




Do different cross-project defect prediction methods identify the same defective modules?

Xiang Chen^{1,3,5}  | Yanzhou Mu^{1,2} | Yubin Qu⁴  | Chao Ni³  | Meng Liu¹ | Tong He¹ | Shangqing Liu⁵

¹School of Information Science and Technology, Nantong University, Nantong, China

²College of Intelligence and Computing, Tianjin University, Tianjing, China

³State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

⁴School of Mechanical and Electrical Engineering, Jiangsu College of Engineering and Technology, Nantong, China

⁵School of Computer Science and Engineering, Nanyang Technology University, Singapore

Correspondence

Xiang Chen, School of Information Science and Technology, Nantong University, Nantong, China.

Email: xchencs@ntu.edu.cn

Funding information

Jiangsu Government Scholarship for Overseas Studies; National Natural Science Foundation of China, Grant/Award Number: 61872263, 61702041, 61202006; The Nantong Application Research Plan, Grant/Award Number: JC2018134; The Open Project of State Key Laboratory for Novel Software Technology at Nanjing University, Grant/Award Number: KFKT2019B14

Abstract

Cross-project defect prediction (CPDP) is needed when the target projects are new projects or the projects have less training data, since these projects do not have sufficient historical data to build high-quality prediction models. The researchers have proposed many CPDP methods, and previous studies have conducted extensive comparisons on the performance of different CPDP methods. However, to the best of our knowledge, it remains unclear whether different CPDP methods can identify the same defective modules, and this issue has not been thoroughly explored. In this article, we select 12 state-of-the-art CPDP methods, including eight supervised methods and four unsupervised methods. We first compare the performance of these methods in the same experiment settings on five widely used datasets (ie, NASA, SOFTLAB, PROMISE, AEEEM, and ReLink) and rank these methods via the Scott-Knott test. Final results confirm the competitiveness of unsupervised methods. Then we perform diversity analysis on defective modules for these methods by using the McNemar test. Empirical results verify that different CPDP methods may lead to difference in the modules predicted as defective, especially when the comparison is performed between the supervised methods and unsupervised methods. Finally, we also find there exist a certain number of defective modules, which cannot be correctly identified by any of the CPDP methods or can be correctly identified by only one CPDP method. These findings can be utilized to design more effective methods to further improve the performance of CPDP.

KEYWORDS

cross-project defect prediction, diversity analysis, empirical study, software defect prediction

1 | INTRODUCTION

Software defect prediction (SDP) is a popular and thriving area in the domain of software repository mining. This topic has been widely studied in both research community and industrial community.¹⁻³ SDP can be used to optimize the software testing resource allocation (eg, designing more high-quality test cases or conducting more rigorous code inspection) by identifying potential defective modules in advance. In particular, SDP first mines software repositories (such as version control systems, bug tracking systems, or developer emails) to extract program modules. The granularity of extracted program modules can be set to package, file, method, or even code change as needed. Then it uses manually designed metrics to measure these extracted modules and labels these modules based on the analysis of bug reports and code change logs. These metrics⁴ are designed based on the analysis of code complexity or development process. Finally, SDP models can be constructed based on these gathered SDP datasets.

The majority of existing SDP studies focus on within-project defect prediction (WPDP), which builds the prediction models and predicts defects within the same project. The difficulty of gathering SDP datasets (especially labeling the extracted modules) is the main obstacle of applying SDP to the practical software quality assurance process of enterprises.⁵ Therefore, sometimes new projects do not have enough historical training datasets to build the models. A simple solution is to use the training datasets gathered from other projects. However, the data distribution of different projects may be different, and this heterogeneity poses a great challenge for performing cross-project defect prediction (CPDP)*.

To alleviate the data distribution difference among different projects, the researchers resorted to transfer learning⁶ to solve this issue and many CPDP methods have been proposed.⁷ Previous studies have conducted extensive comparisons for the performance of different CPDP methods. However, to the best of our knowledge, whether different CPDP methods can identify the same defective modules has not been thoroughly investigated.

Our study aims to answer the following research questions.

RQ1: What is the performance difference among different CPDP methods in the same experimental settings?

Experimental settings may be inconsistent in previous empirical studies. For example, different machine learning frameworks (such as Weka, Scikit-learn, and Matlab) may use different default value of hyperparameters in some classifiers. Moreover, previous comparison results for different CPDP methods may not be valid when considering other datasets or other performance measures. Therefore, it is necessary to perform empirical studies based on the same experimental setup and use advanced statistical test method to rank these methods.

To answer this RQ, we select 12 state-of-the-art CPDP methods (ie, eight supervised methods and four unsupervised methods) based on a set of selection criteria. We use implementations of CPDP methods based on Weka framework, and all the classifiers use the default value of hyperparameters to guarantee that all the methods use the same experimental settings. Then, we choose five widely used datasets (ie, NASA, SOFTLAB, PROMISE, AEEEM, and ReLink) as our experimental subjects, use F1 and AUC as performance measures, and use the Scott-Knott test to rank all the CPDP methods. In terms of F1 performance measure, we find that the performance of four unsupervised methods is always in the top 4. While in terms of AUC performance measure, we find that most of the unsupervised methods can achieve the best performance. These results reconfirm the competitiveness of unsupervised methods proposed in previous studies.⁸⁻¹¹ Later, we further investigate the feasibility of state-of-the-art CPDP methods by considering two satisfactory criteria suggested by previous studies.^{12,13} Final results show that the satisfactory ratio of these methods on different datasets is pessimistic. Therefore, CPDP is still a challenge issue. Finally, we also analyze the computational cost of different CPDP methods.

RQ2: Whether different CPDP methods can predict the same defective modules?

In this RQ, the diversity on identifying defective modules for different CPDP methods is measured by using the McNemar test. Based on empirical results, we find that the prediction diversity on defective modules exists when comparing different CPDP methods. It is the most obvious when the comparison is performed between the supervised methods and the unsupervised methods. Moreover, for each pair of CPDP (ie, the source project and the target project are determined), we analyze the number of defective modules, which cannot be detected by any of the chosen CPDP methods. Then we also analyze the number of defective modules, which can be detected by only one CPDP method. Final results show that the number of these defective modules cannot be ignored. This finding can be utilized to design more effective CPDP methods in the future. One possible solution is to use stacking ensemble.

The main contributions of this article can be summarized as follows:

- To the best of our knowledge, we are the first to rank state-of-the-art CPDP methods (ie, eight supervised methods and four unsupervised methods) under the same experiment settings in terms of commonly used performance measures (ie, F1 and AUC) on five widely used SDP datasets. Moreover, we also investigate the feasibility of these state-of-the-art methods based on two satisfactory criteria.^{12,13}
- We are the first to conduct a large-scale empirical study to perform diversity analysis on identifying defective modules by using the McNemar test for different CPDP methods. Moreover, we also analyze the number of defective modules, which cannot be correctly identified by any of the CPDP methods or can be correctly identified by one CPDP method.

The rest of this article is organized as follows. Section 2 introduces the background of SDP and related work for cross-project defect prediction. Section 3 introduces the methodology of our study, including research questions, motivation of our empirical studies, empirical subjects, and performance measures. Section 4 performs result analysis. Section 5 discusses potential threats to validity for our empirical study. Section 6 concludes this article and points out some potential future work.

2 | BACKGROUND AND RELATED WORK

In this section, we first introduce the background of SDP. Then we summarize the related work for cross-project defect prediction.

*If the source project and the target project come from different companies, this issue is also called cross-company defect prediction.

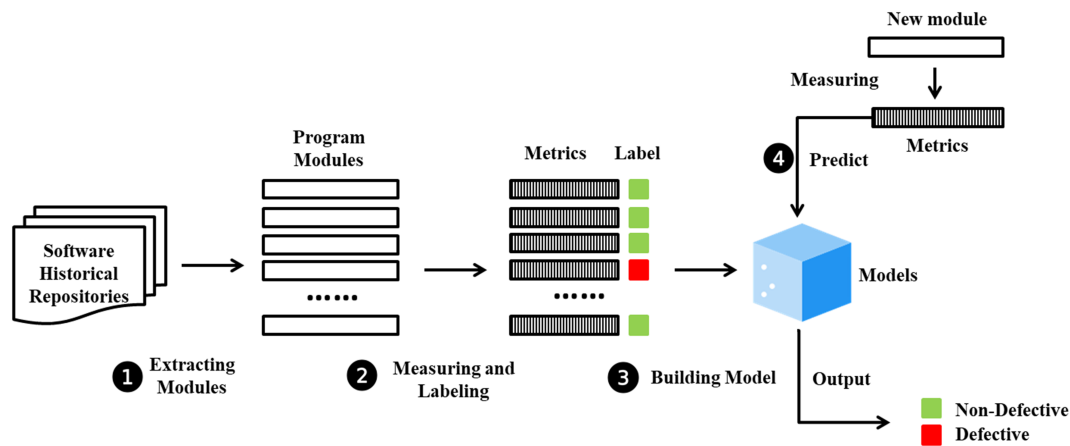


FIGURE 1 The process of software defect prediction when the prediction target is set to defect proneness

2.1 | Background of SDP

Software defects may be introduced in the source code in different phases of software development process. The root cause of defects may come from misunderstanding the clients' requirements, uncontrollable development process, or the lack of developers' experience in programming languages or domain knowledge. Hidden defects in the projects will produce unexpected results and even result in huge economic loss for enterprises in the worst cases after these projects are deployed.

Since available resources allocated for software testing and code inspection are usually limited, effective methods are required by software quality assurance groups to identify potential defective software modules as early as possible, which help to optimize the allocation of these testing resources. SDP is one of such feasible methods. The prediction target of SDP can be set to defect proneness, or defect number/density for the new program modules as needed. When the target is set to defect proneness, the process of SDP can be summarized in Figure 1[†]. Firstly, software modules are automatically extracted from software historical repositories, such as version control systems (eg, SVN, CVS, and GIT) and bug tracking systems (eg, Bugzilla and Jira). The version control systems contain the source codes and commit messages, while the bug tracking systems include bug reports. Secondly, software metrics (ie, features) are designed and then used to measure extracted program modules and the type (ie, defective or nondefective) of the modules are labeled by analyzing bug reports and commit messages. Metrics are often designed based on the analysis of the code complexity, the characteristics of development process, or the developers' experience. Thirdly, SDP models are trained based on the gathered SDP datasets, which are later used to predict defect proneness (ie, defective or nondefective) of new software modules.

2.2 | Related work for cross-project defect prediction

Most of previous studies focused on within-project defect prediction (WPDP), which builds the prediction models and predicts defects within the same project. In real software development, projects needing defect prediction may be new projects or have less training data in most cases. That means these projects do not have sufficient historical data to build high-quality prediction models. A simple solution is to directly use training data gathered from other projects to construct the models. However, application domain, utilized programming languages, development process, and developer experience of different projects may not be the same. This will result in the large data distribution difference in most cases.

The researchers conducted a set of empirical studies to investigate the feasibility of CPDP based on real-world software projects. Briand et al¹⁴ firstly found CPDP can achieve satisfactory performance on two medium-sized Java projects developed by Oracle corporation. Then, Zimmermann et al¹² conducted more large-scale empirical studies. They analyzed 12 real-world projects from open-source communities and Microsoft corporation. After running 622 cross-project predictions, they found only 3.4% (ie, 21 cross-project predictions) can achieve satisfactory performance. Later, He et al¹³ analyzed 10 open-source projects. After running 160 586 cross-project predictions, they found only 0.32% to 4.67% can achieve satisfactory performance based on different classifiers. Jureczko and Madeyski¹⁵ analyzed 38 proprietary, open-source, and academic projects. They found the results of CPDP are still unsatisfactory. Moreover, Kamei et al^{16,17} confirmed the unsatisfactory performance of CPDP in the context of just-in-time defect prediction.¹⁸ Rahman et al¹⁹ investigated the feasibility of CPDP in terms of effort-aware performance measures (ie, taking into consideration the limitation of available testing resources) and surprisingly found that the performance of CPDP is no worse than WPDP and is substantially better than the random method. However, based on the empirical results in most of the above studies, the performance of CPDP is still far from satisfactory, and this issue can be considered as a complicated and challenge problem. Turhan²⁰ analyzed the reasons for poor performance of CPDP through dataset shift concept. He classified different forms of dataset shift into six categories, such as covariate shift,

[†]In Figure 1, we use a red color to denote the defective modules and use a green color to denote the nondefective modules.

prior probability shift, sample selection bias, imbalanced data, domain shift, and source component shift. His analysis forms the theoretical support for the follow-up CPDP studies.

Until now, the researchers have proposed many CPDP methods to reduce data distribution between the source project and the target project. In this subsection, we simply classify existing methods into two categories: supervised CPDP methods and unsupervised CPDP methods. Recently Hosseini et al⁷ performed a systematic literature review and meta-analysis on previous CPDP studies between 2002 and 2015. Moreover, Herbold et al²¹ provided a benchmark platform for CPDP methods proposed between 2008 and 2015.

2.2.1 | Supervised CPDP methods

Most studies fall into this category. These studies can be further categorized into three subcategories. In the first subcategory, the researchers assumed that the source project and the target project utilize the same metrics to measure the extracted modules. Moreover, they only use the data in the source project to construct the models. (a) Some researchers focus on metric value transformation. The objective of metric value transformation is to make the transformed values of the metric satisfy the normal distribution assumption. Commonly used transformation methods include logarithmic transformation, min-max normalization, z score normalization, rank transformation, or Box-Cox transformation.²²⁻²⁴ (b) Some researchers focus on selecting source projects, which are more similar to the target project. For example, He et al¹³ used distribution characteristics of metrics to choose suitable projects from candidate source projects. Liu et al²⁵ proposed a source project estimator (SPE), which can select the projects with highest distribution similarity to the target project from the candidate source projects. Krishna et al^{26,27} considered the usage of bellwether. Given a set of candidate source projects, the bellwether is the project, which can achieve better performance than all the others. (c) Some researchers focus on instance selection and weight setting for the modules in the source project. For example, Menzies et al,^{28,29} Bettenburg et al,^{30,31} and Herbold et al³² considered local models, which cluster the available training data into homogeneous regions and trained classifiers for each region. Turhan et al⁵ proposed the Burak filter, which selects most similar instances from the source project via *k*-nearest neighbor. Peters et al³³ proposed the Peters filter by analyzing the structure of the target project. Herbold³⁴ presented two strategies (ie, EM clustering and *k*-nearest neighbor) to perform instance selection. Hosseini et al^{35,36} considered genetic algorithm to perform instance selection. Bin et al³⁷ considered more strategies for instance selection and surprisingly found there is no need to perform instance selection. Xu et al³⁸ leveraged dissimilarity-based sparse subset selection method. Ma et al³⁹ proposed the transfer naive Bayes (TNB) method, which can assign weights for the instances in the source project. Poon et al⁴⁰ proposed a credibility theory-based naive Bayes method to establish a novel reweighting mechanism between the source project and the target project. (d) Some researchers focus on feature mapping and selection.⁴¹⁻⁴³ For example, Nam et al²³ applied a state-of-the-art transfer learning method (ie, TCA⁴⁴) to make feature distribution similar and then proposed TCA+, which can automatically select a suitable normalization method. He et al⁴⁵ investigated the feasibility of the CPDP models constructed by a simplified feature subset. Ni et al^{46,47} proposed a cluster-based feature selection method FeSCH, which is a two-phase method. The feature clustering phase clusters features via a density-based clustering method and the feature selection phase selects features from each cluster by a ranking strategy. (e) Some researchers resort to advanced machine learning methods. For example, Panichella et al⁴⁸ proposed a combined method CODEP based on ensemble learning. Later, Zhang et al⁴⁹ considered other seven ensemble learning-based methods. Since most of defects exist in a few modules of projects under testing, there exists the class imbalanced problem in most of the gathered datasets. Ryu et al⁵⁰⁻⁵² proposed CPDP methods by considering the class imbalanced problem. Limsettho et al⁵³ proposed a method CDE-SMOTE by using class distribution estimation and synthetic minority oversampling technique (SMOTE). Jing et al⁵⁴ considered subclass discriminant analysis (SDA) method. Wang et al⁵⁵ resorted to deep learning. They utilized deep belief network (DBN) to automatically learn semantic features from token vectors extracted from abstract syntax trees (ASTs) of program modules. Canfora et al^{56,57} and Ryu et al⁵⁸ used multiobjective optimization-based methods to construct CPDP models. Wang et al⁵⁹ proposed a top-*k* learning to rank method, which can help to assign higher rank to modules with highest severity. Yang and Wen⁶⁰ utilized ridge and lasso regression models for cross-version defect prediction.

In the second subcategory, the researchers found that different researchers may use different metrics to measure the extracted modules, and this results in the hypothesis of the methods in the first subcategory invalid. This problem is more challenge and is called heterogeneous defect prediction. Nam et al^{61,62} proposed the HDP method to perform defect prediction across projects with heterogeneous metric sets. This method includes the metric selection phase and metric matching phase. Then Yu et al⁶³ presented a feature matching method to convert the heterogeneous features into the matched features and presented a feature transfer method to transfer the matched features from the source project to the target project. Jing et al⁶⁴ proposed unified metric representation (UMR) for the data of the source project and the target project, then they used canonical correlation analysis (CCA) to make the data distribution similar. Li et al⁶⁵ proposed a new cost-sensitive transfer kernel canonical correlation analysis method. This method can not only make the data distribution more similar but also utilize the different misclassification costs for defective and nondefective modules to alleviate the class imbalanced problem. Meanwhile, they⁶⁶ proposed a novel ensemble multiple kernel correlation alignment (EMKCA)-based method.

In the third subcategory, some researchers assumed that there are a few additional labeled program modules in the target project, and these modules can be utilized to construct models. Turhan et al⁶⁷ investigated the feasibility of using mixed project data (ie, a few labeled modules in the target project and labeled modules in the source project). Ryu et al⁵² proposed a transfer cost-sensitive boosting method, which considers both

knowledge transfer and class imbalanced learning for CPDP when given a small number of labeled modules in the target project. Xia et al⁶⁸ proposed a hybrid model construction approach HYDRA, which includes the genetic algorithm phase and the ensemble learning phase. Chen et al⁶⁹ proposed the DTB method. This method first used a data gravitation method to reshape the whole distribution of data in the source project to fit the data in the target project. Then it used the transfer boosting method to utilize a few labeled modules in the target project to eliminate negative instances in the source project. Zhang et al⁷⁰ utilized graph-based semisupervised learning method. Wu et al⁷¹ utilized a semisupervised dictionary learning method and proposed a cost-sensitive kernelized semisupervised dictionary learning method.

2.2.2 | Unsupervised CPDP methods

These methods attempted to perform defect prediction on the modules of the target project immediately. The assumption of these methods is that the metric values of defective modules have the tendency to be higher than the metric values of non-defective modules. Nam and Kim⁸ proposed CLA and CLAMI methods. The key idea of these two methods was to label an unlabeled dataset by using the magnitude of metric values. Zhang et al⁹ proposed a connectivity-based unsupervised method via spectral clustering. Recently, Zhou et al¹⁰ found that simple unsupervised methods (ie, ManualDown and ManualUp) seem to have better prediction performance than complex CPDP methods. Their findings indicated that previous studies on CPDP perhaps make a simple problem complex.

Except for the above studies, some researchers studied CPDP in view of privacy preservation. The privacy issue is an obstacle for the project owners to share their gathered datasets. Peters and Menzies⁷² explored the MORPH method, which is a data mutator and is used to move the data a random distance while taking care not to cross class boundaries. Then Peters et al⁷³ considered the CLIFF method, which is an instance pruner and is used to remove irrelevant instances. Later, they⁷⁴ proposed LACE2, which reduces the amount of shared data by using multiparty data sharing. Fan et al⁷⁵ found that a simple unsupervised method ManualDown should be used as a baseline method for comparison when new privacy-preserving data-sharing algorithms are proposed. We⁷⁶ recently applied differential privacy (DP) to privacy-preserving SDP model sharing. Some researchers investigated the cross-version defect prediction scenario. Amasaki et al⁷⁷ analyzed the applicability of cross-project defect prediction method for multiversion projects. Then they⁷⁸ analyzed whether cross-project defect prediction methods can be used for cross-version defect prediction. Bennin et al⁷⁹ investigated the performance of cross-version defect prediction in terms of effort-aware performance measures.

In previous studies, some researchers performed diversity analysis for within-project defect prediction. Bowes et al⁸⁰ compared the performance of four classifiers (ie, random forest, naive Bayes, RPart, and SVM) in WPDP. They found these classifiers can achieve similar performance in terms of F1 and MCC measures. However, they found there exists difference between these classifiers in terms of the specific defective modules each detects and does not detect. Some researchers performed diversity analysis for cross-project defect prediction.^{24,48} However, they only performed diversity analysis in view of metric transformation methods or classifiers. For example, since different software metrics rarely follow a normal distribution, Zhang et al²⁴ investigated whether different transformation methods (ie, log transformation, rank transformation, and Box-Cox transformation) cause distinct predictions on the same defective module. While Panichella et al⁴⁸ compared six different classifiers (ie, logistic regression, RBFNetwork, ADTree, decision table, multilayer perception, and Bayes Network) in CPDP and draw the same conclusion in Zhang et al.²⁴ However, to the best of our knowledge, no researchers have investigated whether different state-of-the-art CPDP methods can find the same defective modules. Therefore, we consider eight supervised CPDP methods and four unsupervised CPDP methods, which have not been investigated in previous studies^{24,48} and then perform diversity analysis on five widely used SDP datasets.

3 | METHODOLOGY

In this section, we first show the motivation of our empirical study by a case study. Then we introduce the selection criteria in our empirical studies for choosing CPDP methods. Based on these criteria, eight supervised methods and four unsupervised methods are finally chosen. Later, we show the details of these chosen CPDP methods and their experimental setup. Moreover, we introduce empirical subjects and performance measures. Finally, we give the details of the diversity analysis method used in our study.

3.1 | Motivation study

In this motivation example, we choose two projects (ie, PDE and JDT) from AEEEM dataset (the introduction of AEEEM dataset can be found in Section 3.3). Then we choose two CPDP methods: Watanabe-PROMISE08⁸¹ (ie, SM3 method) and Ma-IST12³⁹ (ie, SM5 method). The details of these two CPDP methods can be found in Section 3.2. For this case study, we use Venn diagram to show the prediction diversity on defective modules for these two CPDP methods and final results can be found in Figure 2. In Figure 2A, the project PDE is set to the target project and the project JDT is set to the source project. There are 209 defective modules in the project PDE. For these defective modules, only 29 modules can be correctly predicted by both SM3 method and SM5 methods simultaneously. Fifty-five modules can be only correctly predicted by SM3 method, and 13 modules can be only correctly predicted by SM5 method. Moreover, we also find there exist 112 defective modules, which cannot

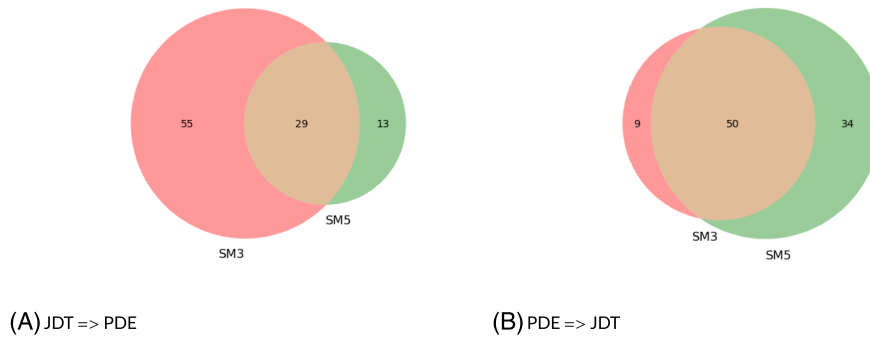


FIGURE 2 A motivation example to show that the prediction diversity on defective modules exists for different cross-project defect prediction (CPDP) methods

be correctly predicted by either of these two methods. In Figure 2B, the project JDT is set to the target project, and the project PDE is set to the source project. We can also find that the prediction diversity on defective modules still exists for these two CPDP methods.

This motivation study shows prediction diversity on defective modules exists in this case study. However, whether this phenomenon exists when considering other projects or other CPDP methods is still unknown. Therefore, in our empirical study, we will investigate whether this phenomenon commonly exists by considering more state-of-the-art CPDP methods and more datasets widely used in previous CPDP studies.

3.2 | CPDP methods

As analyzed in related work, CPDP is an active research issue for SDP, and many CPDP methods have been proposed. In the rest of this subsection, we first introduce the selection criteria for CPDP methods[‡]. Then we summarize the details of our selected CPDP methods, including the method abbreviation, brief description, and experimental setup in our empirical studies.

3.2.1 | Supervised CPDP methods

For supervised CPDP methods, we propose the following four selection criteria in our empirical studies.

Criterion 1: These CPDP methods should achieve better performance in a recent benchmark study.²¹

Criterion 2: These CPDP methods should be often set to baselines in previous CPDP methods.

Criterion 3: These CPDP methods should not need to use any labeled modules in the target project to construct models.

Criterion 4: These CPDP methods should assume that the source project and the target project use the same metrics to measure the extracted program modules.

The first two criteria can help to choose classical and high-performance CPDP methods. The last two criteria can guarantee that a fair comparison is made between different CPDP methods, since considering these two criteria can make the CPDP methods use the same training data and the same testing data for model performance evaluation. Based on criterion 1, we choose four methods,^{5,22,29,81} since Herbold et al²¹ found these methods can achieve better performance in their developed CrossPare platform. Based on criterion 2, we choose four additional methods,^{23,39,45,48} since these methods are often used as baseline methods. Based on criterion 3, we do not select CPDP methods in the third subcategory of the supervised methods, such as in previous studies.^{52,68-71} Based on criterion 4, we do not select CPDP methods in the second subcategory of the supervised methods, such as in previous studies.^{61-64,66}

Therefore, we choose eight supervised CPDP methods in total. Then, we briefly introduce the details of these chosen CPDP methods and experimental settings in our empirical study. More details of these methods can be found in the corresponding papers.

CamargoCruz-ESEM09 (SM1)

This method was proposed by Camargo Cruz and Ochimizu.²² They standardized the data in the source project and the target project based on the logarithm and the median value of the metric. This data preprocessing step can not only improve symmetry and normality of the metric but also reduce the number of outliers. Then this method used a decision tree classifier to construct prediction models.

Turhan-EMSE09 (SM2)

This method was proposed by Turhan et al.⁵ This method was also called Burak filter. The rationale behind this method was that similar modules in the source projects can provide better prediction performance for a given target project. Therefore, they considered k -nearest neighbour algorithm to select instances in the source projects based on the analysis for the modules in the target project. In this method, Euclidean distance was used to

[‡]Notice we only consider CPDP methods, which have been published before January 2019.

measure the similarity between different program modules and the value of k was set to 10. Based on the selected modules in the source project, the decision tree classifier was used to construct the prediction models.

Watanabe-PROMISE08 (SM3)

This method was proposed by Watanabe et al.⁸¹ This method aimed to compensate the difference between the source project and the target project by a standardization technique that rescales the data. Then this method used a decision tree classifier to construct prediction models.

He-IST15 (SM4)

This method was proposed by He et al.⁴⁵ They proposed an easy-to-use approach to simplify the set of software metrics based on filter-based methods for feature selection, which could help developers to build suitable prediction models with the most representative metrics according to their specific requirements. In particular, this method only used the best k metrics. To determine the best metrics, they chose the optimal value for k based on the overlap with metrics determined by correlation-based feature subset (CFS). Moreover, if the optimal value for k was larger than 30, k is set to 30 to limit the size of the power set, which is used to determine the optimal feature subset. Then this method used random forest classifier to construct prediction models.

Ma-IST12 (SM5)

This method was proposed by Ma et al.³⁹ They proposed a novel algorithm called TNB by using the information of all the proper features in the training data. Their solution estimated the distribution of the test data and transferred cross-project data information into the weights of the training data. On these weighted data, the defect prediction models were built by using decision tree classifier.

Nam-ICSE13 (SM6)

This method was proposed by Nam et al.²³ They first provided a set of rules for selecting an appropriate normalization method (such as no normalization, min-max normalization, z score standardization) based on a given pair of the source project and the target project. Then they applied a state-of-the-art transfer learning approach TCA⁴⁴ on the normalized data to make feature distributions more similar in the source and target projects. Finally, this method used naive Bayes classifier to construct prediction models.

Panichella-SANER14 (SM7)

This method was proposed by Panichella et al.⁴⁸ They considered six different classifiers (ie, logistic regression, Bayes network, radial basis function network, multilayer perceptron, alternating decision trees, and decision table) and found that different classifiers may identify different defective modules. Therefore, they proposed a combined method Combined DEfect Predictor (CODEP). In particular, this method applied six classifiers on the data in the source project to obtain the defect-proneness for each program module. Then they used the output of each classifier in the previous step as new metrics and constructed final models by using logistic regression classifier.

Menzies-TSE13 (SM8)

This method was proposed by Menzies et al.²⁹ They created a local model via clustering of the data in the source project with the WHERE algorithm and then used random forest classifier to perform prediction on the target project.

3.2.2 | Unsupervised CPDP methods

For unsupervised CPDP methods, we consider four methods in total. These methods are the latest proposed methods and published in the top-tier journal or conference in the software engineering domain, such as ICSE, ASE, and TOSEM. These unsupervised methods do not need any labeled data from the source projects to construct the models. Therefore, it can effectively avoid the challenges caused by the value distribution difference between the source project and the target project. Moreover, the implementation of some unsupervised methods is simple and the computational cost is low.

Nam-ASE15 (UM1 and UM2)

Nam and Kim⁸ presented the CLA method and CLAMI method. The key idea of these two unsupervised methods was to label an unlabelled dataset by using the magnitude of metric values. Hence, CLA and CLAMI have the advantages of automated manner and no manual labeling effort required. The first two phases of these methods were (a) clustering unlabeled program modules and (b) labeling these modules in clusters. CLA only consisted of these two phases. CLAMI had two additional phases to generate the training dataset: (c) metric selection and (d) instance selection, since these two phases can be used to further improve the dataset quality. Then CLAMI used logistic regression classifier to construct models. In this article, we use UM1 to denote CLA and UM2 to denote CLAMI.

TABLE 1 The statistics of AEEEM dataset

Name	# Modules	# (%) Defective Modules
EQ (Equinox)	324	129 (39.8)
JDT (Eclipse JDT Core)	997	206 (20.7)
LC (Apache Lucene)	691	64 (9.3)
ML (Mylyn)	1862	245 (13.2)
PDE (Eclipse PDE UI)	1497	209 (14.0)

Zhang-ICSE16 (UM3)

Zhang et al⁹ presented to leverage spectral clustering (SC) to tackle the heterogeneity between the source project and the target project for cross-project defect prediction. They first used z score to normalize each metric, Then they used three steps for spectral clustering: (a) the first step was to calculate the Laplacian matrix L_{sym} , (b) the second step was to perform the eigendecomposition on the matrix L_{sym} , and (c) the third step was to divide all the program modules into two clusters (ie, the defective cluster and the non-defective cluster). To identify the defective cluster, they used the following heuristic: For most of the metrics, defective software modules generally had larger values than nondefective software modules. Based on this heuristic, they used the average row sums of the normalized metrics of each cluster. The cluster with larger average row sum was classified as the defective cluster, and another cluster was classified as the nondefective cluster.

Zhou-TOSEM18 (UM4)

Zhou et al¹⁰ took into consideration the metric lines of code (LOC) when building the defect prediction models. A common sense in previous empirical studies was that modules with larger LOC have more tendency to have defects, which means that software modules with larger LOC are more likely to be defective. Therefore, ManualDown method proposed by them sorted program modules in the descending order according to the value of LOC. ⁵In their empirical studies, they classified the top 50% modules as defective and the remaining 50% as nondefective.

3.3 | Experimental subjects

In our empirical study, we choose five publicly available datasets (ie, AEEEM, ReLink, PROMISE, NASA, and SOFTLAB). These five datasets have been widely used in previous CPDP studies and have high quality. The diversity of these datasets (eg, different metrics, different software domains, and different granularity of modules) can guarantee the generalization of our empirical results. In the rest of this subsection, we introduce these datasets in sequence.

3.3.1 | AEEEM database

This dataset was gathered by D'Ambros et al.⁸² The dataset includes five projects (ie, EQ, JDT, LC, ML, and PDE). The granularity of program modules is set to file. Sixty-one metrics are used to measure the extracted modules. In particular, 17 metrics are based on the source code complexity, five metrics are based on the previous-defect information, five metrics are based on the entropy of code change, 17 metrics are based on the entropy of source code, and 17 metrics are based on the churn of source code. The statistics of AEEEM dataset can be found in Table 1. This table lists the name of the project, the number of program modules, and the number (percentage) of defective modules.

3.3.2 | ReLink dataset

This dataset was gathered by Wu et al⁸³ and the label information was manually verified and corrected. The granularity of program modules is set to file. Twenty-six metrics are used to measure the extracted program modules by using Understand tool.[¶]This dataset includes three datasets (ie, Apache, Safe, and ZXing) and the statistics of ReLink dataset can be found in Table 2.

⁸For five datasets in our empirical studies, which will be introduced in Section 3.3, we choose the following metrics as the metric LOC according to suggestions by Zhou et al¹⁰: LOC_EXECUTABLE (The number of lines of executable code for a module) metric in the NASA dataset, executable_loc (The number of lines of executable code for a module) metric in the SOFTLAB dataset, ck_oo_numberOfLinesOfCode (Number of Lines of code) metric in the AEEEM dataset, CountLineCode (Number of lines of code) metric in the ReLink dataset or loc (Number of lines of code) metric in the PROMISE dataset.

[¶]<https://scitools.com/>

TABLE 2 The statistics of ReLink dataset

Name	# Modules	# (%) Defective Modules
Apache (Apache HTTP Server)	194	98 (50.52)
Safe (OpenIntents Safe)	56	22 (39.29)
ZXing	399	118 (29.57)

TABLE 3 The statistics of PROMISE dataset

Name	# Modules	# (%) Defective Modules
Ant-1.7	745	166 (22)
Camel-1.6	965	188 (19)
Ivy-2.0	352	40 (11)
Jedit-4.0	306	75 (25)
Log4j-1.2	205	189 (92)
Lucene-2.4	340	203 (60)
Poi-3.0	442	281 (64)
Prop-6.0	661	66 (10)
Tomcat-6.0	858	77 (9)
Xalan-2.7	910	898 (99)

3.3.3 | PROMISE dataset

This dataset was gathered by Jureczko and Madeyski¹⁵ with the help of two tools (ie, BugInfo and CKJM). The granularity of program modules is set to class. Twenty metrics based on the code complexity are used to measure the extracted program modules. More details of these metrics can be found in Jureczko and Madeyski.¹⁵ The statistics of PROMISE dataset can be found in Table 3.

3.3.4 | NASA dataset

This dataset was original gathered by Menzies et al.⁸⁴ and then was further cleaned by Shepperd et al.⁸⁵ The granularity of program modules is set to function/method. Each project in NASA represents a NASA software system or subsystem, which contains the corresponding defect-marking data and various static code metrics, which include size, readability, and complexity features. The statistics of NASA dataset can be found in Table 4.

TABLE 4 The statistics of NASA dataset

Name	# Modules	# (%) Defective Modules
CM1	344	42 (12.21)
JM1	9593	1759 (18.34)
KC1	2096	325 (15.51)
KC3	200	36 (18.00)
MC1	9277	68 (0.73)
MC2	127	44 (34.65)
MW1	264	27 (10.23)
PC1	759	61 (8.04)
PC2	1585	16 (1.01)
PC3	1125	140 (12.44)
PC4	1399	178 (12.72)
PC5	17 001	503 (2.96)

3.3.5 | SOFTLAB dataset

This dataset was gathered by Turhan et al.⁵ It consists of five projects, which are embedded controller software designed for white goods. The extracted modules are measured by 29 metrics. This dataset includes six projects (ie, from AR1 to AR6) and the statistics of SOFTLAB dataset can be found in Table 5.

There are five, three, 10, 12, and five projects in AEEEM, ReLink, PROMISE, NASA and SOFTLAB datasets, respectively. Therefore, the total number of possible pairs for CPDP in these five datasets are 20 ($=5 \times 4$), 6 ($=3 \times 2$), 90 ($=10 \times 9$), 132 ($=12 \times 11$), and 20 ($=5 \times 4$), respectively.

Finally, we use Table 6 to summarize the usage statistics of these five experimental subjects in previous CPDP studies. This table includes the dataset name, the number of accumulated usage times (ie, # usage), the related studies, and the time of the first usage and its corresponding study. Based on this table, we can find that these experimental subjects have been widely used in previous studies. It is not hard to find that these experimental subjects have been used at least 10 times in previous CPDP studies. In these five experimental subjects, PROMISE has been most frequently used and the number of accumulated usage times is up to 45 times.

3.4 | Performance measures

In SDP, if we treat defective modules as positive instances and nondefective modules as negative instances, we can classify program modules into four types according to the actual type and the predicted type of these modules: true positive, false positive, true negative, and false negative. We use *TP*, *FP*, *TN*, and *FN* to denote the number of true positives, false positives, true negatives, and false negatives, respectively. The confusion matrix in the context of SDP can be summarized in Table 7, and it forms the fundamental basis for computing threshold-dependent performance measures. For this type of measures, if the predicted defective probability for a new module is larger than the given threshold (0.5 is set in most cases), this module will be classified as the defective module, otherwise, it will be classified as the nondefective module. In this article, we introduce three threshold-dependent performance measures: precision, recall, and F1.

The performance measure precision returns the ratio of the number of defective modules, which are correctly classified as defective to the number of modules, which are classified as defective. This performance measure can be defined as follows:

$$precision = \frac{TP}{TP + FP} \quad (1)$$

The performance measure recall returns the ratio of the number of defective modules, which are correctly classified as defective to the total number of defective modules. This measure can be defined as follows:

TABLE 5 The statistics of SOFTLAB dataset

Name	# Modules	# (%) Defective Modules
AR1	121	9 (7.44)
AR3	63	8 (12.7)
AR4	107	20 (18.69)
AR5	36	8 (22.22)
AR6	101	15 (14.85)

TABLE 6 The usage statistics of experimental subjects

Name	# Usage	Related Studies	Time of First Usage
AEEEM	16	9,10,23,24,26,27,32,46,47,54,61,62,64-66,71	2013 ²³
ReLink	14	8,10,23,24,26,27,47,54,61,62,64-66,71	2013 ²³
PROMISE	45	13,15,24-34 35-38,40,45,48,49,52,55-59 9,10,60-63,65,66,68,69,71-74 75,77,78	2010 ¹⁵
NASA	18	5,26,32,39,50,51,54,61-67 9,10,70,71	2009 ⁵
SOFTLAB	11	5,10,39,50,54,61,62,64-67	2009 ⁵

TABLE 7 Confusion matrix based on the actual type and the predicted type of program modules

Actual Type	Predicted Type Defective Modules	Nondefective Modules
Defective modules	TP	FN
Nondefective modules	FP	TN

$$recall = \frac{TP}{TP + FN}. \quad (2)$$

Since there exists trade-off between precision and recall in practice, higher precision means lower recall and vice versa in most cases. Therefore, it is hard to achieve both high precision and high recall at the same time. Here, we use the performance measure F1, which is the harmonic mean between precision and recall, to evaluate the overall performance of the trained CPDP models. This measure can be defined as follows:

$$F1 = \frac{2 \times precision \times recall}{precision + recall}. \quad (3)$$

However, some of previous studies^{55,86} argued that these threshold-dependent performance measures are problematic. For example, these measures depend on an arbitrarily selected threshold, and these measures are sensitive to class imbalanced problem existed in most of the gathered SDP datasets. Recently, the researchers are more inclined to use the area under the receiver operator characteristic curve (AUC) to measure the discrimination power of the constructed models. AUC is computed by computing the area under the curve, which plots the true positive rate (TPR) against the false positive rate (FPR), while varying the threshold that is used to determine whether a program module is classified as defective or nondefective. The value of AUC ranges between 0 (ie, the worst performance) and 1 (ie, the best performance). The higher the AUC value, the better the prediction performance of the constructed models. Notice a value of 0.5 indicates the performance similar to the random method.

3.5 | Diversity analysis method

To evaluate whether different CPDP methods result in distinct predictions on defective modules, we test the following null hypothesis:

Ho: Both CPDP methods M1 and M2 can identify similar defective modules in the target project.

We use the McNemar test⁸⁷ with the 95% confidence level to perform diversity analysis on defective modules between different CPDP methods. This diversity analysis method has also been used by Zhang et al²⁴ in their previous study. The McNemar test is a nonparametric test, and it does not need the assumptions on the distribution of a subject variable. Notice all the CPDP models are constructed on the same source project, and then applied to the same target project; therefore, the McNemar test is applicable to our study.

To perform the McNemar test, we need to construct a contingency matrix based on the prediction results by two CPDP methods (ie, M1 and M2), which is shown in Table 8. In this contingency matrix, N_{cc} denotes the number of defective modules that both CPDP methods achieve correct predictions. N_{cw} denotes the number of defective modules that the CPDP method M1 achieves a correct prediction while the CPDP method M2 achieves a wrong prediction. N_{wc} denotes the number of defective modules that the CPDP method M2 achieves a correct prediction while the method CPDP M1 achieves a wrong prediction. Finally, N_{ww} denotes the number of defective modules that both CPDP methods have wrong predictions.

In our empirical studies, we use *mcnemar.exact* function provided by R package *exact2x2*[#] to perform the McNemar test. If *P* value is smaller than .05, we will reject the null hypothesis *Ho* (ie, the defective modules identified by these two methods M1 and M2 are almost the same). We use an artificially constructed illustrative example in Table 9 to show the rationality of this diversity analysis method. In this table, there are 10 defective modules (ie, {m0,m1,...,m9}). The prediction results of three different methods (ie, Method 1, Method 2, and Method 3) can be found in the last three columns. Here, 1 means that this module is predicted as the defective module and 0 means that this module is predicted as the nondefective module by the corresponding method. Based on the McNemar test, the *P* value of Method 1 vs Method 2 is .03, and this means these two methods can almost identify distinct defective modules, while the *P* value of Method 1 vs Method 3 is .56, and this means these two methods almost identify the same defective modules. Moreover, we show the results of the McNemar test for the motivation examples introduced in Figure 2. When comparing chosen two CPDP methods SM3 and SM5, the *P* value of these two methods is 3.52E-07 and 1.38E-04 in two CPDP cases (ie, JDT ≥ PDE and PDE ≥ JDT), respectively. This shows these two CPDP methods cannot identify similar defective modules in the target project in these two cases.

[#]<https://cran.r-project.org/web/packages/exact2x2/index.html>

TABLE 8 A contingency matrix based on the prediction results by two CPDP methods M1 and M2

M1 vs M2	Correct Predictions	Wrong Predictions
Correct predictions	N_{cc}	N_{cw}
Wrong predictions	N_{wc}	N_{ww}

TABLE 9 An illustrative example to show the rationality of the diversity analysis method

Defective Module	Method 1	Method 2	Method 3
m_0	1	0	1
m_1	1	0	1
m_2	1	0	1
m_3	1	0	1
m_4	1	0	1
m_5	1	0	1
m_6	1	0	1
m_7	1	1	0
m_8	0	1	1
m_9	0	0	1

4 | RESULT ANALYSIS

4.1 | Result analysis for RQ1

RQ1: What is the performance difference among different CPDP methods in the same experimental settings?

4.1.1 | Motivation

There are some issues for comparing different CPDP methods in previous studies. First, some studies directly used the results from the original papers.¹⁰ However, the default value of hyperparameters for the same classifier when using different machine learning framework may not be the same, and this is the potential threat to previous empirical studies. For example, the number of decision trees in random forest classifier ranges from 50 for Matlab, 100 for Weka, to 500 for random-forest *R* package.⁸⁶ Second, different studies may use different datasets or performance measures in their empirical studies. Therefore, the previous empirical results may not be valid when considering other datasets or other performance measures. Third, if there exist many candidate source projects, different researchers may use different processing methods. For example, some researchers combine all the data in the candidate projects into a single datasets.⁵ Some researchers treat each source project individually.²³ Therefore, it is necessary to perform a comparative study for performance comparison among state-of-the-art CPDP methods in the same experimental settings.

4.1.2 | Approach

In our studies, the experimental settings are designed as follows. First the implementations of CPDP methods are based on the Weka framework and all the classifiers use the default value of hyperparameters. Second, we perform a comparative study for all the CPDP methods based on all the experimental subjects in terms of two widely used performance measures (ie, F1 and AUC). Finally, to perform a pair of CPDP, we treat each source project individually.²³

To rank all the CPDP methods in terms of a specific prediction performance, we use the Scott-Knott test,⁸⁸ which was recommended by Ghotra et al⁸⁹ when they ranked different SDP methods. The Scott-Knott test does not suffer from the overlapping groups issue in post hoc tests (such as the Friedman-Nemenyi test). The Scott-Knott test is utilized to analyze whether some CPDP methods outperform others and can generate a global ranking of these CPDP methods. In particular, the Scott-Knott test performs the grouping process in a recursive manner. Firstly, the Scott-Knott test uses a hierarchical cluster analysis method to partition all the CPDP methods into two ranks based on the mean performance in terms of a specific performance measure (such as AUC or F1). Then if the divided ranks are significantly different, the Scott-Knott test is recursively executed again in each rank to further divide the ranks. When ranks can no longer be divided into statistically distinct ranks, this test will terminate.

We use Wilcoxon signed-rank test to analyze whether the performance difference between two CPDP methods are statistically significant. We also use the Benjamini-Hochberg (BH) procedure to adjust P values if we make multiple comparisons. Then if the test result shows a significant difference, we compute Cliff δ , which is a nonparametric effect size measure, to examine whether the magnitude of the difference between two CPDP methods is substantial or not. The meaning of different Cliff δ value and their corresponding interpretation suggested by Benjamini⁹⁰ can be found in Table 10. In summary, a CPDP method performs significantly better or worse than another CPDP method, if BH corrected P value is less than .05 and the effectiveness level is not negligible based on Cliff δ , while the difference between two CPDP methods is not significant, if P value is not less than .05 or P value is less than .05 and the effectiveness level is negligible.

4.1.3 | Results

The comparison results based on the Scott-Knott test on different experimental subjects can be found in Figures 3 to 7. Based on these five figures, in terms of F1 performance measure, we can find that the performance of four unsupervised methods is always in the top 4. Especially on AEEEM, ReLink, SOFTLAB, and PROMISE datasets, these methods can be divided in the the ground with the first rank. In terms of AUC performance measure, we can also find that most of the unsupervised methods can be divided in the group with the first rank. These results based on the same experimental setup further reconfirm the findings in previous studies on unsupervised CPDP methods and verify the competitiveness of these unsupervised methods.⁸⁻¹⁰

Then we investigate the feasibility of state-of-the-art CPDP methods in our empirical studies. Here, we compute the satisfactory ratio of different CPDP methods on a specific datasets to evaluate the feasibility of these methods. Taking ReLink dataset as an example, there are 6 ($=3 \times 2$) pairs of CPDP, if the CPDP method can achieve satisfactory performance in three pairs, then the satisfactory ratio is 50% ($=3/6$). In this article, we consider two different satisfactory criteria suggested by previous studies. These suggestions are based on threshold-dependent performance measures (such as precision, recall, and accuracy). The first suggestion on the satisfactory criterion (ie, accuracy, precision, and recall are all larger than

TABLE 10 Cliff δ and corresponding effectiveness level⁹⁰

Cliff δ	Effectiveness Level
$ \delta < 0.147$	Negligible
$0.147 \leq \delta < 0.33$	Small
$0.33 \leq \delta < 0.474$	Medium
$0.474 \leq \delta $	Large

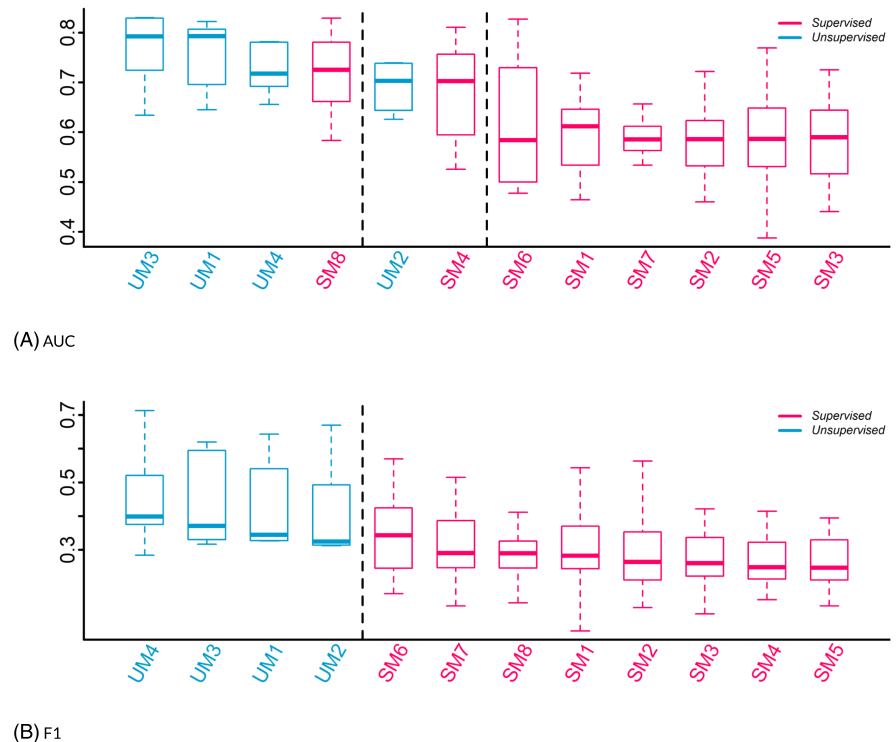


FIGURE 3 Scott-Knott test results on AEEEM dataset

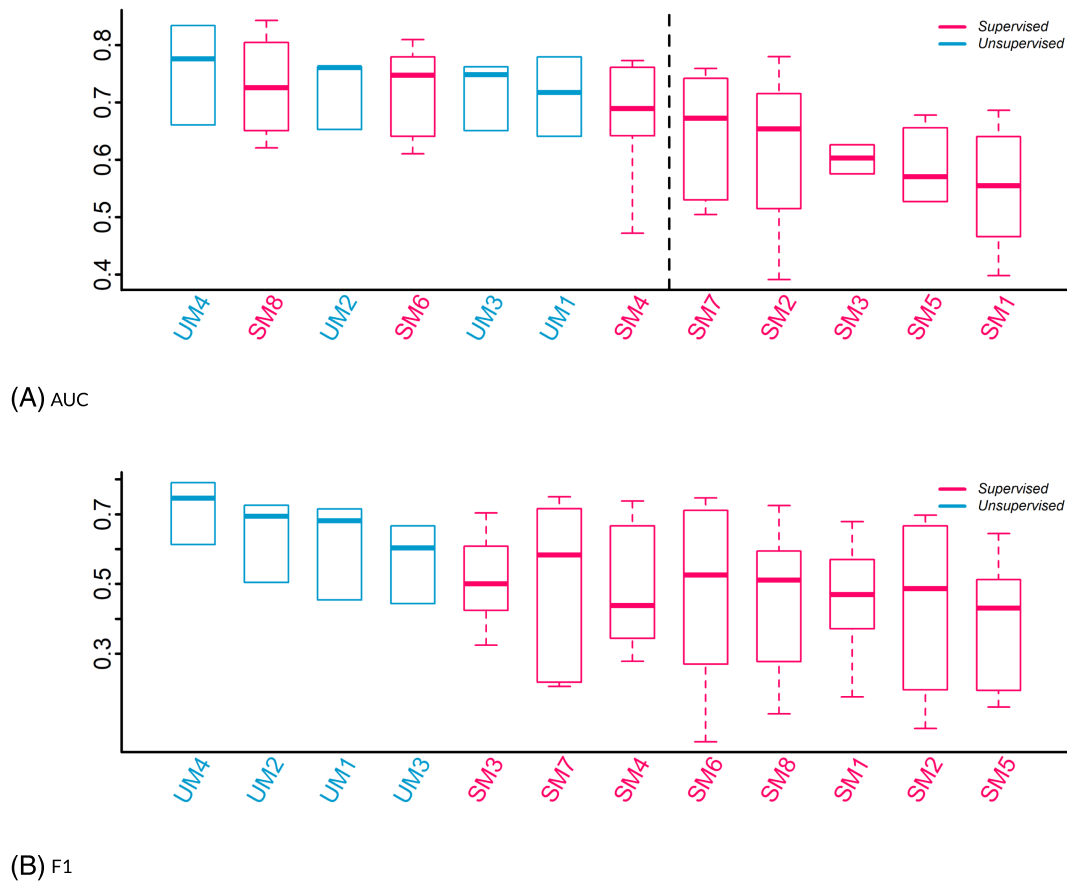


FIGURE 4 Scott-Knott test results on ReLink dataset

75%) is proposed by Zimmermann et al.¹² The second suggestion on the satisfactory criterion (ie, recall is larger than 70% and precision is larger than 50%) is proposed by He et al,¹³ since He et al noticed that high precision is difficult to achieve due to the class imbalanced problem in most of the gathered SDP datasets. Obviously, the second suggestion¹³ sets a higher criterion for the satisfactory performance than the first suggestion.¹² Based on these two different satisfactory criteria, we want to find the recent advances achieved by state-of-the-art CPDP methods.

The final results can be found in Table 11. In this table, we use SC1 to denote the criterion suggested by Zimmermann et al¹² and SC2 to denote the criterion suggested by He et al.¹³ The number of CPDP pairs for each dataset is listed in the parentheses after the name of the dataset. When considering SC1, none of the chosen CPDP methods can obtain satisfactory prediction performance on NASA, ReLink and AEEEM datasets. When considering SC2, on NASA dataset, there is still no chosen CPDP method, which can obtain satisfactory prediction performance. In summary, the satisfactory ratio of state-of-the-art CPDP methods on different datasets is still pessimistic. Therefore, CPDP is still a serious challenge issue in current defect prediction research domain.

Finally, we analyze the computational cost of different CPDP methods. In this article, we only record the computational cost on the model construction phase. All the CPDP methods are run on Win 10 operation system (Intel i5-4210U CPU with 8 GB of Memory). The average computational cost on each CPDP pair for different CPDP methods can be found in Table 12. Notice we do not list the computational cost of the CPDP method UM4. Since this method only uses the ranking operation on a specific metric, it is very simple and the computational cost can be negligible. In this table, we can find the following: (a) For the unsupervised methods, the method UM1 has the lowest computational cost, since this method only uses a simple cluster algorithm. The method UM3 has highest computational cost due to the high cost of some matrix operations used in this method, especially when the experimental subjects have more program modules or consider more metrics to measure the extracted modules. (b) For the supervised methods, the methods SM7 and SM6 have the highest computational cost. The high computational cost of the method SM6 is caused by the used TCA method and the reason for the high computational cost of SM7 is that this ensemble learning based method is based on six different classifiers.

4.2 | Result Analysis for RQ2

RQ2: Whether different CPDP methods can predict the same defective modules?

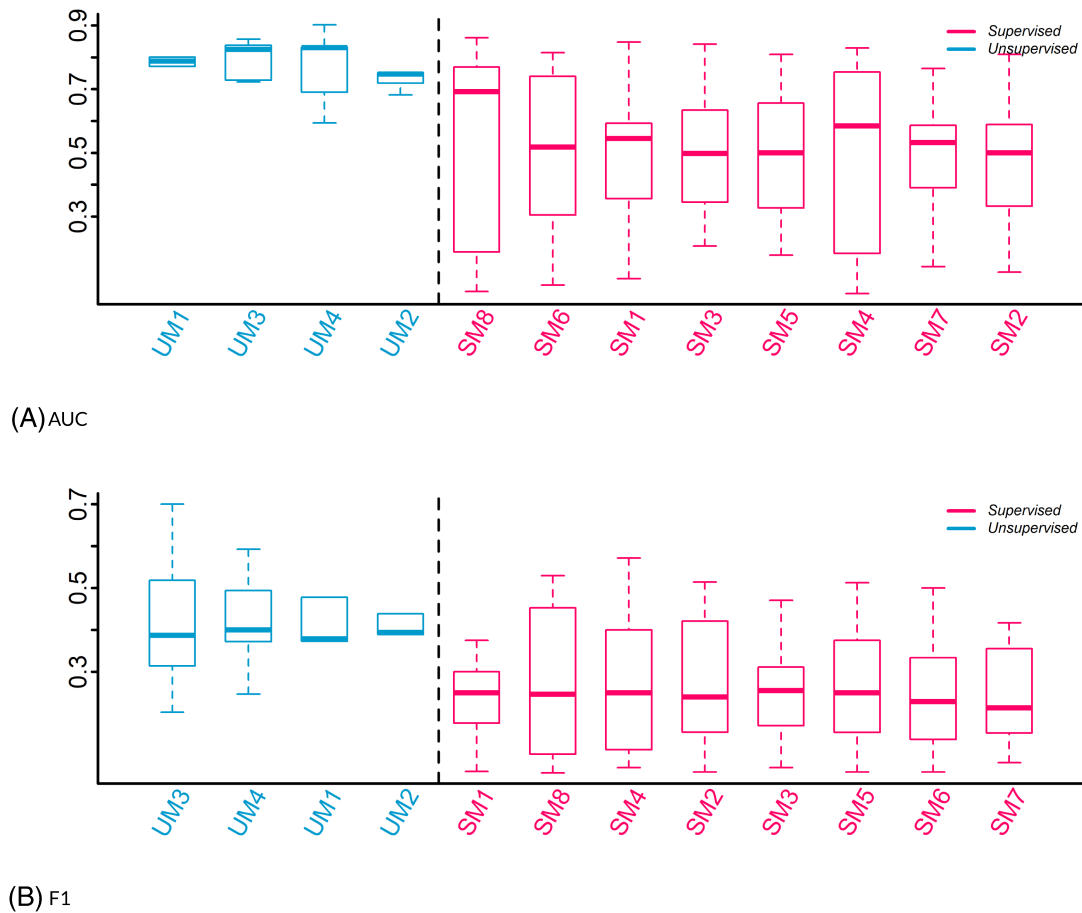


FIGURE 5 Scott-Knott test results on SOFTLAB dataset

4.2.1 | Motivation

In RQ1, we mainly compare and rank the performance of different CPDP methods in terms of AUC and F1 performance measures. In this RQ, we conduct an in-depth analysis through the analysis of defective modules identified by each CPDP method. We want to investigate whether there exists a difference between two different CPDP methods in terms of the specific defective modules each method identifies and does not identify.

4.2.2 | Approach

To evaluate whether different CPDP methods result in distinct predictions, we use the McNemar test to perform diversity analysis on defective modules. Moreover, we also want to identify the defective modules, which cannot be correctly predicted by any of the chosen CPDP methods. These findings can help to design more effective CPDP methods.

4.2.3 | Results

We first perform diversity analysis on defective modules for different supervised CPDP methods. The final results can be found in Table 13. In this table, we find that prediction diversity phenomenon on defective modules for different supervised CPDP methods exists in most cases. When analyzing from the method perspective, we can find prediction diversity on defective modules exists in half of CPDP pairs between different supervised methods except for SM2 vs SM3, SM2 vs SM5, and SM3 vs SM5. In the best case, when SM4 vs SM6, prediction diversity on defective modules exists in 83% of CPDP pairs. When analyzing from the dataset perspective, we can find the prediction diversity on defective modules exists in half of CPDP pairs except for SOFTLAB dataset.

Then we perform diversity analysis on defective modules for different unsupervised CPDP methods. The final results can be found in Table 14. In this table, we find that prediction diversity phenomenon on defective modules for different unsupervised CPDP methods still exists. However, this phenomenon is less obvious than the comparison between supervised methods. When analyzing from the method perspective, we can find prediction diversity on defective modules exists in half of CPDP pairs between different supervised methods except for UM1 vs UM2, UM1 vs

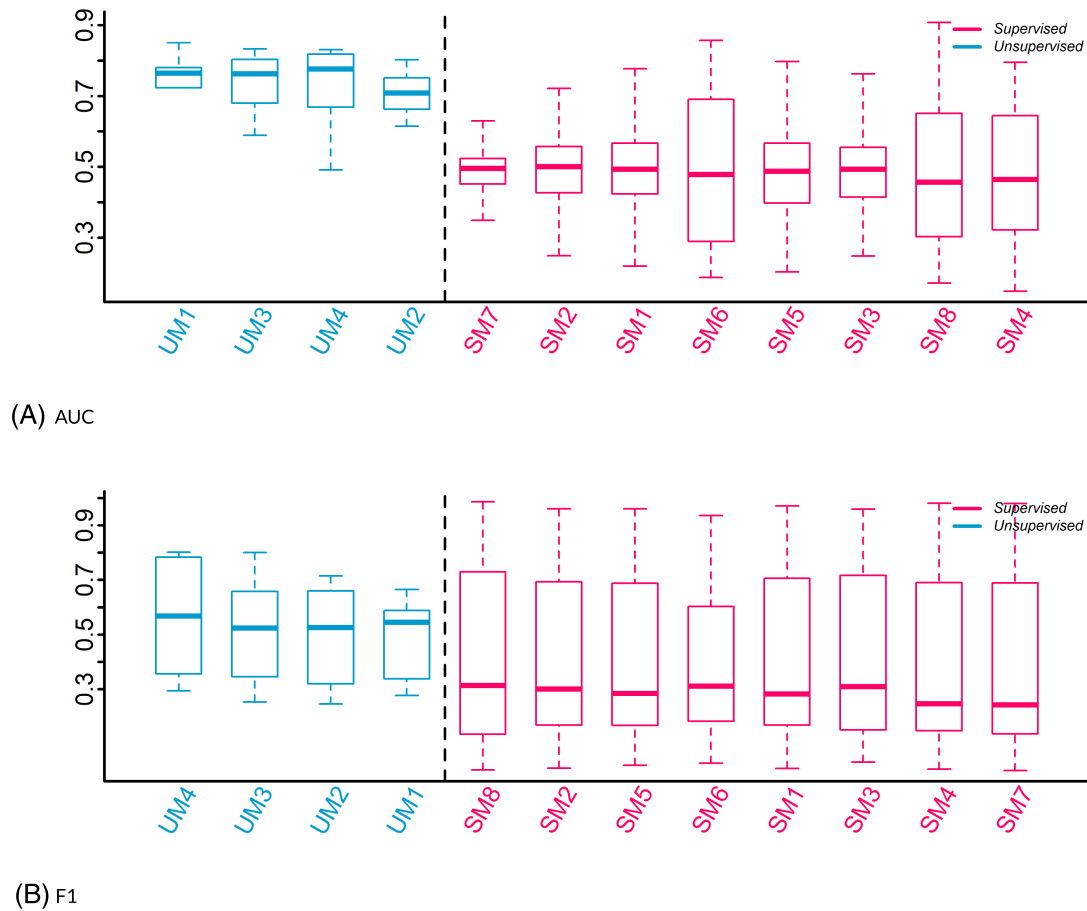


FIGURE 6 Scott-Knott test results on PROMISE dataset

UM4, and UM2 vs UM4. In the best case, when SM1 vs UM3 or UM2 vs UM3, prediction diversity on defective modules exists in 63% of CPDP pairs. When analyzing from the dataset perspective, we can find the prediction diversity on defective modules exists in half of CPDP pairs except for ReLink, AEEEM, and SOFTLAB datasets.

Finally, we perform diversity analysis on defective modules between supervised CPDP methods and unsupervised CPDP methods. The final results can be found in Table 15. In this table, we find that prediction diversity phenomenon on defective modules when comparing supervised methods and unsupervised methods still exists. When analyzing from the method perspective, we can find prediction diversity on defective modules exists in half of CPDP pairs between different supervised methods in all the method comparisons. In the best case, when SM4 vs UM2, prediction diversity on defective modules exists in 94% of CPDP pairs. When analyzing from the dataset perspective, we can find the prediction diversity on defective modules exists in half of CPDP pairs in all the datasets.

In summary, the prediction diversity on defective modules is most obvious when the comparison is performed between supervised methods and unsupervised methods, while this phenomenon is least obvious when the comparison is performed between different unsupervised methods.

Moreover, we analyze the number of defective modules (ie, =0), which cannot be identified by any of the chosen CPDP methods and the number of defective modules (ie, =1), which can be only identified by one of the chosen CPDP methods. Due to the page limitation of this article, we only list the results on AEEEM dataset and ReLink dataset. These results can be found in Tables 16 and 17, respectively. In these two tables, SM denotes that we only consider supervised CPDP methods, UM denotes that we only consider unsupervised CPDP methods, and ALL denotes that we consider all the chosen CPDP methods. The results on other datasets can be found in the project website. In Table 16, when PDE is set to the source project and ML is set to the target project; there are 28, 53, and 25 defective modules, which cannot be identified by any of the supervised methods, the unsupervised methods, and all the methods. Moreover, there are 28, 61, and 17 defective modules, which can only be identified by one of the supervised methods, one of the unsupervised methods, and one of all the chosen methods.

Based on these results, we first find there exists performance bottleneck in state-of-the-art CPDP methods, since some of defective modules cannot be identified by any of our chosen CPDP methods. Then we find there exists performance improvement opportunity for design new CPDP methods. One potential solution is to use ensemble learning. Ensemble learning can construct a set of individual classifiers trained on the same dataset, and these classifiers are combined to perform a prediction task. An overall prediction decision is made by the ensemble based on the

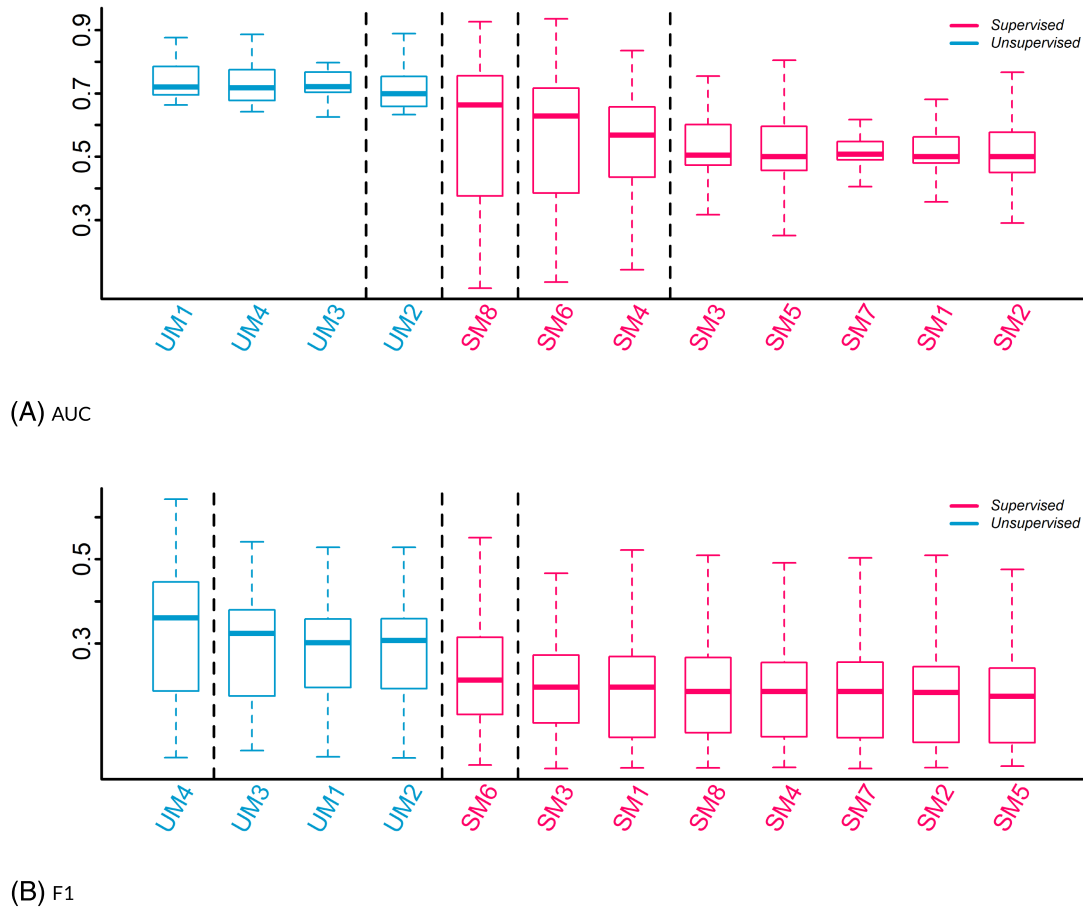


FIGURE 7 Scott-Knott test results on NASA dataset

TABLE 11 The ratio of CPDP with satisfactory performance on the five datasets with different satisfactory criteria

CPDP Method	PROMISE (90)		NASA (132)		SOFTLAB (20)		ReLink (6)		AEEEM (20)	
	SC1	SC2	SC1	SC2	SC1	SC2	SC1	SC2	SC1	SC2
SM1	8.89%	24.44%	0.00%	0.00%	0.00%	5.00%	0.00%	16.67%	0.00%	0.00%
SM2	10.00%	24.44%	0.00%	0.00%	0.00%	0.00%	0.00%	33.33%	0.00%	0.00%
SM3	8.89%	22.22%	0.00%	0.00%	0.00%	5.00%	0.00%	16.67%	0.00%	0.00%
SM4	11.11%	25.56%	0.00%	0.00%	0.00%	0.00%	0.00%	16.67%	0.00%	0.00%
SM5	11.11%	21.11%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
SM6	7.78%	20.00%	0.00%	0.00%	0.00%	0.00%	0.00%	50.00%	0.00%	0.00%
SM7	10.00%	23.33%	0.00%	0.00%	0.00%	0.00%	0.00%	33.33%	0.00%	0.00%
SM8	12.22%	25.56%	0.00%	0.00%	0.00%	0.00%	0.00%	16.67%	0.00%	0.00%
UM1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	66.67%	0.00%	0.00%
UM2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	66.67%	0.00%	20.00%
UM3	0.00%	10.00%	0.00%	0.00%	0.00%	20.00%	0.00%	0.00%	0.00%	0.00%
UM4	0.00%	20.00%	0.00%	8.33%	0.00%	0.00%	33.33%	100.00%	0.00%	20.00%

predictions of the individual classifiers. However, based on Tables 16 and 17, we find that using majority voting strategy may not be suitable for CPDP issue, since using this strategy will miss the unique subsets of defective modules, which can be only identified by individual CPDP method. One possible solution is to use stacking ensemble, which uses an additional classifier to make the final prediction. The effectiveness of the stacking ensemble is verified in some of previous CPDP studies.^{48,91} However, a substantial amount of future work is needed to design effective stacking ensemble-based methods, which will fully exploit our findings in this article.

TABLE 12 Computational cost of different CPDP methods (unit: millisecond)

Method	Relink	AEEM	SOFTLAB	PROMISE	NASA
SM1	149.17	189.25	50.45	31.94	242.65
SM2	151.67	217.70	49.85	39.97	304.04
SM3	140.50	158.40	42.75	31.14	204.08
SM4	6875.83	14.55	16.64	114.30	608.64
SM5	171.50	192.55	46.00	40.06	229.42
SM6	353.33	1262.80	84.25	162.06	8867.14
SM7	1240.83	23265.55	536.10	1882.28	9843.52
SM8	309.67	472.15	87.65	151.32	1082.02
UM1	88.00	145.20	59.20	50.70	83.75
UM2	130.33	184.60	79.20	85.00	240.08
UM3	960.10	16916.29	436.28	3427.56	2261465.90

TABLE 13 Diversity analysis on defective modules for different supervised CPDP methods

Method	ReLink	AEEM	PROMISE	NASA	SOFTLAB	In Summary
SM1 vs SM2	5/6	10/20	44/90	83/132	3/20	145/268
SM1 vs SM3	3/6	11/20	46/90	46/132	2/20	108/268
SM1 vs SM4	5/6	16/20	45/90	68/132	3/20	137/268
SM1 vs SM5	5/6	9/20	47/90	72/132	1/20	134/268
SM1 vs SM6	6/6	19/20	72/90	106/132	8/20	211/268
SM1 vs SM7	6/6	14/20	47/90	78/132	4/20	149/268
SM1 vs SM8	6/6	18/20	51/90	66/132	4/20	145/268
SM2 vs SM3	5/6	13/20	43/90	78/132	6/20	145/268
SM2 vs SM4	3/6	10/20	41/90	62/132	1/20	117/268
SM2 vs SM5	4/6	11/20	33/90	40/132	3/20	91/268
SM2 vs SM6	5/6	18/20	67/90	92/132	7/20	189/268
SM2 vs SM7	3/6	12/20	36/90	65/132	2/20	118/268
SM2 vs SM8	3/6	11/20	40/90	61/132	2/20	117/268
SM3 vs SM4	6/6	15/20	34/90	74/132	4/20	133/268
SM3 vs SM5	5/6	9/20	40/90	66/132	4/20	124/268
SM3 vs SM6	6/6	17/20	78/90	98/132	9/20	208/268
SM3 vs SM7	6/6	14/20	43/90	79/132	7/20	149/268
SM3 vs SM8	4/6	14/20	50/90	63/132	9/20	140/268
SM4 vs SM5	2/6	15/20	33/90	64/132	1/20	115/268
SM4 vs SM6	3/6	18/20	77/90	104/132	4/20	206/268
SM4 vs SM7	2/6	13/20	32/90	62/132	3/20	112/268
SM4 vs SM8	4/6	12/20	29/90	62/132	0/20	107/268
SM5 vs SM6	6/6	19/20	69/90	89/132	7/20	190/268
SM5 vs SM7	4/6	10/20	38/90	63/132	4/20	119/268
SM5 vs SM8	3/6	14/20	42/90	59/132	3/20	121/268
SM6 vs SM7	1/6	17/20	71/90	98/132	7/20	194/268
SM6 vs SM8	4/6	18/20	77/90	103/132	5/20	207/268
SM7 vs SM8	2/6	13/20	38/90	54/132	3/20	110/268
In Summary	117/168	390/560	1363/2520	2055/3696	116/560	4041/7504

TABLE 14 Diversity analysis on defective modules for different unsupervised CPDP methods

Method	ReLink	AEEM	PROMISE	NASA	SOFTLAB	In Summary
UM1 vs UM2	1/3	2/5	5/10	3/12	0/5	11/35
UM1 vs UM3	2/3	1/5	9/10	10/12	0/5	22/35
UM1 vs UM4	1/3	2/5	8/10	3/12	0/5	14/35
UM2 vs UM3	2/3	3/5	6/10	10/12	1/5	22/35
UM2 vs UM4	0/3	2/5	2/10	3/12	0/5	7/35
UM3 vs UM4	2/3	4/5	4/10	10/12	0/5	20/35
In Summary	8/18	14/30	34/60	39/72	1/30	96/210

TABLE 15 Diversity analysis on defective modules between supervised CPDP methods and unsupervised CPDP methods

Method	ReLink	AEEM	PROMISE	NASA	SOFTLAB	In Summary
SM1 vs UM1	4/6	18/20	74/90	124/132	10/20	230/268
SM1 vs UM2	5/6	17/20	73/90	124/132	10/20	229/268
SM1 vs UM3	4/6	18/20	75/90	116/132	7/20	220/268
SM1 vs UM4	5/6	18/20	76/90	116/132	8/20	223/268
SM2 vs UM1	4/6	19/20	78/90	117/132	12/20	230/268
SM2 vs UM2	4/6	18/20	77/90	116/132	12/20	227/268
SM2 vs UM3	4/6	19/20	77/90	117/132	12/20	229/268
SM2 vs UM4	4/6	18/20	80/90	112/132	12/20	226/268
SM3 vs UM1	4/6	17/20	76/90	122/132	10/20	229/268
SM3 vs UM2	5/6	16/20	73/90	122/132	10/20	226/268
SM3 vs UM3	4/6	17/20	75/90	119/132	4/20	219/268
SM3 vs UM4	5/6	17/20	75/90	114/132	6/20	217/268
SM4 vs UM1	5/6	20/20	77/90	118/132	12/20	232/268
SM4 vs UM2	5/6	20/20	74/90	118/132	12/20	229/268
SM4 vs UM3	5/6	20/20	72/90	121/132	7/20	225/268
SM4 vs UM4	5/6	19/20	73/90	120/132	10/20	227/268
SM5 vs UM1	6/6	17/20	77/90	117/132	11/20	228/268
SM5 vs UM2	6/6	16/20	72/90	118/132	11/20	223/268
SM5 vs UM3	5/6	17/20	72/90	111/132	10/20	215/268
SM5 vs UM4	6/6	18/20	71/90	116/132	11/20	222/268
SM6 vs UM1	5/6	18/20	67/90	107/132	10/20	207/268
SM6 vs UM2	5/6	17/20	71/90	108/132	10/20	211/268
SM6 vs UM3	5/6	18/20	75/90	100/132	10/20	208/268
SM6 vs UM4	5/6	16/20	76/90	102/132	9/20	208/268
SM7 vs UM1	4/6	16/20	74/90	120/132	11/20	225/268
SM7 vs UM2	4/6	18/20	73/90	121/132	11/20	227/268
SM7 vs UM3	5/6	17/20	74/90	125/132	9/20	230/268
SM7 vs UM4	3/6	16/20	78/90	121/132	12/20	230/268
SM8 vs UM1	5/6	20/20	80/90	117/132	12/20	234/268
SM8 vs UM2	5/6	20/20	78/90	117/132	11/20	231/268
SM8 vs UM3	4/6	19/20	77/90	118/132	12/20	230/268
SM8 vs UM4	5/6	20/20	77/90	116/132	12/20	230/268
In summary	150/192	574/640	2397/2880	3730/4224	326/640	7177/8576

TABLE 16 Defective modules detection ability for different CPDP methods on AEEEM dataset

Source \geq Target	=0 by SM	=1 by SM	=0 by UM	=1 by UM	=0 by ALL	=1 by ALL
EQ \geq JDT	1	3	26	17	1	0
EQ \geq LC	2	4	7	9	0	2
EQ \geq ML	51	11	53	61	37	15
EQ \geq PDE	0	10	30	34	0	7
JDT \geq EQ	0	32	15	22	0	8
JDT \geq LC	0	13	7	9	0	3
JDT \geq ML	90	29	53	61	41	55
JDT \geq PDE	0	75	30	34	0	25
LC \geq EQ	75	20	15	22	14	18
LC \geq JDT	0	33	26	17	0	17
LC \geq ML	82	5	53	61	33	45
LC \geq PDE	100	39	30	34	28	29
ML \geq EQ	64	20	15	22	15	15
ML \geq JDT	0	40	26	17	0	16
ML \geq LC	25	16	7	9	7	6
ML \geq PDE	88	33	30	34	26	21
PDE \geq EQ	0	61	15	22	0	14
PDE \geq JDT	18	24	26	17	10	16
PDE \geq LC	10	7	7	9	2	5
PDE \geq ML	56	50	53	61	30	43

TABLE 17 Defective modules detection ability for different CPDP methods on ReLink dataset

Source \geq Target	=0 by SM	=1 by SM	=0 by UM	=1 by UM	=0 by ALL	=1 by ALL
Apache \geq Zxing	2	23	32	6	2	23
Apache \geq Safe	2	2	5	0	2	2
Zxing \geq Apache	3	9	25	3	3	8
Zxing \geq Safe	2	3	5	0	2	2
Safe \geq Apache	8	11	25	3	7	12
Safe \geq Zxing	29	18	32	6	23	2

5 | THREATS TO VALIDITY

In this section, we mainly discuss the potential threats to validity in our empirical studies.

Threats to internal validity are mainly concerned with the uncontrolled internal factors that might have influence on the experimental results. The main internal threat is the potential faults in the implementations of these CPDP methods. To reduce this threat, we consider the implementations of supervised CPDP methods provided by CrossPare platform.²¹ For unsupervised CPDP methods, we implement these methods⁸⁻¹⁰ according to the description of these methods, and the prediction performance is very close to the results in the corresponding papers.

Threats to external validity are about whether the observed experimental results can be generalized to other experimental subjects. First, our empirical results are based on the analysis of five datasets gathered from real-world projects. Though these datasets have been widely used in previous CPDP studies and the usage statistics can be found in Table 6, these empirical results might not hold when considering other projects from the open-source community or commercial enterprises. Second, the selection of CPDP methods may influence the empirical results. In this article, we select eight supervised CPDP methods based on four criteria in Section 3.2.1 and four recently proposed unsupervised CPDP methods. Therefore, it can be ensured that the selected CPDP methods reflect the latest research progress on CPDP issue. Third, dataset quality issue is unavoidable during the process of dataset gathering. For example, when using SZZ^{9,22} to label extracted program modules, it may classify style changes (such as modifications to code comments) as bug-introducing changes, and this can mislabel nondefective modules as defective modules. Some

researchers have manually checked the gathered datasets.⁸³ However, this problem is still an open problem and needs more investigations to further improve the dataset quality.

Threats to conclusion validity are mainly concerned with inappropriate use of statistical techniques. To better rank all the CPDP methods in terms of a specific performance measure, we use the Scott-Knott test,⁸⁸ since recent studies⁸⁹ have suggested that the Scott-Knott test is superior to some post hoc tests (such as the Friedman-Nemenyi test).

Threats to construct validity are about whether the performance measures used in the empirical studies reflect the real-world situation. In previous studies, many performance measures are used. In this article, we use AUC and F1 to evaluate the performance of different CPDP methods. F1 is chosen since this performance measure is widely used in previous CPDP studies.⁷ However, F1 is a threshold-dependent performance measure. The value of F1 depends on the selected threshold to decide whether a software module is defective or nondefective. Moreover, F1 is sensitive to the class imbalanced problem. Therefore, we further consider AUC to evaluate the performance of different CPDP methods. Except for these traditional performance measures, we want to consider some effort-aware performance measures,^{93,94} which consider the limitation of available testing resources.

6 | CONCLUSION AND FUTURE WORK

In this article, we mainly investigate an interesting problem: Do different CPDP methods identify the same defective modules? In our empirical studies, we choose 12 state-of-the-art CPDP methods and five commonly used datasets from real-world projects. Final results verify that different CPDP methods may lead to difference in the identified defective modules, especially when the comparison is performed between the supervised methods and the unsupervised methods. Moreover, we also find there exist a certain number of defective modules, which cannot be correctly identified by any of CPDP methods. These findings can be utilized to design more effective methods to further improve the performance of CPDP.

In the future, we want to further investigate the following issues. First, we want to verify the generalization of our empirical results by considering more datasets gathered from the commercial projects and the open-source projects. Second, since identifying defects with high-severity is more important than identifying defects with low severity, our study provides an opportunity for future studies to identify CPDP methods, which have stronger detection ability in high-severity defects. Finally, this study shows there is a large room for CPDP performance improvement, and more effective CPDP methods based on ensemble learning or deep learning should be designed in the future.

ACKNOWLEDGEMENTS

The authors would like to thank the editors and the anonymous reviewers for their insightful comments and suggestions, which can substantially improve the quality of this work. Xiang Chen and Yanzhou Mu have contributed equally for this work and they are co-first authors. Detailed experimental results can be found in our project web site: <https://github.com/EzioQR/jsep.git>.

ORCID

Xiang Chen  <https://orcid.org/0000-0002-1180-3891>

Yubin Qu  <https://orcid.org/0000-0001-5222-4020>

Chao Ni  <https://orcid.org/0000-0002-2906-0598>

REFERENCES

- Hall T, Beecham S, Bowes D, Gray D, Counsell S. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*. 2012;38(6):1276-1304.
- Kamei Y, Shihab E. Defect prediction: accomplishments and future challenges. In: *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering*; 2016:33-45.
- Li Z, Jing XY, Zhu X. Progress on approaches to software defect prediction. *IET Soft*. 2018;12(3):161-175.
- Radjenovic D, Hericko M, Torkar R, Zivkovic A. Software fault prediction metrics: a systematic literature review. *Inf Softw Technol*. 2013;55(8):1397-1418.
- Turhan B, Menzies T, Bener AB, Di Stefano J. On the relative value of cross-company and within-company data for defect prediction. *Empir Softw Eng*. 2009;14(5):540-578.
- Pan SJ, Yang Q. A survey on transfer learning. *IEEE Trans Knowl Data Eng*. 2010;22(10):1345-1359.
- Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Trans Softw Eng*. 2019;45(2):111-147.
- Nam J, Kim S. CLAMI: defect prediction on unlabeled datasets. In: *Proceedings of International Conference on Automated Software Engineering*; 2015:452-463.
- Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: *Proceedings of the International Conference on Software Engineering*; 2016:309-320.

10. Zhou Y, Yang Y, Lu H, et al. How far we have progressed in the journey? An examination of cross-project defect prediction. *ACM Trans Softw Eng Methodol*. 2018;27(1):1:1-1:51.
11. Chen X, Zhang D, Zhao Y, Cui Z, Ni C. Software defect number prediction: unsupervised vs supervised methods. *Inf Softw Technol*. 2019;106:161-181.
12. Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: *Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering*; 2009:91-100.
13. He Z, Shu F, Yang Y, Li M, Wang Q. An investigation on the feasibility of cross-project defect prediction. *Autom. Softw Eng*. 2012;19(2):167-199.
14. Briand LC, Melo WL, Wüst J. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE transactions on Software Engineering*. 2002;7:706-720.
15. Jureczko M, Madeyski L. Towards identifying software project clusters with regard to defect prediction. In: *Proceedings of the International Conference on Predictive Models in Software Engineering*; 2010:9:1-9:10.
16. Fukushima T, Kamei Y, McIntosh S, Yamashita K, Ubayashi N. An empirical study of just-in-time defect prediction using cross-project models. In: *Proceedings of the Working Conference on Mining Software Repositories*; 2014:172-181.
17. Kamei Y, Fukushima T, McIntosh S, Yamashita K, Ubayashi N, Hassan AE. Studying just-in-time defect prediction using cross-project models. *Empir Softw Eng*. 2016;21(5):2072-2106.
18. Kamei Y, Shihab E, Adams B, et al. A large-scale empirical study of just-in-time quality assurance. *IEEE Trans Software Eng*. 2013;39(6):757-773.
19. Rahman F, Posnett D, Devanbu P. Recalling the imprecision of cross-project defect prediction. In: *Proceedings of the International Symposium on Foundations of Software Engineering*; 2012:61:1-61:11.
20. Turhan B. On the dataset shift problem in software engineering prediction models. *Empir Softw Eng*. 2012;17(1-2):62-74.
21. Herbold S, Trautsch A, Grabowski J. A comparative study to benchmark cross-project defect prediction approaches. *IEEE Trans Softw Eng*. 2018;44(9):811-833.
22. Camargo Cruz AE, Ochimizu K. Towards logistic regression models for predicting fault-prone code across software projects. In: *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*; 2009:460-463.
23. Nam J, Pan SJ, Kim S. Transfer defect learning. In: *Proceedings of the International Conference on Software Engineering*; 2013:382-391.
24. Zhang F, Keivanloo I, Zou Y. Data transformation in cross-project defect prediction. *Empir Softw Eng*. 2017;22(6):3186-3218.
25. Liu C, Yang D, Xia X, Yan M, Zhang X. A two-phase transfer learning model for cross-project defect prediction. *Inf Softw Technol*. 2019;107:125-136.
26. Krishna R, Menzies T, Fu W. Too much automation? The bellwether effect and its implications for transfer learning. In: *Proceedings of International Conference on Automated Software Engineering*; 2016:122-131.
27. Krishna R, Menzies T. Bellwethers: A baseline method for transfer Learning. *IEEE Transactions on Software Engineering*. 2018. <https://doi.org/10.1109/TSE.2018.2821670>. <https://ieeexplore.ieee.org/document/8329264>
28. Menzies T, Butcher A, Marcus A, Zimmermann T, Cok D. Local vs. global models for effort estimation and defect prediction. In: *Proceedings of International Conference on Automated Software Engineering*; 2011:343-351.
29. Menzies T, Butcher A, Cok D, et al. Local versus global lessons for defect prediction and effort estimation. *IEEE Trans Softw Eng*. 2013;39(6):822-834.
30. Bettenburg N, Nagappan M, Hassan AE. Think locally, act globally: improving defect and effort prediction models. In: *Proceedings of the Working Conference on Mining Software Repositories*; 2012:60-69.
31. Bettenburg N, Nagappan M, Hassan AE. Towards improving statistical modeling of software engineering data: think locally. *Act Glob Empir Softw Eng*. 2015;20(2):294-335.
32. Herbold S, Trautsch A, Grabowski J. Global vs. local models for cross-project defect prediction. *Empir Softw Eng*. 2017;22(4):1866-1902.
33. Peters F, Menzies T, Marcus A. Better cross company defect prediction. In: *Proceedings of the Working Conference on Mining Software Repositories*; 2013:409-418.
34. Herbold S. Training data selection for cross-project defect prediction. In: *Proceedings of the International Conference on Predictive Models in Software Engineering*; 2013:6:1-6:10.
35. Hosseini S, Turhan B, Mäntylä M. Search based training data selection for cross project defect prediction. In: *Proceedings of the International Conference on Predictive Models in Software Engineering*; 2016:3:1-3:10.
36. Hosseini S, Turhan B, Mäntylä M. A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. *Inf Softw Technol*. 2018;95:296-312.
37. Bin Y, Zhou K, Lu H, Zhou Y, Xu B. Training data selection for cross-project defection prediction: which approach is better?. In: *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*; 2017:354-363.
38. Xu Z, Li S, Tang Y, et al. Cross version defect prediction with representative data via sparse subset selection. In: *Proceedings of the Conference on Program Comprehension*; 2018:132-143.
39. Ma Y, Luo G, Zeng X, Chen A. Transfer learning for cross-company software defect prediction. *Inf Softw Technol*. 2012;54(3):248-256.
40. Poon WN, Bennin KE, Huang J, Phannachitta P, Keung JW. Cross-project defect prediction using a credibility theory based naive Bayes classifier. In: *Proceedings of the International Conference on Software Quality, Reliability and Security*; 2017:434-441.
41. Ni C, Chen X, Wu F, Shen Y, Gu Q. An empirical study on Pareto based multi-objective feature selection for software defect prediction. *J Syst Softw*. 2019;152:215-238.

42. Liu W, Liu S, Gu Q, Chen J, Chen X, Chen D. Empirical studies of a two-stage data preprocessing approach for software fault prediction. *IEEE Trans Reliab.* 2016;65(1):38-53.
43. Liu S, Chen X, Liu W, Chen J, Gu Q, Chen D. FECAR: a feature selection framework for software defect prediction. In: Proceedings of the Annual Computer Software and Applications Conference; 2014:426-435.
44. Pan SJ, Tsang IW, Kwok JT, Yang Q. Domain adaptation via transfer component analysis. *IEEE Trans Neural Netw.* 2011;22(2):199-210.
45. He P, Li B, Liu X, Chen J, Ma Y. An empirical study on software defect prediction with a simplified metric set. *Inf Softw Technol.* 2015;59:170-190.
46. Ni C, Liu W, Gu Q, Chen X, Chen D. FeSCH: a feature selection method using clusters of hybrid-data for cross-project defect prediction. In: Proceedings of Annual Computer Software and Applications Conference; 2017:51-56.
47. Ni C, Liu WS, Chen X, Gu Q, Chen DX, Huang QG. A cluster based feature selection method for cross-project software defect prediction. *J Comput Sci Technol.* 2017;32(6):1090-1107.
48. Panichella A, Oliveto R, De Lucia A. Cross-project defect prediction models: L'union fait la force. In: Proceedings of the International Conference on Software Maintenance, Reengineering and Reverse Engineering; 2014:164-173.
49. Zhang Y, Lo D, Xia X, Sun J. An empirical study of classifier combination for cross-project defect prediction. In: Proceedings of Annual Computer Software and Applications Conference; 2015:264-269.
50. Ryu D, Choi O, Baik J. Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Empir Softw Eng.* 2016;21(1):43-71.
51. Ryu D, Jang JI, Baik J. A hybrid instance selection using nearest-neighbor for cross-project defect prediction. *J Comput Sci Technol.* 2015;30(5):969-980.
52. Ryu D, Jang JI, Baik J. A transfer cost-sensitive boosting approach for cross-project defect prediction. *Softw Qual J.* 2017;25(1):235-272.
53. Limsettho N, Bennin KE, Keung JW, Hata H, Matsumoto K. Cross project defect prediction using class distribution estimation and oversampling. *Inf Softw Technol.* 2018;100:87-102.
54. Jing XY, Wu F, Dong X, Xu B. An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Trans Softw Eng.* 2017;43(4):321-339.
55. Wang S, Liu T, Nam J, Tan L. Deep semantic feature learning for software defect prediction. *IEEE Trans Softw Eng.* 2018:1-1.
56. Canfora G, De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichella S. Multi-objective cross-project defect prediction. In: Proceedings of the International Conference on Software Testing, Verification and Validation; 2013:252-261.
57. Canfora G, Lucia AD, Penta MD, Oliveto R, Panichella A, Panichella S. Defect prediction as a multiobjective optimization problem. *Softw Test Verif Rel.* 2015;25(4):426-459.
58. Ryu D, Baik J. Effective multi-objective naïve Bayes learning for cross-project defect prediction. *Appl Soft Comput.* 2016;49:1062-1077.
59. Feng Wang JH. A top-k learning to rank approach to cross-project software defect prediction. In: Proceedings of the Asia-Pacific Software Engineering Conference; 2018.
60. Yang X, Wen W. Ridge and Lasso regression models for cross-version defect prediction. *IEEE Trans Rel.* 2018;67(3):885-896.
61. Nam J, Kim S. Heterogeneous defect prediction. In: Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering; 2015:508-519.
62. Nam J, Fu W, Kim S, Menzies T, Tan L. Heterogeneous defect prediction. *IEEE Trans Softw Eng.* 2018;44(9):874-896.
63. Yu Q, Jiang S, Zhang Y. A feature matching and transfer approach for cross-company defect prediction. *J Syst Softw.* 2017;132:366-378.
64. Jing X, Wu F, Dong X, Qi F, Xu B. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In: Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering; 2015:496-507.
65. Li Z, Jing XY, Wu F, Zhu X, Xu B, Ying S. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. *Autom Softw Eng.* 2018;25(2):201-245.
66. Li Z, Jing XY, Zhu X, Zhang H. Heterogeneous defect prediction through multiple kernel learning and ensemble learning. In: Proceedings of the International Conference on Software Maintenance and Evolution; 2017:91-102.
67. Turhan B, Misirlı AT, Bener A. Empirical evaluation of the effects of mixed project data on learning defect predictors. *Inf Softw Technol.* 2013;55(6):1101-1118.
68. Xia X, David L, Pan SJ, Nagappan N, Wang X. Hydra: massively compositional model for cross-project defect prediction. *IEEE Trans Softw Eng.* 2016;42(10):977-998.
69. Chen L, Fang B, Shang Z, Tang Y. Negative samples reduction in cross-company software defects prediction. *Inf Softw Technol.* 2015;62:67-77.
70. Zhang ZW, Jing XY, Wang TJ. Label propagation based semi-supervised learning for software defect prediction. *Autom Softw Eng.* 2017;24(1):47-69.
71. Wu F, Jing XY, Sun Y, et al. Cross-project and within-project semisupervised software defect prediction: a unified approach. *IEEE Trans Reliab.* 2018;67(2):581-597.
72. Peters F, Menzies T. Privacy and utility for defect prediction: experiments with morph. In: Proceedings of the International Conference on Software Engineering; 2012:189-199.
73. Peters F, Menzies T, Gong L, Zhang H. Balancing privacy and utility in cross-company defect prediction. *IEEE Trans Softw Eng.* 2013;39(8):1054-1068.
74. Peters F, Menzies T, Layman L. LACE2: Better privacy-preserving data sharing for cross project defect prediction. In: Proceedings of the International Conference on Software Engineering; 2015:801-811.
75. Fan Y, Lv C, Zhang X, Zhou G, Zhou Y. The utility challenge of privacy-preserving data-sharing in cross-company defect prediction: an empirical study of the CLIFF & MORPH algorithm. In: Proceedings of the International Conference on Software Maintenance and Evolution; 2017:80-90.

76. Zhang D, Chen X, Cui Z, Ju X. Software defect prediction model sharing under differential privacy. In: Proceedings of the International Conference on Advanced and Trusted Computing; 2018:1547-1554.
77. Amasaki S. On applicability of cross-project defect prediction method for multi-versions projects. In: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering; 2017:93-96.
78. Amasaki S. Cross-version defect prediction using cross-project defect prediction approaches: does it work?. In: Proceedings of the International Conference on Predictive Models and Data Analytics in Software Engineering; 2018:32-41.
79. Bennin KE, Toda K, Kamei Y, Keung J, Monden A, Ubayashi N. Empirical evaluation of cross-release effort-aware defect prediction models. In: Proceedings of the International Conference on Software Quality, Reliability and Security; 2016:214-221.
80. Bowes D, Hall T, Petrić J. Software defect prediction: do different classifiers find the same defects?. *Soft Qual J.* 2017;26(2):525-552.
81. Watanabe S, Kaiya H, Kaijiri K. Adapting a fault prediction model to allow inter language reuse. In: Proceedings of the International Conference on Predictive Models in Software Engineering; 2008:19-24.
82. D'Ambros M, Lanza M, Robbes R. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empir Softw Eng.* 2012;17(4-5):531-577.
83. Wu R, Zhang H, Kim S, Cheung SC. Relink: recovering links between bugs and changes. In: Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering; 2011:15-25.
84. Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. *IEEE Trans Softw Eng.* 2007;33(1):2-13.
85. Shepperd M, Song Q, Sun Z, Mair C. Data quality: some comments on the nasa software defect datasets. *IEEE Trans Softw Eng.* 2013;39(1):1208-1215.
86. Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the International Conference on Software Engineering; 2016:321-332.
87. Dietterich TG. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Comput.* 1998;10(7):1895-1923.
88. Jelihovschi EG, Faria JC, Allaman IB. ScottKnott: a package for performing the Scott-Knott clustering algorithm in R. *TEMA (São Carlos).* 2014;15(1):3-17.
89. Ghotra B, McIntosh S, Hassan AE. Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proceedings of the International Conference on Software Engineering; 2015:789-800.
90. Benjamini Y, Hochberg Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J R Stat Soc Ser B Methodol* 1995;57(1):289-300.
91. Petrić J, Bowes D, Hall T, Christianson B, Baddoo N. Building an ensemble for software defect prediction based on diversity selection. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement; 2016:46.
92. Śliwowski J, Zimmermann T, Zeller A. When do changes induce fixes?. In: Proceedings of the International Workshop on Mining Software Repositories; 2005:1-5.
93. Chen X, Zhao Y, Wang Q, Yuan Z. MULTI: Multi-objective effort-aware just-in-time software defect prediction. *Inf Softw Technol.* 2018;93:1-13.
94. Huang Q, Xia X, Lo D. Supervised vs unsupervised models: a holistic look at effort-aware just-in-time defect prediction. In: Proceedings of the International Conference on Software Maintenance and Evolution; 2017:159-170.

How to cite this article: Chen X, Mu Y, Qu Y, et al. Do different cross-project defect prediction methods identify the same defective modules?. *J Softw Evol Proc.* 2020;32:e2234. <https://doi.org/10.1002/smr.2234>