennin, K. Nakasai, A. Monden, and K. Matsumoto, "Using bandit algorithms for project selection in cross-project defect prediction," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2021, pp. 649–653.

[73] L. Layman, G. Kudrjavets, and N. Nagappan, "Iterative identification of fault-prone binaries using in-process metrics," in *Proc. 2nd ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, 2008, pp. 206–212.

[74] B. Turhan, A. Tosun, and A. Bener, "Empirical evaluation of mixed-project defect prediction models," in *Proc. 37th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Aug. 2011, pp. 396–403.

[75] S. Uchigaki, S. Uchida, K. Toda, and A. Monden, "An ensemble approach of simple regression models to cross-project fault prediction," in *Proc. 13th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput.*, Aug. 2012, pp. 476–481.

[76] S. Amasaki, K. Kawata, and T. Yokogawa, "Improving cross-project defect prediction methods with data simplification," in *Proc. 41st Euromicro Conf. Softw. Eng. Adv. Appl.*, Aug. 2015, pp. 96–103.

[77] D. Ryu and J. Baik, "Effective multi-objective naïve Bayes learning for cross-project defect prediction," *Appl. Soft Comput.*, vol. 49, pp. 1062–1077, Dec. 2016.

[78] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Softw. Eng.*, vol. 21, no. 1, pp. 43–71, Feb. 2016.

[79] F. Wang, J. Huang, and Y. Ma, "A top-k learning to rank approach to cross-project software defect prediction," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2018, pp. 335–344.

[80] L. Gong, S. Jiang, R. Wang, and L. Jiang, "Empirical evaluation of the impact of class overlap on software defect prediction," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 698–709.

[81] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *Softw. Quality J.*, vol. 25, no. 1, pp. 235–272, Mar. 2017.

[82] L. Gong, S. Jiang, L. Bo, L. Jiang, and J. Qian, "A novel class-imbalance learning approach for both within-project and cross-project defect prediction," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 40–54, Mar. 2020.

[83] Z. Li, C. Qi, L. Zhang, and J. Ren, "Discriminant subspace alignment for cross-project defect prediction," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug. 2019, pp. 1728–1733.

[84] S. Herbold, "Training data selection for cross-project defect prediction," in *Proc. 9th Int. Conf. Predictive Models Softw. Eng.*, Oct. 2013, pp. 1–10.

[85] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proc. 11th Work. Conf. Mining Softw. Repositories (MSR)*, 2014, pp. 172–181.

[86] W. N. Poon, K. E. Bennin, J. Huang, P. Phannachitta, and J. W. Keung, "Cross-project defect prediction using a credibility theory based naive Bayes classifier," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, Jul. 2017, pp. 434–441.

[87] S. Qiu, L. Lu, and S. Jiang, "Multiple-components weights model for cross-project software defect prediction," *IET Softw.*, vol. 12, no. 4, pp. 345–355, 2018.

[88] Z. Cai, L. Lu, and S. Qiu, "An abstract syntax tree encoding method for cross-project defect prediction," *IEEE Access*, vol. 7, pp. 170844–170853, 2019.

[89] H. Li, X. Li, X. Chen, X. Xie, Y. Mu, and Z. Feng, "Cross-project defect prediction via ASTToken2 Vec and BLSTM-based neural network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.

[90] Z. Xu, S. Pang, T. Zhang, X.-P. Luo, J. Liu, Y.-T. Tang, X. Yu, and L. Xue, "Cross project defect prediction via balanced distribution adaptation based transfer learning," *J. Comput. Sci. Technol.*, vol. 34, no. 5, pp. 1039–1062, Sep. 2019.

[91] 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.

[92] A. Bispo, R. Prudencio, and D. Veras, "Instance selection and class balancing techniques for cross project defect prediction," in *Proc. 7th Brazilian Conf. Intell. Syst. (BRACIS)*, Oct. 2018, pp. 552–557.

[93] N. Limsettho, K. E. Bennin, J. W. Keung, H. Hata, and K. Matsumoto, "Cross project defect prediction using class distribution estimation and oversampling," *Inf. Softw. Technol.*, vol. 100, pp. 87–102, Aug. 2018.

[94] H. Qing, L. Biwen, S. Beijun, and Y. Xia, "Cross-project software defect prediction using feature-based transfer learning," in *Proc. 7th Asia–Pacific Symp. Internetware*, Nov. 2015, pp. 74–82.

[95] S. Huang, Y. Wu, H. Ji, and C. Bai, "A three-stage defect prediction model for cross-project defect prediction," in *Proc. Int. Conf. Dependable Syst. Their Appl. (DSA)*, Oct. 2017, p. 169.

[96] Y. Bin, K. Zhou, H. Lu, Y. Zhou, and B. Xu, "Training data selection for cross-project defection prediction: Which approach is better?" in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Nov. 2017, pp. 354–363.

[97] S. Hosseini, B. Turhan, and M. Mäntylä, "Search based training data selection for cross project defect prediction," in *Proc. 12th Int. Conf. Predictive Models Data Anal. Softw. Eng.*, Sep. 2016, pp. 1–10.

[98] Q. Yu, S. Jiang, and J. Qian, "Which is more important for cross-project defect prediction: Instance or feature?" in *Proc. Int. Conf. Softw. Anal., Test. Evol. (SATE)*, Nov. 2016, pp. 90–95.

[99] K. Kawata, S. Amasaki, and T. Yokogawa, "Improving relevancy filter methods for cross-project defect prediction," in *Proc. 3rd Int. Conf. Appl. Comput. Inf. Technol./2nd Int. Conf. Comput. Sci. Intell.*, Jul. 2015, pp. 2–7.

[100] A. Kaur and K. Kaur, "Value and applicability of academic projects defect datasets in cross-project software defect prediction," in *Proc. 2nd Int. Conf. Comput. Intell. Netw. (CINE)*, Jan. 2016, pp. 154–159.

[101] I. Kaur and N. Kapoor, "Token based approach for cross project prediction of fault prone modules," in *Proc. Int. Conf. Comput. Techn. Inf. Commun. Technol. (ICCTICT)*, Mar. 2016, pp. 215–221.

[102] Z. Li, J. Niu, X.-Y. Jing, W. Yu, and C. Qi, "Cross-project defect prediction via landmark selection-based kernelized discriminant subspace alignment," *IEEE Trans. Rel.*, vol. 70, no. 3, pp. 996–1013, Sep. 2021.

[103] Q. Zou, L. Lu, Z. Yang, and H. Xu, "Multi-source cross project defect prediction with joint Wasserstein distance and ensemble learning," in *Proc. IEEE 32nd Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2021, pp. 57–68.

[104] M. F. Sohan, M. I. Jabiullah, S. S. M. M. Rahman, and S. M. H. Mahmud, "Assessing the effect of imbalanced learning on cross-project software defect prediction," in *Proc. 10th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2019, pp. 1–6.

[105] S. Hosseini and B. Turhan, "A comparison of similarity based instance selection methods for cross project defect prediction," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, Mar. 2021, pp. 1455–1464.

[106] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proc. 38th Int. Conf. Softw. Eng.*, May 2016, pp. 297–308.

[107] Y. Wu, S. Huang, H. Ji, C. Zheng, and C. Bai, "A novel Bayes defect predictor based on information diffusion function," *Knowl.-Based Syst.*, vol. 144, pp. 1–8, Mar. 2018.

[108] J. Sun, X. Jing, and X. Dong, "Manifold learning for cross-project software defect prediction," in *Proc. 5th IEEE Int. Conf. Cloud Comput. Intell. Syst. (CCIS)*, Nov. 2018, pp. 567–571.

[109] Q. Yu, J. Qian, S. Jiang, Z. Wu, and G. Zhang, "An empirical study on the effectiveness of feature selection for cross-project defect prediction," *IEEE Access*, vol. 7, pp. 35710–35718, 2019.

[110] J. Chen, K. Hu, Y. Yu, Z. Chen, Q. Xuan, Y. Liu, and V. Filkov, "Software visualization and deep transfer learning for effective software defect prediction," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 578–589.

[111] J. Tian and Y. Tian, "A model based on program slice and deep learning for software defect prediction," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2020, pp. 1–6.

[112] J. Huang, X. Guan, and S. Li, "Software defect prediction model based on attention mechanism," in *Proc. Int. Conf. Comput. Eng. Appl. (ICCEA)*, Jun. 2021, pp. 338–345.

[113] W. Liu, Y. Zhu, X. Chen, Q. Gu, X. Wang, and S. Gu, "S^2 LMMD: Cross-project software defect prediction via statement semantic learning and maximum mean discrepancy," in *Proc. 28th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2021, pp. 369–379.

[114] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, Jul. 2015, pp. 264–269.

[115] J. Niu, Z. Li, and C. Qi, "Correlation metric selection based correlation alignment for cross-project defect prediction," in *Proc. 20th Int. Conf. Ubiquitous Comput. Commun. (IUCC/CIT/DSCI/SmartCNS)*, Dec. 2021, pp. 490–495.

[116] J. Wu, Y. Wu, N. Niu, and M. Zhou, "MHCPDP: Multi-source heterogeneous cross-project defect prediction via multi-source transfer learning and autoencoder," *Softw. Quality J.*, vol. 29, no. 2, pp. 405–430, Jun. 2021.

[117] I. Abunadi and M. Alenezi, "Towards cross project vulnerability prediction in open source web applications," in *Proc. The Int. Conf. Eng. (MIS-ICEMIS)*, 2015, pp. 1–5.

[118] K. Li, Z. Xiang, T. Chen, S. Wang, and K. C. Tan, "Understanding the automated parameter optimization on transfer learning for cross-project defect prediction: An empirical study," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 566–577.

[119] S. Hosseini and B. Turhan, "An exploratory study of search based training data selection for cross project defect prediction," in *Proc. 44th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2018, pp. 244–251.

[120] S. Qiu, L. Lu, and S. Jiang, "Joint distribution matching model for distribution–adaptation-based cross-project defect prediction," *IET Softw.*, vol. 13, no. 5, pp. 393–402, 2019.

[121] Y. Zhang, D. Lo, X. Xia, and J. Sun, "Combined classifier for cross-project defect prediction: An extended empirical study," *Frontiers Comput. Sci.*, vol. 12, no. 2, pp. 280–296, Apr. 2018.

[122] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'Union fait la force," in *Proc. Softw. Evol. Week-IEEE Conf. Softw. Maintenance, Reeng., Reverse Eng. (CSMR-WCRE)*, Feb. 2014, pp. 164–173.

[123] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "HYDRA: Massively compositional model for cross-project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 42, no. 10, pp. 977–998, Oct. 2016.

[124] C. Ni, X. Xia, D. Lo, X. Chen, and Q. Gu, "Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 48, no. 3, pp. 786–802, Mar. 2022.

[125] F. Wu, X.-Y. Jing, Y. Sun, J. Sun, L. Huang, F. Cui, and Y. Sun, "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Trans. Rel.*, vol. 67, no. 2, pp. 581–597, Jun. 2018.

[126] L. Goel, M. Sharma, S. K. Khatri, and D. Damodaran, "Implementation of data sampling in class imbalance learning for cross project defect prediction: An empirical study," in *Proc. 5th Int. Symp. Innov. Inf. Commun. Technol. (ISIICT)*, Oct. 2018, pp. 1–6.

[127] O. Mizuno and Y. Hirata, "A cross-project evaluation of text-based fault-prone module prediction," in *Proc. 6th Int. Workshop Empirical Softw. Eng. Pract.*, Nov. 2014, pp. 43–48.

[128] G. Holmes, A. Donkin, and I. H. Witten, "WEKA: A machine learning workbench," in *Proc. ANZIIS 94 - Austral. New Zealnd Intell. Inf. Syst. Conf.*, 1994, pp. 357–361.

[129] *Understand 2.0*. Accessed: Mar. 18, 2022. [Online]. Available: http://www.scitools.com/products/understand/

[130] D. Spinellis, "Tool writing: A forgotten art? (Software tools)," *IEEE Softw.*, vol. 22, no. 4, pp. 9–11, Jul./Aug. 2005.

[131] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. 6th Int. Conf. Predictive Models Softw. Eng. (PROMISE)*, 2010, pp. 1–10.

[132] *Mine: Maximal Information-Based Nonparametric Exploration*. Accessed: Mar. 18, 2022. [Online]. Available: http://www.exploredata.net/

[133] E. Kocaguneli, A. Tosun, A. B. Bener, B. Turhan, and B. Caglayan, "Prest: An intelligent software metrics extraction, analysis and defect prediction tool," in *Proc. SEKE*, 2009, pp. 637–642.

[134] *CVX: MATLAB Software for Disciplined Convex Programming*. Accessed: Mar. 18, 2022. [Online]. Available: http://cvxr.com/cvx/

[135] *Javalang*. Accessed: Mar. 18, 2022. [Online]. Available: https://pypi.org/project/javalang/

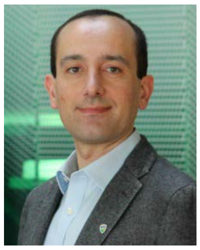
[136] S. Herbold, "CrossPare: A tool for benchmarking cross-project defect predictions," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. Workshop (ASEW)*, Nov. 2015, pp. 90–96.

[137] *Wala*. Accessed: Mar. 18, 2022. [Online]. Available: https://github.com/wala/WALA/

[138] M. Hofmann and R. Klinkenberg, *RapidMiner: Data Mining Use Cases and Business Analytics Applications*. Boca Raton, FL, USA: CRC Press, 2016.

[139] J. Bergstra, D. Yamins, and D. Cox, "Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms," in *Proc. 12th Python Sci. Conf.*, vol. 13, 2013, p. 20.

SOURABH PAL is currently a Junior Researcher with the Cyber-Physical System Laboratory, Innopolis University. Before joining Innopolis University, he was a Researcher at SIEMENS Corporate Research, India. He has over four years of research experience in software engineering. His research interests include empirical software engineering, mining software repository, software quality, application of AI to software engineering, software metrics, and cyber-physical systems.



ALBERTO SILLITTI received the Ph.D. degree in electrical and computer engineering from the University of Genoa, Italy, in 2005. He has been involved in several EU funded projects related to open source software, services architectures, and agile methods in which he applies noninvasive measurement approaches. Currently, he is the Co-Founder, the CEO, and the Chief Scientist at the Centre for Applied Software Engineering, Italy. Previously, he was a Full Professor with the Faculty of Computer Science and Engineering, Innopolis University (Russian Federation) (2016–2022), where he lead the Cyber-Physical Systems Laboratory and the Institute of Information Systems. He also worked as the Associate Dean for consulting activities. He has authored more than 200 papers published in international conferences and journals. His research interests include open source development, agile methods, empirical software engineering, noninvasive measurement, software quality, cyber-physical systems, and mobile and web services, with a focus on mobile and energy-aware software development and quality for cyber-physical systems. He has served as a member for the Program Committee of several international conferences, as a Program Chair for OSS, in 2007 and 2019, XP, in 2010 and 2011, SEDA, in 2012, 2013, and 2014, and the General Chair for SEDA, in 2018.