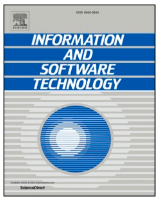
ARTICLE IN PRESS

Information and Software Technology xxx (xxxx) xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Revisiting heterogeneous defect prediction methods: How far are we?

Xiang Chen [*,a,b], Yanzhou Mu [c], Ke Liu [a], Zhanqi Cui [d], Chao Ni [e]

[a] School of Information Science and Technology, Nantong University, Nantong, China
[b] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin, China
[c] College of Intelligence and Computing, Tianjin University, Tianjin, China
[d] Computer School, Beijing Information Science and Technology University, Beijing, China
[e] School of Software Technology, Zhejiang University, Ningbo, China

ARTICLE INFO

Keywords:
Software defect prediction
Heterogeneous defect prediction
Unsupervised defect prediction
Non-effort-aware performance indicators
Effort-aware performance indicators
Diversity analysis
Empirical studies

ABSTRACT

Context: Cross-project defect prediction applies to the scenarios that the target projects are new projects. Most of the previous studies tried to utilize the training data from other projects (i.e., the source projects). However, metrics used by practitioners to measure the extracted program modules from different projects may not be the same, and performing heterogeneous defect prediction (HDP) is challenging.

Objective: Researchers have proposed many novel HDP methods with promising performance until now. Recently, unsupervised defect prediction (UDP) methods have received more attention and show competitive performance. However, to our best knowledge, whether HDP methods can perform significantly better than UDP methods has not yet been thoroughly investigated.

Method: In this article, we perform a comparative study to have a holistic look at this issue. Specifically, we compare five HDP methods with four UDP methods on 34 projects in five groups under the same experimental setup from three different perspectives: non-effort-aware performance indicators (NPIs), effort-aware performance indicators (EPIs) and diversity analysis on identifying defective modules.

Result: We have the following findings: (1) HDP methods do not perform significantly better than some of UDP methods in terms of two NPIs and four EPIs. (2) According to two satisfactory criteria recommended by previous studies, the satisfactory ratio of existing HDP methods is pessimistic. (3) The diversity of prediction for defective modules across HDP *vs.* UDP methods is more than that within HDP methods or UDP methods.

Conclusion: The above findings implicate there is still a long way for the HDP issue to go. Given this, we present some observations about the road ahead for HDP.

1. Introduction

Software defect prediction (SDP) [1] is an active research topic in software repository mining domain. SDP methods can be used to optimize the software quality assurance resource allocation by predicting potential defective program modules in advance. Most of the previous studies focus on within-project defect prediction (WPDP), which constructs the SDP models and then predicts defective modules within the same project. However, in practice, the target projects for SDP are often new start-up projects or only have a few labeled data in most cases. In previous studies [2], most of the researchers mainly utilize the training data collected from other projects (i.e., the source projects) and resort to transfer learning [3] to alleviate the data distribution difference between different projects. This problem is called as cross-project defect prediction (CPDP).

Most of the previous CPDP studies assume that both the source project and target project use the same metrics to measure the extracted program modules (i.e., homogeneous data). However, practitioners may use different metrics to measure the extracted program modules. The reasons can be summarized as follows: First, some metrics are designed based on a specific programming language. Therefore these metrics cannot be used to measure modules written by other programming languages. Second, some metrics can be supported only by commercial tools (such as Understand tool[1]), and the cost of buying these

* Corresponding author at: School of Information Science and Technology, Nantong University, Nantong, China.
E-mail addresses: xchencs@ntu.edu.cn (X. Chen), myz_2019218009@tju.edu.cn (Y. Mu), l51415370@163.com (K. Liu), czq@bistu.edu.cn (Z. Cui), jacknichao920209@gmail.com (C. Ni).

[1] https://scitools.com/

https://doi.org/10.1016/j.infsof.2020.106441
Received 27 February 2020; Received in revised form 6 August 2020; Accepted 26 September 2020
Available online 30 September 2020
0950-5849/© 2020 Elsevier B.V. All rights reserved.

commercial tools cannot be afforded for small-scale start-up enterprises [4]. Compared to CPDP with homogeneous metrics, performing CPDP with heterogeneous metrics is more challenging, and this problem is called as heterogeneous defect prediction (HDP) [5,6].

Motivation. Until now, researchers have proposed many novel supervised HDP methods. For example, Nam et al. [4,5] proposed a HDP method based on metric selection and metric mapping. Li et al. [7–9] proposed a set of novel HDP methods based on kernel correlation alignment and ensemble learning to solve the linearly inseparable problem and the class imbalanced problem. Recently, we notice some studies [10–13] have shown the competitiveness of unsupervised defect prediction (UDP) methods for CPDP.

Given the target project, UDP methods can directly identify the defective modules, while HDP methods can identify the defective modules via the models constructed on the source project with heterogeneous metrics. Therefore, we can make a fair comparison between HDP and UDP methods. However, to the best of our knowledge, this issue has not been thoroughly investigated in previous HDP studies [4,5, 7–9]. In addition, most of the existing studies often evaluate the performance of HDP methods in terms of non-effort-aware performance indicators, such as *F1, AUC*. This will result in the bias in previous empirical conclusions, and the performance comparison results in terms of other performance indicators (such as indicators considering testing efforts) are still unknown. Finally, there exists a lack of comprehensive comparison between different HDP methods, especially some recently proposed methods [7–9].

In our empirical studies, we choose state-of-the-art HDP methods and UDP methods proposed from 2014 to 2019 to make a thorough comparison. In particular, the five chosen HDP methods include a metric selection and matching based method [4,5], three metric transformation-based methods [7–9], and one distribution characteristics based method [14]. The four chosen UDP methods include the connection based method [11], the methods CLA and CLAMI [12], and the ranking-based method [10]. Then we choose five groups of datasets (i.e., AEEEM, ReLink, PROMISE, NASA, and SOFTLAB). These datasets are collected from 34 different projects from the open-source community and commercial enterprises, and they have been widely used in previous HDP studies [4,5,7–9].

To systematically investigate this issue, we aim to answer the following three research questions.

RQ1: Do these HDP methods perform significantly better than existing UDP methods in terms of non-effort-aware performance indicators (NPIs)?

For RQ1, we mainly consider two NPIs (i.e., *F1* and *AUC*) and perform statistic significance test based on the Wilcoxon signed-rank test and Cliff's δ [15]. Final results show HDP methods cannot perform significantly better than all the UDP methods in terms of these two NPIs. Moreover, according to two satisfactory criteria recommended by previous CPDP studies [16,17], the satisfactory ratio of existing HDP methods is still pessimistic.

RQ2: Do these HDP methods perform significantly better than existing UDP methods in terms of effort-aware performance indicators (EPIs)?

Different from NPIs, EPIs consider the efforts used for inspecting the program modules. For RQ2, we mainly consider four EPIs. In particular, the first two EPIs are P_{opt} and *ACC*, which were previously used for just-in-time (JIT) defect prediction [18–20]. Results on these two indicators show the simple unsupervised methods proposed by Zhou et al. [10] can still achieve better performance than state-of-the-art HDP methods. The latter two EPIs are *PMI@20%* and *IFA*, which were recently proposed by Huang et al. [21,22] to revisit previous studies on JIT defect prediction. Results on these two indicators show that using the unsupervised methods [10] needs to inspect more program modules when available resources are limited. Moreover, using these simple methods means developers need to inspect many non-defective modules before encountering the first real defective module, which may hurt developers' confidence and patience.

RQ3: Do HDP methods and UDP methods identify the same defective modules?

For RQ3, we can compute the complementary degree of different HDP and UDP methods in identifying defective modules. Bowes et al. [23] performed a sensitivity analysis by using diagrams to analyze whether there is a difference between four classical classifiers in terms of the specific defects each detects and does not detect. However, in our study, we consider five HDP methods and four UDP methods (i.e., there are far more than four methods to compare with each other). Therefore, it is difficult to achieve satisfactory visualization results by using the visualization methods used by Bowes et al. [23]. In our study, we use McNemar's test [24] to perform a diversity analysis on identifying defective modules for different methods. Final results show that the diversity of prediction for defective modules across HDP *vs.* UDP methods is more than that within HDP methods or UDP methods. This shows these two different kinds of methods are complementary to each other, and ensemble learning [25] is a possible way for further HDP studies. Moreover, we also find a certain number of defective modules, which cannot be correctly identified by either method or even both kinds of methods. These findings implicate there still exist bottlenecks in current HDP studies.

Contributions. To our best knowledge, the main contributions of our article can be summarized as follows.

- We design and conduct large-scale empirical studies to systematically compare HDP methods with UDP methods on 34 projects in five groups under the same experimental setup.
- We make a comprehensive comparison between HDP methods and UDP methods from three different perspectives: NPIs, EPIs, and diversity analysis on identifying defective modules.
- Empirical results show there is still a long way for HDP methods to go, and we present some observations about the road ahead for HDP.

Article organization. Section 2 analyzes the related work for cross-project defect prediction and motivation of our study. Section 3 introduces the details and experimental setting of our chosen HDP methods and UDP methods. Section 4 shows the experimental setup, including research questions, experimental subjects, and performance indicators. Section 5 performs result analysis for three research questions. Section 6 presents some observations about the road ahead for HDP. Section 7 discusses the main threats of our empirical studies. Section 8 concludes this article.

2. Related work and research motivation

In this section, we first summarize the related work for heterogeneous defect prediction. Then, we show the motivations of our study.

2.1. Related work for cross-project defect prediction

When applying SDP [26–29] to real software development, the target projects may be new start-up projects or have less labeled modules. That means these projects do not have sufficient historical training data to construct high-quality prediction models. A simple and straightforward solution is to use the training data collected from other projects directly. However, application domain, utilized programming languages, development process, developers' experience may not be the same for different projects. This difference will result in the non-negligible data distribution difference in most cases and result in poor CPDP performance. For example, Zimmermann et al. [16] and He et al. [17] verified the unsatisfactory performance of CPDP based on real projects from open-source communities and Microsoft corporation.

CPDP is an active research problem in current SDP studies. In the early stage of CPDP, the researchers mainly used the supervised methods (i.e., they used the source project modules to construct model). These methods can be further classified into two categories: supervised homogeneous CPDP methods and supervised heterogeneous CPDP methods (i.e., HDP methods) based on whether the source project and the target project use the same metric set. For the former, the researchers designed novel methods by using metric value transformation, module selection and weight setting, metric mapping, and selection, ensemble learning, class imbalance learning. For the latter, the issue is more challenging, and the detailed related work analysis can be found in the rest of this section. Later, researchers found using the modules of the source project and some labeled programs in the target project can improve the performance of CPDP. These methods are called semi-supervised methods [30]. The researchers often aimed to identify some representative modules in the target project and labeled them manually. These methods are designed mainly based on ensemble learning and TrAdaBoost [31]. Researchers recently found that most of the gathered SDP datasets satisfy the assumption (i.e., the metric value of defective modules tends to be higher than the metric value of non-defective modules). Based on this assumption, they proposed some unsupervised methods [10–12], and these methods can achieve promising results when compared to supervised methods. Since the unsupervised methods do not need any training data, the distribution difference between the source and the target project is no longer an issue, and the unsupervised methods also provide a new way to solve the CPDP problem.

Since HDP is the issue concerned by this article, the related work for other categories can be found in a recent survey [2,10]. It is not hard to find that the HDP problem is more challenging than supervised homogeneous CPDP problem. A simple solution is to use the common metrics used by both the source project and the target project. However, first, the number of common metrics may be very small (sometimes even zero). Second, some informative metrics that help to construct high-quality models may not exist in the common metrics. Third, finding other projects using the same metrics is a challenging task. He et al. [14] proposed a distribution characteristic based HDP method, which considers 16 indicators based on distribution characteristics. Nam et al. [4,5] proposed the HDP method that includes the metric selection phase and the metric matching phase. Then Yu et al. [32] proposed a method FMT. FMT method used feature subset selection to search for the optimal feature subset in the source project. Then it got the distribution curves of all the features in the target project. Finally, it designed a feature mapping method to covert heterogeneous features into the matched features based on the distance of different distribution curves. Jing et al. [6] proposed a method CCA+. They use UMR (unified metric representation) for the source project and the target project. Then they used CCA (canonical correlation analysis) to make the data distribution more similar. Cheng et al. [33] proposed a method CCT-SVM. CCT-SVM took different misclassification costs when constructing the models via support vector machine to alleviate the impact of class imbalanced data. Li et al. [7] proposed a method CTKCCA. This method can not only make the data distribution more similar but also utilize the different misclassification costs for defective and non-defective modules to alleviate the class imbalanced problem. Meanwhile, they [8] proposed a novel method EMKCA. This method first mapped the source project and the target project data into a high dimensional kernel space through multiple kernel learning. Then, it designed a kernel correlation alignment method to further reduce the data distribution in the kernel space. Finally, it incorporated multiple kernel classifiers via ensemble learning to alleviate the class imbalanced problem. Recently, they [9] extended the previous study [8] and proposed a two-stage ensemble learning based approach. The ensemble multi-kernel domain adaptation stage constructed constructed the EMKCA predictor, which combines the advantage of multiple kernel learning and domain adaptation techniques. In ensemble data sampling stage, it employed RES (RESample with replacement) technique to learn different EMKCA predictors and used average ensemble to combine them. Since there exist multiple candidate source projects, Li et al. [34] proposed a method MSMDA. This method can incrementally select distribution-similar source projects for a given target project. Moreover, they designed a sparse representation based double obfuscation algorithm to protect the privacy of dataset owners. Li et al. [35] proposed a method CLSIP. This method exploited the mixed data (i.e., combine the heterogeneous source and target project data) and aimed to transform the source data to the target subspace, where the data distributions of source and target projects become similar, and the structure of source data can be maintained. Then it used different misclassification costs for defective and non-defective modules in the domain adaptation stage to alleviate the class imbalanced problem.

2.2. *Motivations of our empirical study*

After analyzing the experimental setup in previous HDP studies, we summarize the analysis results in Table 1. The column **Dataset** denotes the selected empirical subjects, the column **UDP** denotes whether UDP methods are considered as baselines, the column **NPI** denotes whether NPIs are used to evaluate the model performance, the column **EPI** denotes whether EPIs are used to evaluate the model performance, the column **Diversity** denotes whether diversity analysis is used to analyze the defective modules identified by different methods. Based on Table 1, we analyze the three motivations of our study as follows:

Motivation 1: Lack of comparison with unsupervised methods. Recent studies [10–12] have shown the competitiveness of UDP methods for CPDP. However, whether existing HDP can perform significantly better than these UDP methods has not been thoroughly investigated. In Table 1, we find only one work [4] regards the unsupervised method as the baseline, and only one unsupervised method [11] is considered in this work. We also find in a recent study [10], Zhou et al. compared their proposed methods ManualDown and ManualUp with almost all the existing CPDP methods (including HDP methods [5,6,33]). However, first, they only used the prediction performance reported in the original literature. Therefore, this kind of comparison is unfair since the comparison did not be performed under the same experimental setup. Second, they did not show the comparison results in terms of all the performance indicators considered in our study. Finally, their study did not consider recently proposed HDP methods [7–9].

Motivation 2: Lack of comprehensive performance evaluation in terms of different performance indicators. Existing studies often evaluated the performance of HDP methods only in terms of a type of performance indicators. This will lead to biased conclusions of previous HDP empirical studies. For example, when analyzing the NPIs used in previous HDP studies, we find among nine studies, five studies only considered threshold-independent NPI (i.e., *AUC*), two studies only examined threshold-dependent NPIs (such as *F1, balance, MCC*), and only two studies [7,9] considered both types of NPIs. When analyzing

Table 1
Analysis of experimental setup for previous HDP studies.

Reference	Dataset	UDP	NPI	EPI	Diversity
He et al. [14]	PROMISE, ReLink, AEEEM	×	✓	×	×
Jing et al. [6]	PROMISE, Relink, AEEEM, NASA, SOFTLAB	×	✓	×	×
Cheng et al. [33]	Relink, AEEEM, NASA, SOFTLAB	×	✓	×	×
Nam et al. [5]	PROMISE, Relink, AEEEM, NASA, SOFTLAB	×	✓	×	×
Nam et al. [4]	PROMISE, Relink, AEEEM, NASA, SOFTLAB	✓	✓	×	×
Yu et al. [32]	PROMISE, NASA	×	✓	×	×
Li et al. [7]	PROMISE, Relink, AEEEM, NASA, SOFTLAB	×	✓	×	×
Li et al. [8]	PROMISE, Relink, AEEEM, NASA, SOFTLAB	×	✓	×	×
Li et al. [9]	PROMISE, Relink, AEEEM, NASA, SOFTLAB	×	✓	×	×

the EPIs, we find none of the previous studies used these performance indicators to evaluate the constructed HDP models. Moreover, in our previous study [13], we performed diversity analysis via the McNemar test to investigate whether different homogeneous CPDP methods can identify the same defective modules. However, we find this kind of analysis method has not been used in previous HDP studies.

Motivation 3: Lack of comprehensive comparison between different HDP methods. Since these methods [5,6,14] are the first three proposed methods in the HDP research issue, they are frequently chosen as baselines. For example, the method proposed by He et al. [14] is selected as the baseline in four studies, the method proposed by Nam and Kim [5] is chosen as the baseline in five studies, and the method proposed by Jing et al. [6] is selected as the baseline in five studies. However, the comparison results between all the HDP methods in the same experimental setup are still unknown.

Based on the above motivations, we choose state-of-the-art HDP methods and UDP methods proposed from 2014 to 2019. The comparisons are systematically conducted from three perspectives: NPIs, EPIs, and diversity analysis on identifying defective modules. Moreover, we choose all the experimental subjects used in previous HDP studies. Final results implicate studies on the heterogeneous defect prediction still have a long way to go.

3. Methods

3.1. HDP methods

In our study, we mainly investigate the following five HDP methods, since these HDP methods have been published in refereed conferences or journals in the software engineering research domain (such as TSE, ASEJ, ESEC/FSE, ICSME), and these methods have been proposed in the recent five years. Some of the HDP methods analyzed in related work are not considered in our study, and the reasons can be summarized as follows. The method MSMDA [34] mainly focused on candidate source project selection and privacy protection for dataset owners. The method CLSUP [35] mainly focused on mixed data (i.e., the heterogeneous source and target project data are combined). It is not hard to find these two methods do not have the same experimental setup in our study since our study only concerns the HDP methods, which construct models based solely on the data of the source project and perform predictions only on the target project.

In the rest of this subsection, we will introduce the details of these HDP methods and experimental setup in our empirical studies.

HDP1. This method is proposed by Nam et al. [4,5]. It first uses metric selection to identify and remove redundant and irrelevant features in the source project. Later, it measures the distribution similarity of each source and target metric pair. Then it uses the cutoff threshold to remove poorly matched metrics. After the metric mapping phase, it uses the maximum weighted bipartite matching technique to select a group of matched metrics. Finally, it uses a classifier to build a prediction model using a source dataset with selected and matched metrics. This model can then be used to predict defects on a target dataset with metrics matched to the chosen source metrics.

In our study, the experimental setup for HDP1 follows the suggestions by Nam et al. [4,5]. Specifically, we utilize a gain ratio based feature selection method and select the top 15% features. Then we choose the Kolmogorov-Smirnov test to measure the distribution similarity with the cutoff threshold of 0.05 as the maximum weighted bipartite matching technique to select the best suitable group of matched metrics. Finally, we use Logistic regression as a classifier.

HDP2. This is EMKCA (ensemble multiple kernel correlation alignment) based method, and it is proposed by Li et al. [8]. It first maps the data in the source project and the data in the target project into a high dimensional kernel space through multiple kernel learning. Therefore, defective modules and non-defective modules in the high dimensional kernel space can be better separated. Later, it utilizes a kernel correlation alignment method to further reduce the data distribution of the source and target projects in the kernel space. Finally, it integrates multiple kernel classifiers with ensemble learning to alleviate the class imbalanced problem.

In our study, the experimental setup for HDP2 follows the suggestions by Li et al. [8]. In particular, we use ten base kernels $k(x_i, x_j)$, which include nine Gaussian kernels $e^{-\|x_i-x_j\|/2\sigma^2}$ with different σ[2] in $\{2^{-4}, 2^{-3}, 2^{-2}, \dots, 2^3, 2^4\}$ and a linear kernel on all the metrics. Moreover, we set the value of the parameter r[3] in ICD (Incomplete Cholesky decomposition) [36] to 60. Here ICD is a matrix decomposition method used to reduce the kernel matrix. Finally, we use Logistic regression as a classifier.

HDP3. HDP3 is CTKCCA (cost-sensitive transfer kernel canonical correlation analysis) based method, and it is proposed by Li et al. [7]. In particular, it first employs z-score normalization to preprocess data. It uses transfer kernel canonical correlation analysis to derive the nonlinear feature space and uses ICD to reduce the learned kernel matrix. Finally, it utilizes a cost-sensitive learning technique, which considers the different costs for misclassifying defective modules and non-defective modules. For HDP, the misclassification cost of predicting defective modules as non-defective modules is much higher than the misclassification cost of predicting non-defective modules as defective modules.

In our study, the experimental setup for HDP3 follows the suggestions by Li et al. [7]. In particular, we first use the Gaussian kernel function $k(x_i, x_j) = e^{-\|x_i-x_j\|/2\sigma^2}$ and set the kernel parameter σ to the inverse of mean of square distance for the corresponding source project and target project respectively. Second, the misclassification costs in CTKCCA are adaptively set from 1 to $\#nondefective/\#defective$, where $\#$ *nondefective* and $\#$*defective* denote the number of non-defective modules and defective modules respectively in the source project. Later, we set the value of the parameter r in ICD to 70. Finally, we use Logistic regression as a classifier.

HDP4. HDP4 is TSEL (two-stage ensemble learning) based method, and it is proposed by Li et al. [9]. In particular, this approach contains two phases: ensemble multi-kernel domain adaptation (EMDA) phase and ensemble data sampling (EDS) phase. In the EMDA phase, it proposes an ensemble multiple kernel correlation alignment (EMKCA) predictor, which combines the advantage of multiple kernel learning and domain adaptation techniques. The EDS phase employs resample with replacement technique to learn multiple EMKCA predictors and uses average ensemble to combine them. After these two phases, it has an ensemble of defect predictors, and it can be used to predict defective modules in the target project.

In our study, the experimental setup for HDP4 follows the suggestions by Li et al. [9]. In particular, there are three parameters in HDP4. The first parameter N is used to choose the number of EMKCA predictors, and the value of this parameter is set to 10. The second parameter M is used to decide the number of base kernels and the value of this parameter is set to 10 (i.e., 9 Gaussian kernels $e^{-\|x_i-x_j\|/2\sigma^2}$ with different σ in $\{2^{-4}, 2^{-3}, 2^{-2}, \dots, 2^3, 2^4\}$ and a linear kernel are used as the base kernels). The third parameter r is used in ICD, and the value of this parameter is set to 60. Later, HDP4 uses a Logistic regression classifier to construct prediction models.

HDP5. HDP5 is the first HDP method proposed in 2014. It is a distribution characteristic based method, and this method is called CPDP-IFS. This method is proposed by He et al. [14]. This method considers 16 indicators based on distribution characteristics, including mode, median, mean, harmonic mean, minimum, maximum, range, variation ratio, interquartile range, variance, standard deviation, coefficient of variation, skewness, and kurtosis. Then it employs these distributional

[2] σ is the parameter for the Gaussian kernel.

[3] r determines the rank of the approximate matrix.

characteristic vectors of each program module as new metrics to build HDP models via Logistic regression classifier and predicts defective modules in the target project.

3.2. UDP methods

In our article, we mainly consider four state-of-the-art UDP methods. These UDP methods have also been published in refereed top-tier conferences or journals in the software engineering research domain (such as TOSEM, ASE, ICSE), and these methods have also been proposed in recent five years.

UDP1 and UDP2. Nam and Kim [12] proposed two methods CLA (UDP1) and CLAMI (UDP2). The key idea of these two UDP methods is to label the unlabeled dataset by using the magnitude of metric values. Hence, CLA and CLAMI have the advantages of an automated manner, and no manual efforts required. The first two phases of these two methods are (1) clustering unlabeled modules, and (2) labeling these modules in clusters. CLA only consisted of these two phases. CLAMI has two additional phases to generate the training dataset from the target project's dataset. (3) metric selection and (4) instance selection, since these two phases can be used to improve the quality of the datasets further. Then CLAMI used a Logistic regression classifier to construct a model. Finally, (5) they also selected metrics from the target project's dataset with the same set of metrics selected by the phase (3) as the test set and use the constructed model to perform defect prediction on the test set. Since CLA may have violated metrics and instances, this may cause some instances are incorrectly labeled. While CLAMI can apply metric selection and instance selection to remove these violated metrics and instances, therefore, the constructed model based on the pre-processed training data can help to achieve better performance. In our study, a cutoff threshold used in the first phase is set to the 50th percentile, and this cutoff threshold is used to identify higher metric values.

UDP3. Zhang et al. [11] proposed a connectivity-based UDP method via spectral clustering. This method first uses z-score to normalize the value of each metric, Then it uses three steps for spectral clustering: (1) the first step is to calculate the Laplacian matrix L_{sym}, (2) the second step is to perform the eigendecomposition on the matrix L_{sym}, (3) the third step is to divide all the modules into two clusters (the defective cluster and the non-defective cluster). This method assumes that defective modules generally have larger values than non-defective modules for most metrics. Based on this assumption, it uses the average row sums of the normalized metrics of each cluster. The cluster with a larger average row sum is classified as the defective cluster, and another cluster is classified as the non-defective cluster.

UDP4. Zhou et al. [10] proposed simple unsupervised methods (i.e., ManualDown and ManualUp) based on module size. These methods can be easily implemented, and empirical results surprisedly show these methods have a good performance. In particular, ManualDown assumes a module with larger LOC (lines of code) is more defect-prone, as a larger module tends to have more defects, while ManualUp assumes a module with smaller LOC is more defect-prone, as a smaller module is proportionally more defect-prone and hence should be first inspected. In our study, the experimental setup for UDP4 follows the suggestions by Zhou et al. [10]. First, for five groups of datasets (introduced in Section 4), we choose the following metric as the metric LOC: LOC_EXECUTABLE (The number of lines of executable code for a module) metric in the NASA dataset, ck_oo_numberOfLinesOfCode (Number of Lines of code) metric in the AEEEM dataset, executable_loc (The number of lines of executable code for am module) metric in the SOFTLAB dataset, CountLineCode (Number of lines of code) metric in the ReLink dataset and loc (Number of lines of code) metric in the PROMISE dataset. Second, in terms of NPIs, we utilize the ManualDown method as UDP4, since the to-ranked modules by using this method may contain modules with high defect density. On the contrary, in terms of EPIs, we utilize the ManualUp method as UDP4. Finally, we classified the top 50% modules as defective and the remaining 50% as non-defective. Notice the strategies used by these simple methods are first investigated by Menzies et al. [37] based on Koru et al.'s theory of relative defect proneness [38].

4. Experimental setup

4.1. Research questions

In our empirical studies, we want to investigate the following three research questions.

RQ1: Do these HDP methods perform significantly better than existing UDP methods in terms of NPIs?

RQ2: Do these HDP methods perform significantly better than existing UDP methods in terms of EPIS?

RQ3: How about the diversity analysis results on identifying defective modules between HDP methods and existing UDP methods?

4.2. Experimental subjects

In our empirical study, we choose publicly available datasets from five different groups (i.e., AEEEM, ReLink, PROMISE, NASA and SOFTLAB). The reasons for selecting these experimental subjects can be summarized as follows: First, the diversity of these datasets from different groups can guarantee the generalization of our empirical results, since the software domain, utilized metrics, the granularity of extracted program modules are different in most of these groups. Second, as shown in Table 1, we consider all the experimental subjects, which have been used in the previous HDP studies [5–9], and the representative of these subjects can be guaranteed. Then we will introduce these five groups of datasets in turn.

AEEEM group. The datasets from this group were gathered by D'Ambros et al. [39]. This group includes five projects: EQ, JDT, LC, ML, and PDE. The granularity of modules is set to file, and 61 metrics are used to measure the modules. Specifically, five metrics are based on previous-defect information, 17 metrics are based on source code complexity, five metrics are based on the entropy of change, 17 metrics are based on the churn of source code, and 17 metrics are based on the entropy of source code.

ReLink group. The datasets from this group were gathered by Wu et al. [40], and the label information was verified and corrected manually. The granularity of modules is set to file. Twenty-Six metrics are used to measure the modules via Understand tool. This group includes three data sets: Apache, Safe, and ZXing.

PROMISE group. The datasets from this group were gathered by Jureczko and Madeyski [41] via two tools: BugInfo and CKJM. The granularity of modules is set to class. Twenty metrics based on code complexity are used to measure the modules. A description of these metrics can be found in the study of Jureczko and Madeyski [41].

NASA group. The datasets from this group were originally gathered by Menzies et al. [42] and then were cleaned by Shepperd et al. [43] to improve dataset quality. The granularity of modules is set to function/method. Each project in NASA represents a NASA software system or sub-system, which contains the corresponding defect-marking data and various static code metrics (such as size, readability, complexity features, and so on).

SOFTLAB group. The datasets from this group were gathered by Turhan et al. [44]. It consists of five projects, which are embedded controller software for white goods. The extracted modules are measured by 29 metrics. This group includes five projects (i.e., from AR1 to AR6 except for AR2) from commercial enterprises.

The statistics of these datasets from different groups can be found in Table 2. Table 2 lists the name of the group, the name of the dataset, the number of program modules, and the number (percentage) of defective modules. As analyzed in the previous studies [6,8], datasets from different groups (i.e., gathered by various researchers) often consider different metrics in most cases. For example, the NASA dataset has 21 to

Table 2
The statistics of datasets in different groups.

Group	Project name	Modules		# Metrics	Granularity
		# Modules	# (%) Defective modules		
AEEEM	EQ	324	129(39.81%)	61	Class
	JDT	997	206(20.66%)		
	LC	691	64(9.26%)		
	ML	1862	245(13.16%)		
	PDE	1497	209(14.01%)		
ReLink	Apache	194	98(50.52%)	26	File
	Safe	56	22(39.29%)		
	Zxing	399	118(29.57%)		
PROMISE	ant-1.3	125	20(16.00%)	20	Class
	arc	234	27(11.54%)		
	camel-1.0	339	13(3.83%)		
	poi-1.5	237	141(59.49%)		
	redaktor	176	27(15.34%)		
	skarbonka	45	9(20.00%)		
	tomcat	858	77(8.97%)		
	velocity-1.4	196	147(75.00%)		
	xalan-2.4	723	110(15.21%)		
	xerces-1.2	440	71(16.14%)		
NASA	cm1	344	42(12.21%)	37	Function
	mw1	264	27(10.23%)		
	pc1	759	61(8.04%)		
	pc3	1125	140(12.44%)		
	pc4	1399	178(12.71%)		
	jm1	9593	1759(18.34%)	21	
	pc2	1585	16(1.01%)	36	
	pc5	17001	503(2.96%)	38	
	mc1	9277	68(0.73%)		
	mc2	127	44(34.65%)	39	
	kc3	200	36(18.00%)		
SOFTLAB	ar1	121	9(7.44%)	29	Function
	ar3	63	8(12.70%)		
	ar4	107	20(18.69%)		
	ar5	36	8(22.22%)		
	ar6	101	15(14.85%)		

37 metrics, and the AEEEM dataset has 61 metrics. However, there is only one common metric LOC (lines of codes) used by bot the NASA dataset and the AEEEM dataset. Therefore, performing CPDP with heterogeneous metric sets is a common problem in practice.

Since we only focus on defect prediction across projects with heterogeneous metric sets, we do not conduct defect prediction across projects with the same metrics. For example, if we choose the project EQ in AEEEM datasets as the target project, there are 29 (=3+10+11+5) HDP combinations when considering the projects in other groups. Therefore, there are 876 ($=3 \times 31 + 5 \times 29 + 5 \times 29 + 10 \times 24 + 11 \times 23$) HDP combinations. Notice some NASA datasets do not have the same metrics (As shown in Table 2, there are five subgroups in the NASA group, each considers 37, 21, 36, 38, and 39 metrics respectively). Therefore we also conduct HDP between NASA datasets with different metrics, and there are 86 ($=5 \times 6 + 1 \times 10 + 1 \times 10 + 2 \times 9 + 2 \times 9$) HDP combinations. In summary, we have 962 HDP combinations in our empirical study when considering these 34 datasets.

Notice we do not compare HDP methods with within-project defect prediction methods in our study since this kind of comparison has been widely investigated in previous HDP studies. Most of the empirical results show their proposed HDP methods can obtain comparable prediction performances when compared to WPDP methods [4,5,8]. Therefore, we do not use 2-fold cross-validation to evaluate the performance of different methods. In our study, for HDP methods, we directly utilize the data in the source project as the training data and use the data in the target project as the test data. While UDP methods directly conduct defect predictions on the target project.

4.3. Performance indicators

4.3.1. Non-effort-aware performance indicators

For NPIs, we mainly consider two indicators: *F1* and *AUC*. Notice there are other NPIs (such as $g-measure$, $balance$, $g-mean$) that are used for model performance evaluation in previous CPDP studies [2]. However, *F1* and *AUC* are two most used indicators in previous HDP studies.

In software defect prediction, if we treat defective modules as positive instances and non-defective modules as negative instances, we can classify program modules into four types according to the actual type and the predicted type of these modules. 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 shown in Table 3. For this type of measure, if the predicted defective probability for a new module is larger than a given threshold (0.5 is considered in our study), this module will be classified as the defective module. Otherwise, it will be classified as the non-defective module. In this subsection, we introduce the following three threshold-dependent performance indicators (i.e., *precision, recall* and *F1*) in sequence.

***precision* indicator.** *precision* returns the ratio of the number of defective modules that are correctly classified as defective to the number of modules that are classified as defective. It can be defined as:

$$precision = \frac{TP}{TP + FP} \tag{1}$$

***recall* indicator.** *recall* returns the ratio of the number of defective modules that are correctly classified as defective to the total number of defective modules. It can be defined as:

$$recall = \frac{TP}{TP + FN} \tag{2}$$

***F1* indicator.** There exists a trade-off between *precision* and *recall* in practice. In most cases, a higher value of *precision* means a lower value of *recall* and vice versa. Here we use *F1*, which is the harmonic mean between *precision* and *recall*, to evaluate the performance of the constructed models. It can be defined as:

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{3}$$

***AUC* indicator.** However, previous studies [45,46] found these threshold-dependent performance indicators (such as *F1*) have some problems. First, these performance indicators depend on a previously set threshold value. Second, these indicators are sensitive to the class imbalanced problem, which exists in most of the collected SDP datasets. Recently, researchers are more inclined to use *AUC* (Area Under the receiver operator characteristic Curve) indicator. *AUC* is computed by measuring the area under the curve, which plots the true positive rate against the false positive rate, while varying the value of the threshold to determine whether a module is classified as defective or non-defective. The value range of *AUC* is between 0 and 1. The higher the *AUC* value is, the better the prediction performance of the constructed model is.

4.3.2. Effort-aware performance indicators

For EPIs, we mainly consider four indicators: *ACC*, P_{opt}, *PMI@20%* and *IFA*. These indicators have been used to evaluate effort-aware defect

Table 3
Confusion matrix based on the actual type and the predicted type of program modules.

Actual program type	Predicted program type	
	Defective modules	Non-defective modules
Defective modules	*TP*	*FN*
Non-defective modules	*FP*	*TN*

prediction models [19,21,47,48]. Obviously, these indicators can also be used to evaluate HDP models when taking the effort required to inspect program modules into account.

***ACC* indicator.** *ACC* denotes the recall of defective modules when expending 20% of the entire effort.

***PMI@20%* indicator.** In the context of SDP, *PMI@20%* returns proportion of modules inspected when only having 20% of the entire efforts. Notice we use lines of code (LOC) to measure the effort on the module in this study. A higher *PMI@20%* value means the additional efforts are required due to context switches and additional communication overhead among developers [49].

When only using 20% of the entire efforts, *PMI@20%* and *ACC* can evaluate different methods from two different perspectives. We use a simple example to illustrate the difference between these two indicators. In this example, there are 2000 modules (suppose the total lines of code of these modules are 100,000) in the project, of which 40 modules contain defects. If expending 20% of the entire efforts (i.e., 20,000 lines of code are examined) based on the ranked list by a specific method $m1$, we can only inspect 600 modules, of which ten modules are real defective modules. Then *ACC* of this method $m1$ is $10/40 = 25\%$ and *PMI@20%* of this method $m1$ is $600/2000 = 30\%$.

P_{opt} indicator. P_{opt} is the normalized version of the effort-aware performance indicator. According to the previous study [50], P_{opt} can be formally defined as:

$$P_{opt}(m) = 1 - \frac{area(optimal) - area(m)}{area(optimal) - area(worst)} \quad (4)$$

Here *area*(*m*), *area*(*optimal*) and *area*(*worst*) are the area under the curve corresponding to a proposed method, the optimal method, and the worst method, respectively. For the optimal method, modules are sorted in the descendant order according to their actual defect density. While for the worst method, modules are sorted in the ascendant order according to their actual defect density.

***IFA* indicator.** *IFA* returns the number of initial false alarms encountered before the first real defective module is found. This indicator is inspired by previous work on automatic software fault localization [51]. A higher value of *IFA* means more false positives (i.e., non-defective modules are predicted as defective modules) before detecting the first real defective module and may have a non-ignorable impact on developers' confidence and tolerance for this method [52].

5. Empirical results

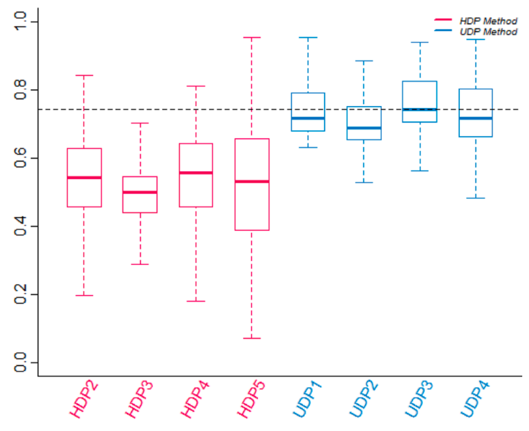
5.1. Results analysis for RQ1

Motivation. First recent studies [10–12] have shown the competitiveness of UDP methods for CPDP. However, whether existing HDP methods can perform significantly better than these UDP methods has not been thoroughly investigated. Second, in a recent study [10], Zhou et al. compared their proposed methods ManualDown and ManualUp with some previously proposed CPDP methods [5,6,33]. However, they only used the prediction performance reported in the original literature, and this kind of comparison is unfair since the comparison was not performed under the same experimental setup. Therefore, in this RQ, we want to compare existing HDP methods with UDP methods in terms of two classical NPIs (i.e., *F1* and *AUC*) in the same experimental setup.

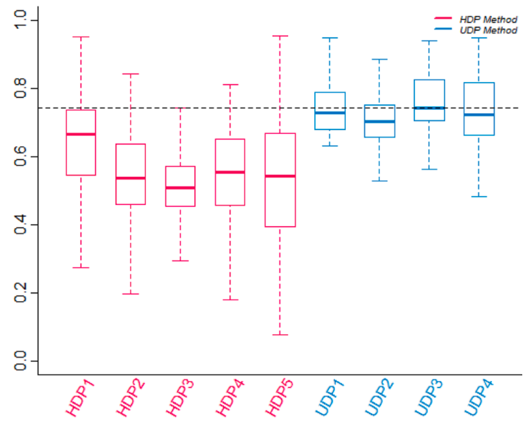
Approach. We use the Wilcoxon signed-rank test[4] at a 95% significance level and Cliff's δ[5] to analyze whether the performance difference between two different methods is statistically significant. Specifically, a method performs significantly better or worse than another method if p value of the Wilcoxon signed-rank test is less than 0.05, and the effectiveness level is not negligible based on Cliff's δ (i.e., $|\delta| \geq 0.147$). Otherwise the difference between two methods is not significant, if p value is not less than 0.05 or p-value is less than 0.05 and the effectiveness level is negligible (i.e., $|\delta| < 0.147$).

Results. Since the method HDP1 cannot succeed on some cross-project defect prediction combinations (i.e., it cannot find a group of matched metrics between the source project and the target project according to the description of this method in Section 3.1). Therefore, we show the results via box plot in two scenarios. In the first scenario (i.e., **Scenario1**), we do not consider the method HDP1, and this scenario will consider the results on all the HDP prediction combinations. In the second scenario (i.e., **Scenario2**), we consider the method HDP1, and this scenario will only consider all the HDP combinations that the method HDP1 can succeed. In our study, the number of all the HDP combinations is 962, and the method HP1 can success in achieve 623 HDP combinations.

The final results in terms of *AUC* in two scenarios on different groups of datasets can be found in Fig. 1. In two subfigures, The blue label denotes UDP methods, and the red label denotes HDP methods. The horizontal dashed line in each subfigure indicates the method's median value with the best performance, which helps visualize the difference



(a) Scenario1



(b) Scenario2

Fig. 1. The results of all the methods via box plot in terms of *AUC* performance indicator.

[4] The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test.

[5] Cliff's δ [15] is a non-parametric effect size measure.

between this method and other methods. From these two subfigures and statistically significant analysis results, we can find all the four UDP methods can perform significantly better than HDP methods in terms of *AUC* performance indicator. Among these four UDP methods, the method UDP4 can achieve the best performance. Moreover, we report win/tie/loss result of comparing HDP methods with UDP methods in terms of *AUC* by grouping results by the target project. Supposing method1 *vs*. method2, "Win" means the number of HDP combinations method1 can perform significantly better than method2. "Tie" means the number of HDP combinations the performance between method1 and method2 has no statistical significance. "Loss" means the number of HDP combinations method1 can perform significantly worse than method2. Since the number of projects is 34, the sum of "Win", "Tie" and "Loss" is 34 for each cell in this table. The comparison results can be found in Table 4. From this table, we can find: (1) In scenario1, UDP methods can win HDP2, HDP3, HDP4 and HDP5 at least 31, 31, 27 and 29 times. (2) In scenario2, UDP methods can win HDP2, HDP3, HDP4 and HDP5 at least 28, 31, 25 and 27. When compared to HDP1, UDP methods except for UDP2 can win at least 21 times. These results show UDP methods can perform significantly better than HDP methods in the majority of cases in terms of *AUC*.

The comparison results in terms of *F1* in two scenarios on different groups of datasets can be found in Fig. 2. From these two subfigures and statistically significant analysis results, we can find all the four UDP methods can significantly perform better than HDP methods in terms of *F1* performance indicator. Among these four UDP methods, the method UDP4 can achieve the best performance. Win/Tie/Loss result of comparing HDP methods and UDP methods in terms of *F1* can be found in Table 5. From this table, we can find: (1) In scenario1, UDP methods can win HDP2, HDP3, HDP4 and HDP5 at least 31, 29, 31 and 27 times. (2) In scenario2, UDP methods can win HDP1, HDP2, HDP3, HDP4 and HDP5 at least 26, 30, 29, 30, 30 and 26 times. These results show UDP methods can perform significantly better than HDP methods in the majority of cases when considering *F1*.

Finally, we want to explore and verify the feasibility of the state-of-the-art HDP and UDP methods investigated in our study through computing a satisfactory ratio of different kinds of HDP methods or UDP methods on a specific group. We mainly consider the following two satisfactory criteria:

SC1. This satisfactory criterion is suggested by Zimmermann et al. [16]. For this criterion, the constructed model is acceptable when the values of two indicators *precision, recall* and *accuracy* are both larger than 75%.

SC2. This satisfactory criterion is suggested by He et al. [17]. For this criterion, the constructed model is acceptable when the values of two indicators *precision* and *recall* are larger than 50% and 70%, respectively. The reason is the class imbalanced problem can make the constructed models hard to achieve high *precision* value [17]. Obviously, the criterion SC2 is weaker than the criterion SC1.

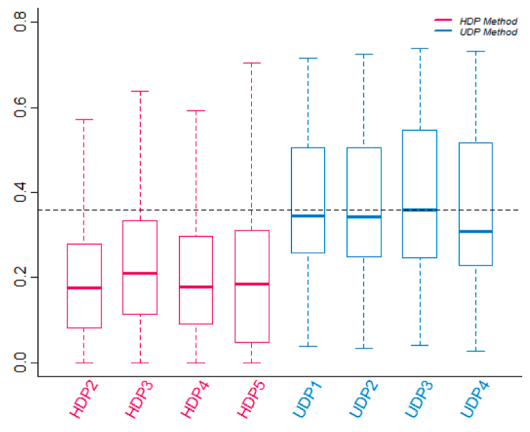
Table 4
Win/Tie/Loss analysis in terms of *AUC* performance indicator.

(a) Scenario1

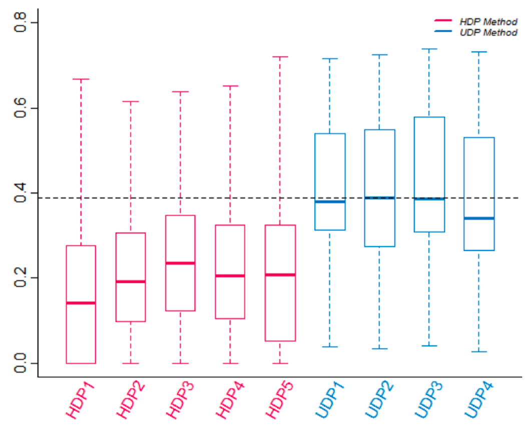
Method Name	UDP1	UDP2	UDP3	UDP4
HDP2	2/1/31	2/1/31	2/1/31	3/0/31
HDP3	3/0/31	2/1/31	2/1/31	3/0/31
HDP4	3/0/31	3/4/27	2/2/30	3/2/29
HDP5	0/2/32	1/4/29	1/1/32	0/4/30

(b) Scenario2

Method Name	UDP1	UDP2	UDP3	UDP4
HDP1	3/10/21	4/19/11	1/11/22	5/7/22
HDP2	2/1/31	2/4/28	2/2/30	3/2/29
HDP3	3/0/31	2/1/31	2/1/31	3/0/31
HDP4	3/1/30	3/6/25	2/3/29	3/3/28
HDP5	0/2/32	1/6/27	1/2/31	0/6/28



(a) Scenario1



(b) Scenario2

Fig. 2. The results of all the methods via box plot in terms of *F1* performance indicator.

Table 5
Win/Tie/Loss analysis in terms of *F1* performance indicator.

(a) Scenario1

Method Name	UDP1	UDP2	UDP3	UDP4
HDP2	0/2/32	0/1/33	1/2/31	0/3/31
HDP3	2/1/31	3/0/31	3/2/29	3/0/31
HDP4	0/2/32	1/2/31	1/2/31	1/2/31
HDP5	0/1/33	0/4/30	1/6/27	2/4/28

(b) Scenario2

Method Name	UDP1	UDP2	UDP3	UDP4
HDP1	0/6/28	1/5/28	0/6/28	1/7/26
HDP2	0/2/32	0/3/31	1/3/30	0/3/31
HDP3	2/1/31	2/2/30	3/2/29	3/0/31
HDP4	0/1/33	0/3/31	1/3/30	0/3/31
HDP5	0/4/30	0/6/28	1/5/28	0/8/26

The final results can be found in Table 6. Here we compute the satisfactory ratio of different methods when the projects in a specific group are set to the target project when considering SC1 or SC2. If we Take ReLink dataset as an example, there are 93 (=(5+10+11+5) × 3)

Table 6
The ratio of HDP methods and UDP methods with acceptable performance on the five groups of datasets with different satisfactory criteria.

Methods	PROMISE		NASA		SoftLab		ReLink		AEEEM	
	SC1	SC2	SC1	SC2	SC1	SC2	SC1	SC2	SC1	SC2
HDP1	0.00%	0.00%	0.00%	0.00%	0.74%	3.68%	0.00%	0.00%	0.00%	0.00%
HDP2	0.00%	0.00%	0.00%	0.00%	0.00%	0.69%	0.00%	5.38%	0.00%	0.00%
HDP3	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
HDP4	0.00%	0.42%	0.00%	0.00%	0.00%	0.00%	0.00%	9.68%	0.00%	0.00%
HDP5	0.00%	3.75%	0.00%	0.00%	0.00%	2.76%	0.00%	20.43%	0.00%	0.00%
UDP1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	66.67%	0.00%	0.00%
UDP2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	66.67%	0.00%	20.00%
UDP3	0.00%	10.00%	0.00%	0.00%	0.00%	20.00%	0.00%	0.00%	0.00%	0.00%
UDP4	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	66.67%	0.00%	20.00%

HDP combinations when considering the HDP4 method, if this method can achieve satisfactory performance in nine pairs, then the satisfactory ratio is 9.68% (=9/93). Notice, the number of considered HDP combinations may less than 31 when analyzing the method HDP1 since HDP1 cannot succeed in some of HDP combinations. When considering SC1, all the HDP methods cannot achieve satisfactory performance when the projects of PROMISE, NASA, Relink, AEEEM are set to the target projects. When considering SC2, all the HDP methods cannot achieve satisfactory performance when the projects of NASA and AEEEM are set to the target projects. Therefore, we conclude that the performance of HDP methods (even UDP methods) is still unsatisfactory, and designing more effective HDP methods is still a challenging task.

Summary for RQ1: In terms of NPIs, existing HDP methods cannot perform significantly better than all the UDP methods in most cases. Moreover, the performance of current HDP methods (even UDP methods) is still unsatisfactory when considering two satisfactory criteria suggested by previous CPDP studies.

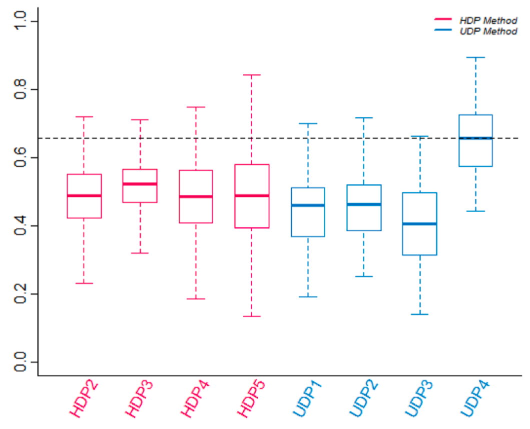
5.2. Results analysis for RQ2

Motivation. EPIs have been investigated in just-in-time defect prediction [19–21,53] and is an active topic in the past three years. However, as discussed in Section 2.2, we find none of the previous HDP studies use EPIs to evaluated the constructed HDP models. Therefore, whether previous empirical results in Just-in-time defect prediction still hold in the HDP issue is unknown. In this RQ, we want to compare existing HDP methods with UDP methods in terms of four EPIs in the same experimental setup.

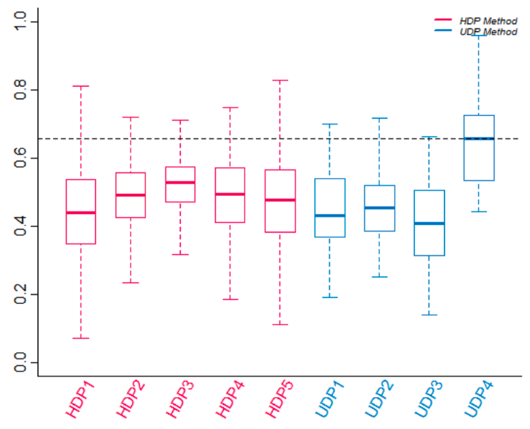
Approach. To answer this RQ, we also use box plot to show all the results of the HDP and UDP methods in terms of a specific EPI. Then we perform statistically significant analysis based on the Wilcoxon signed-rank test and Cliff's δ [15].

Results. The results of all the methods in terms of P_{opt} can be found in Fig. 3. From these two subfigures and statistically significant analysis results, we can find UDP method UDP4 can perform significantly better than HDP methods in terms of P_{opt} performance indicator. Win/Tie/Loss result of comparing HDP methods and UDP methods in terms of P_{opt} can be found in Table 7. From this table, we can find: (1) In the scenario1, UDP methods UDP4 can win HDP methods at least 27 times. However, UDP methods UDP1, UDP2 and UDP3 can only win HDP methods at most 11, 11 and 7 times. (2) In the scenario2, UDP methods UDP4 can win HDP methods at least 25 times. However, UDP methods UDP1, UDP2 and UDP3 can only win HDP methods at most 10, 16 and 6 times. These results show UDP methods UDP4 can perform significantly better than HDP methods in the majority of cases when considering P_{opt}.

The results of all the methods in terms of *ACC* can be found in Fig. 4. From these two subfigures and statistically significant analysis results, we can find similar to the performance indicator P_{opt}, UDP methods UDP4 can also perform significantly better than HDP methods in terms of *ACC* performance indicator. Win/Tie/Loss result of comparing HDP methods and UDP methods in terms of *ACC* can be found in Table 8. From this table, we can find: (1) In the scenario1, UDP method UDP4 can win HDP methods at least 25 times. However, UDP methods UDP1, UDP2 and UDP3 can only win HDP methods at most 12, 14 and 7 times. (2) In the scenario2, UDP method UDP4 can win HDP methods at least 25 times. However, UDP methods UDP1, UDP2 and UDP3 can only win HDP methods at most 12, 18 and 8 times. These results show UDP method UDP4 can perform significantly better than HDP methods in the majority of cases when considering *ACC*.



(a) Scenario1



(b) Scenario2

Fig. 3. The results of all the methods via box plot in terms of P_{opt} performance indicator.

Table 7
Win/Tie/Loss analysis in terms of P_{opt} performance indicator.

(a) Scenario1

Method name	UDP1	UDP2	UDP3	UDP4
HDP2	24/1/9	18/5/11	25/4/5	2/2/30
HDP3	21/2/11	22/3/9	27/1/6	6/0/28
HDP4	22/4/8	21/3/10	25/3/6	2/3/29
HDP5	16/9/9	12/14/8	20/7/7	2/5/27

(b) Scenario2

Method Name	UDP1	UDP2	UDP3	UDP4
HDP1	8/20/6	9/9/16	15/14/5	4/4/26
HDP2	24/4/6	17/8/9	28/1/5	2/4/28
HDP3	21/3/10	22/3/9	26/2/6	6/0/28
HDP4	21/5/8	19/7/8	23/5/6	2/4/28
HDP5	14/12/8	6/21/7	16/12/6	1/8/25

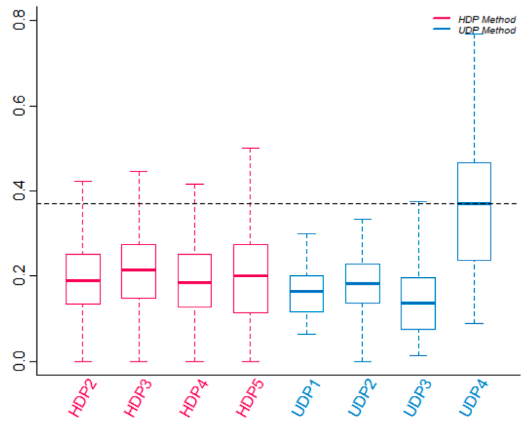
Table 8
Win/Tie/Loss analysis in terms of *ACC* performance indicator.

(a) Scenario1

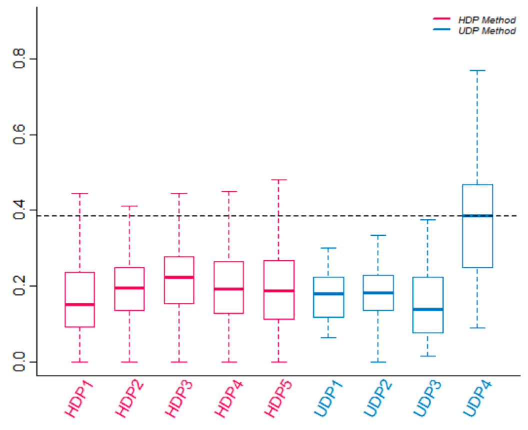
Method Name	UDP1	UDP2	UDP3	UDP4
HDP2	18/7/9	10/10/14	23/5/6	3/6/25
HDP3	24/4/6	19/4/11	25/4/5	6/3/25
HDP4	18/4/12	12/10/12	23/4/7	4/2/28
HDP5	14/13/7	7/20/7	18/9/7	3/6/25

(b) Scenario2

Method Name	UDP1	UDP2	UDP3	UDP4
HDP1	5/17/12	6/10/18	12/14/8	3/6/25
HDP2	17/11/6	10/14/10	22/7/5	2/7/25
HDP3	25/3/6	19/6/9	26/3/5	6/3/25
HDP4	16/9/9	10/14/10	22/6/6	3/4/27
HDP5	13/13/8	5/20/9	16/11/7	2/7/25



(a) Scenario1

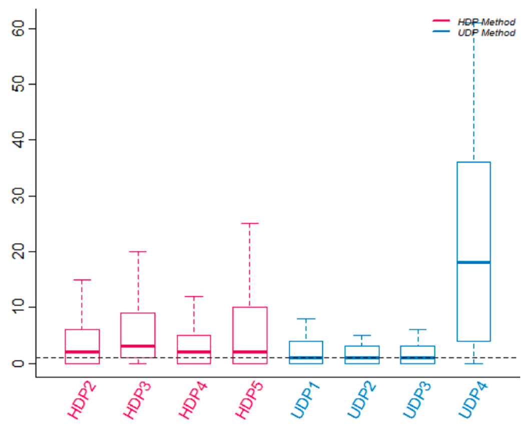


(b) Scenario2

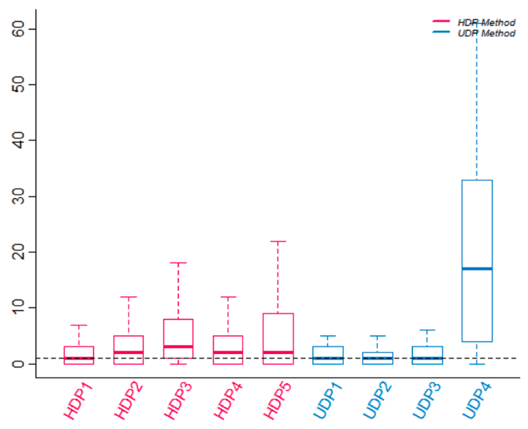
Fig. 4. The results of all the methods via box plot in terms of *ACC* performance indicator.

Based on the results in terms of *ACC* and P_{opt}, we find except for just-in-time defect prediction [19] and cross-project defect prediction [10], the simple unsupervised methods proposed by Zhou et al. [10] can still achieve better performance than state-of-the-art HDP methods.

The results of all methods in terms of *IFA* and *PMI*@20% can be found in Figs. 5 and 6 respectively. From these figures, we can find the method UDP4 has the highest *PMI*@20% value, which means developers need to inspect more program modules when available resources are limited. Moreover, the method UDP4 also has the highest *IFA*, which

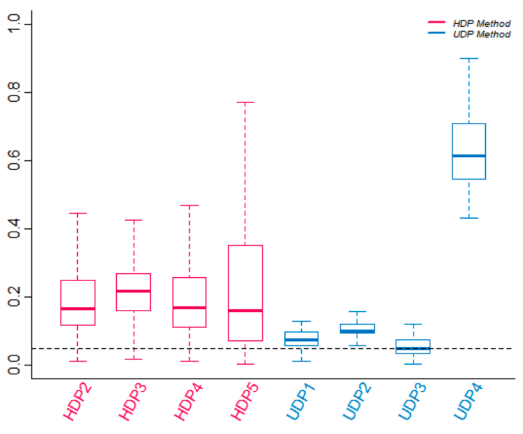


(a) Scenario1

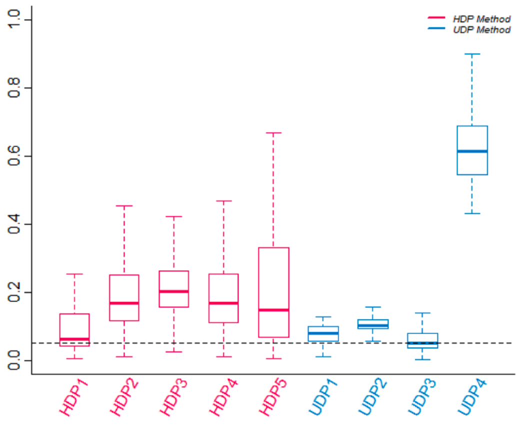


(b) Scenario2

Fig. 5. The results of all the methods via box plot in terms of *IFA* performance indicator.



(a) Scenario1



(b) Scenario2

Fig. 6. The results of all the methods via box plot in terms of *PMI*@20% performance indicator.

means many highly ranked modules are non-defective modules, and this may have a negative impact on developers' confidence and patience. These findings are in consistent with the findings found in just-in-time defect prediction by Huang et al. [22]. Finally, we are surprised to find HDP methods have higher *IFA* value and *PMI*@20% value than the remaining UDP methods, but the gap is not particularly significant.

Summary for RQ2: In terms of *ACC* and P_{opt}, the method UDP4 can perform significantly better than HDP methods. However, in terms of *IFA* and *PMI*@20%, the method UDP4 needs to inspect more modules when the available cost is limited and inspect many non-defective modules before encountering the first real defective module.

5.3. Results analysis for RQ3

Motivation. In the previous two RQs, we compare and rank the performance of HDP methods and UDP methods in terms of EPIs and NPIs in the same experimental setup. In this RQ, we want to perform an in-depth analysis of the defective modules correctly identified by different methods. Based on this analysis, we can first identify some defective modules that can only be identified by specific HDP methods or UDP methods. These findings can be used to design more effective HDP methods in the future. Second, we can identify the defective modules that cannot be identified by any HDP methods or UDP methods. This is helpful to identify the performance bottlenecks of current HDP studies.

Approach. To evaluate whether different methods (i.e., HDP methods or UDP methods) result in distinct predictions on defective modules, we want to test the following null hypothesis:

Ho: The methods $M1$ **and** $M2$**can identify similar defective program modules in the target project.**

We use McNemar's test [24] to perform a diversity analysis with the 95% confidence level. McNemar's test has been used in previous CPDP studies [13,54] for this kind of analysis. The details of McNemar's test can be found in these two studies [13,54]. Based on the result of McNemar's test, if the p value is smaller than 0.05, this means these two methods can almost identify distinct defective modules. While the p value is larger than 0.05, this means these two methods almost identify the same defective modules.

Results. We first analyze the diversity on identifying defective modules between HDP methods.

The final results can be found in Table 9. In this table, we can find the diversity on identifying defective modules exists in half of HDP combinations except for HDP2 *vs.* HDP4. In the best case, when HDP4 *vs.* HDP5, diversity on identifying defective modules exists in 73.8% of HDP combinations. When analyzing the dataset group's perspective, we can find that the diversity of identifying defective modules exists in half of HDP combinations. In the best case, diversity in identifying defective modules exists in 80% of HDP combinations when concerning the AEEEM dataset group.

We second analyze the diversity of identifying defective modules between UDP methods. The final results can be found in Table 10. In this table, we can find the diversity in identifying defective modules is less obvious than the comparison between HDP methods. In particular, The diversity only exists in half of UDP combinations for UDP3 *vs.* UDP4. In this best case, when UDP3 *vs.* UDP4, diversity on identifying defective modules exists in 50% of UDP combinations. When analyzing from the perspective of dataset group, we can find the diversity of identifying defective modules between two methods exists in half of UDP combinations when concerning NASA dataset groups. In the best case,

Table 9
Diversity analysis on identifying defective modules between HDP methods, Notice for X/Y numbers reported in each cell, when a project in a specific dataset group is set to the target project, X represents the number of successful HDP combinations with prediction diversity on defective modules for two methods, Y represents the number of all the successful HDP combinations.

Comparison	Relink	AEEEM	SoftLab	PROMISE	NASA	Summary
HDP2 *vs.* HDP3	70/93	90/145	68/145	96/240	174/339	498/962
HDP2 *vs.* HDP4	29/93	41/145	13/145	51/240	103/339	237/962
HDP2 *vs.* HDP5	73/93	116/145	80/145	169/240	252/339	690/962
HDP2 *vs.* HDP1	41/63	67/91	65/136	96/154	116/179	385/623
HDP3 *vs.* HDP4	75/93	86/145	74/145	113/240	176/339	524/962
HDP3 *vs.* HDP5	72/93	113/145	76/145	155/240	260/339	676/962
HDP3 *vs.* HDP1	58/63	67/91	97/136	116/154	124/179	462/623
HDP4 *vs.* HDP5	73/93	116/145	86/145	163/240	272/339	710/962
HDP4 *vs.* HDP1	46/63	62/91	74/136	103/154	127/179	412/623
HDP5 *vs.* HDP1	44/63	69/91	74/136	117/154	150/179	454/623
Summary	581/810	827/1234	707/1414	1179/2056	1754/2750	5048/8264

Table 10
Diversity analysis on identifying defective modules between UDP methods, Notice the meaning of *X*/*Y* numbers reported in each cell is the same as the illustration in Table 9.

Comparison	Relink	AEEEM	SoftLab	PROMISE	NASA	Summary
UDP1 *vs.* UDP2	1/3	2/5	0/5	3/10	3/11	9/34
UDP1 *vs.* UDP3	2/3	1/5	0/5	3/10	8/11	14/34
UDP1 *vs.* UDP4	1/3	2/5	0/5	6/10	4/11	13/34
UDP2 *vs.* UDP3	2/3	3/5	1/5	3/10	6/11	15/34
UDP2 *vs.* UDP4	0/3	2/5	0/5	3/10	4/11	9/34
UDP3 *vs.* UDP4	2/3	4/5	0/5	2/10	9/11	17/34
Summary	8/18	14/30	1/30	20/60	34/66	77/204

diversity in identifying defective modules exists in 51.5% of UDP combinations when concerning the NASA dataset group.

We third analyze the diversity in identifying defective modules between HDP methods and UDP methods. The experimental results can be found in Table 11. In this table, we can find the diversity in identifying defective modules is more obvious than the comparison between HDP methods or between UDP methods. In particular, we can find the diversity in identifying defective modules exists in half of HDP combinations in all the comparisons between HDP methods and UDP methods. In the best case, when HDP1 *vs.* UDP2, diversity on identifying defective modules exists in 91.3% of HDP and UDP combinations. When analyzing from the perspective of the dataset group, we can find the diversity in identifying defective modules exists in half of HDP combinations except for the SOFTLAB dataset groups. In the best case, diversity on identifying defective modules exists in 96.5% of HDP and UDP combinations when concerning the AEEEM dataset group.

Table 11
Diversity analysis on identifying defective modules between HDP Methods and UDP methods, Notice the meaning of *X*/*Y* numbers reported in each cell is the same as the illustration in Table 9.

Comparison	Relink	AEEEM	SoftLab	PROMISE	NASA	Summary
HDP1 *vs.* UDP1	59/63	86/91	107/136	130/154	170/179	552/623
HDP1 *vs.* UDP2	59/63	87/91	107/136	144/154	172/179	569/623
HDP1 *vs.* UDP3	58/63	87/91	98/136	138/154	165/179	546/623
HDP1 *vs.* UDP4	59/63	88/91	101/136	143/154	167/179	558/623
HDP2 *vs.* UDP1	80/93	143/145	56/145	160/240	305/339	744/962
HDP2 *vs.* UDP2	80/93	143/145	56/145	188/240	314/339	781/962
HDP2 *vs.* UDP3	74/93	144/145	39/145	165/240	289/339	711/962
HDP2 *vs.* UDP4	82/93	143/145	50/145	195/240	311/339	781/962
HDP3 *vs.* UDP1	90/93	145/145	44/145	99/240	278/339	656/962
HDP3 *vs.* UDP2	87/93	145/145	17/145	139/240	278/339	666/962
HDP3 *vs.* UDP3	69/93	145/145	41/145	81/240	274/339	610/962
HDP3 *vs.* UDP4	87/93	145/145	44/145	130/240	281/339	687/962
HDP4 *vs.* UDP1	66/93	145/145	55/145	161/240	301/339	728/962
HDP4 *vs.* UDP2	73/93	144/145	57/145	186/240	304/339	764/962
HDP4 *vs.* UDP3	60/93	145/145	28/145	167/240	284/339	684/962
HDP4 *vs.* UDP4	73/93	144/145	45/145	193/240	303/339	758/962
HDP5 *vs.* UDP1	79/93	127/145	89/145	161/240	296/339	752/962
HDP5 *vs.* UDP2	82/93	129/145	89/145	156/240	303/339	759/962
HDP5 *vs.* UDP3	77/93	123/145	85/145	156/240	280/339	721/962
HDP5 *vs.* UDP4	81/93	133/145	84/145	145/240	306/339	749/962
Summary	1475/1740	2591/2684	1292/2864	3037/4456	5381/6140	13776/17884

In summary, the diversity of prediction for defective modules across HDP *vs.* UDP methods is more than that within HDP methods or UDP methods. Therefore, as suggested by previous SDP studies [13,23,54], ensemble learning [25] is a potential way to further improve the performance of existing HDP methods by incorporating UDP methods.

Finally, we analyze the number of defective modules that cannot be identified by existing HDP methods/UDP methods. In this article, we only list the results when the projects in ReLink group is set as the target project. The final results can be found in Tables 12–14. In these three tables, since the method HDP1 cannot succeed in some HDP combinations (e.g., ant-1.3⇒Zxing), we only consider the HDP combinations the method HDP1 can achieve success. Moreover, "#= 0 by HDP" column and "#=0 by UDP" column represents the number of defective modules that cannot be identified by existing HDP methods or UDP methods respectively. The "(%)" represents the proportion of these defective modules to all the defective modules, The "#=0 by ALL" column represents the number of defective modules that cannot be identified by both HDP and UDP methods. For example, in this Table, when ant-1.3 is set to the source project and Apache is set to the target project, 40 defective modules in Apache cannot be identified by any HDP method, 25 defective modules cannot be identified by any UDP method, and moreover 17 defective modules cannot be identified by both HDP and UDP methods, which account for 40.82%, 25.51% and 17.35% of the number of defective modules in Apache respectively. Notice the results for other groups can be found in our project's website and similar findings can also be found. Based on the above results, we can find there exist a certain number of defective modules, which cannot be correctly predicted by either kind of method or even both kinds of methods. This means existing HDP methods and UDP methods still have certain limitations in the ability to identify defective modules.

Summary for RQ3: The diversity of prediction for defective modules across HDP vs. UDP methods is more than that within HDP methods or UDP methods. Moreover, there exist a certain number of defective modules, which cannot be correctly predicted by either kind of method or even both kinds of methods.

Table 12
Defective module detection analysis when apache project in ReLink group is set to the target project.

Source⇒Target	#(%) =0 by HDP	#(%) =0 by UDP	#(%) =0 by ALL
ant-1.3⇒Apache	40(40.82%)	25(25.51%)	17(17.35%)
ar3⇒Apache	4(4.08%)	25(25.51%)	0(0.00%)
ar4⇒Apache	8(8.16%)	25(25.51%)	6(6.12%)
ar5⇒Apache	34(34.69%)	25(25.51%)	16(16.33%)
arc⇒Apache	46(46.94%)	25(25.51%)	13(13.27%)
camel-1.0⇒Apache	9(9.18%)	25(25.51%)	8(8.16%)
CM1⇒Apache	2(2.04%)	25(25.51%)	1(1.02%)
EQ⇒Apache	1(1.02%)	25(25.51%)	1(1.02%)
JDT⇒Apache	1(1.02%)	25(25.51%)	1(1.02%)
JM1⇒Apache	9(9.18%)	25(25.51%)	7(7.14%)
KC3⇒Apache	17(17.35%)	25(25.51%)	12(12.24%)
MC2⇒Apache	8(8.16%)	25(25.51%)	5(5.10%)
MW1⇒Apache	11(11.22%)	25(25.51%)	4(4.08%)
skarbonka⇒Apache	44(44.90%)	25(25.51%)	15(15.31%)
tomcat⇒Apache	2(2.04%)	25(25.51%)	2(2.04%)
xalan-2.4⇒Apache	25(25.51%)	25(25.51%)	17(17.35%)

Table 13
Defective module detection analysis when safe project in ReLink group is set to the target project.

Source⇒Target	#(%) =0 by HDP	#(%) =0 by UDP	#(%) =0 by ALL
ant-1.3⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)
ar1⇒Safe	2(9.09%)	5(22.73%)	0(0.00%)
ar3⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
ar4⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)
ar5⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
ar6⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
arc⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)
camel-1.0⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
CM1⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
EQ⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
JDT⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
JM1⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)
KC3⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)
LC⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
MC1⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)
MC2⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)
ML⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
MW1⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
PC1⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
PC2⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
PC3⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
PC4⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
PC5⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
PDE⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
poi-1.5⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
skarbonka⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)
tomcat⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
velocity-1.4⇒Safe	3(13.64%)	5(22.73%)	0(0.00%)
xalan-2.4⇒Safe	0(0.00%)	5(22.73%)	0(0.00%)
xerces-1.2⇒Safe	1(4.55%)	5(22.73%)	1(4.55%)

Table 14
Defective module detection analysis when Zxing project in ReLink group is set to the target project.

Source⇒Target	#(%) =0 by HDP	#(%) =0 by UDP	#(%) =0 by ALL
ar1⇒Zxing	29(24.58%)	32(27.12%)	3(2.54%)
ar3⇒Zxing	2(1.69%)	32(27.12%)	0(0.00%)
ar4⇒Zxing	69(58.47%)	32(27.12%)	29(24.58%)
ar5⇒Zxing	6(5.08%)	32(27.12%)	1(0.85%)
ar6⇒Zxing	63(53.39%)	32(27.12%)	23(19.49%)
arc⇒Zxing	87(73.73%)	32(27.12%)	28(23.73%)
camel-1.0⇒Zxing	69(58.47%)	32(27.12%)	27(22.88%)
EQ⇒Zxing	0(0.00%)	32(27.12%)	0(0.00%)
KC3⇒Zxing	82(69.49%)	32(27.12%)	28(23.73%)
LC⇒Zxing	0(0.00%)	32(27.12%)	0(0.00%)
MC2⇒Zxing	46(38.98%)	32(27.12%)	10(8.47%)
ML⇒Zxing	11(9.32%)	32(27.12%)	2(1.69%)
MW1⇒Zxing	0(0.00%)	32(27.12%)	0(0.00%)
skarbonka⇒Zxing	17(14.41%)	32(27.12%)	0(0.00%)
tomcat⇒Zxing	46(38.98%)	32(27.12%)	26(22.03%)
velocity-1.4⇒Zxing	7(5.93%)	32(27.12%)	0(0.00%)
xalan-2.4⇒Zxing	80(67.80%)	32(27.12%)	27(22.88%)

5.4. Discussion

Since HDP1 utilizes gain ratio-based feature selection method and select the top 15% features and other HDP methods do not consider any feature selection methods, this may pose a threat to internal validity for comparing HDP1 with other HDP methods. To alleviate this issue, we also gather the results of HDP1 without performing feature selection (i.e., HDP1-NFS). The final results via box plot in terms of different performance indicators can be found in Fig. 7. From this figure, we can find the performance of HDP1-NFS is significantly worse than HDP1 in terms of *AUC* and P_{opt}. In terms of *F1, ACC, IFA* and *PMI*@20%, HDP1-NFS has a similar performance with HDP1. In summary, the performance of HDP1-NFS is no significantly better than HDP1. Therefore, our empirical results for comparing HDP methods with UDP methods still hold, such as HDP methods do not perform significantly better than some of UDP methods.

6. Road ahead for heterogeneous defect prediction

In the early two systematic literature reviews (SLRs) for SDP [1] and CPDP [2], Some recommendations provided by these two SLRs still deserve the attention of future HDP studies. For example, the researchers cannot ignore the data quality problem caused by SZZ [55]. The researchers should perform feature selection/extraction on the gathered SDP datasets. The researchers should evaluate the performance of the proposed method on more commercial projects. The researchers should compare the performance of the proposed method with WPDP methods (such as random forest with hyperparameter optimization). The researchers should share their datasets and scripts for the convenience of replicating their study.

In this section, we present more observations about the road ahead for HDP according to the characteristic of HDP problem and the analysis of the previous HDP studies.

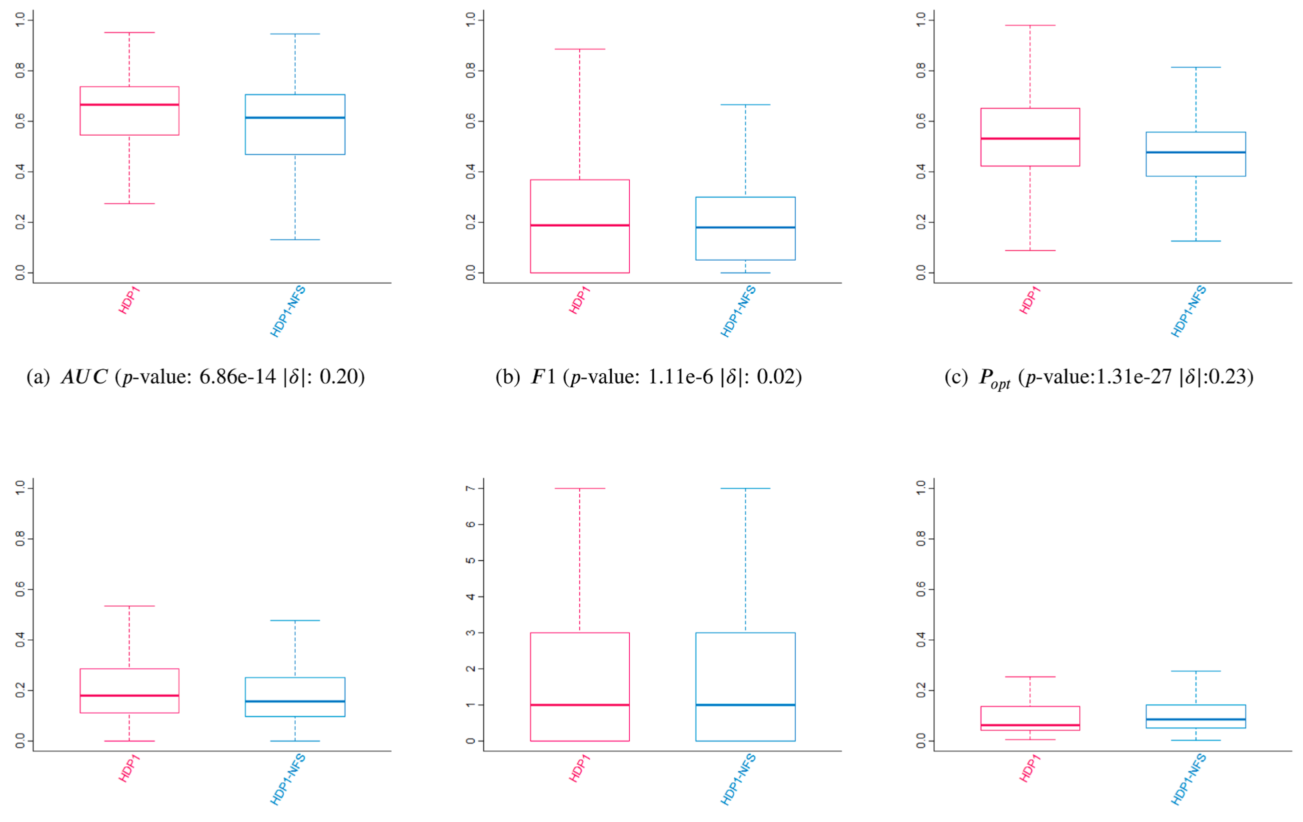
Considering the class imbalanced problem. The class imbalanced problem exists in most of the collected SDP datasets, and this problem is a major reason for the poor performance on predicting defective modules. This problem has been paid attention to in some of the previous HDP studies [7–9,33]. However, the existing solutions mainly focus on the cost-sensitive class imbalance methods or ensemble learning-based methods. In the future, researchers can resort to more types of the class imbalanced methods [56,56] or other advanced class imbalanced method [57] discussed in previously SDP studies.

Using mixed data. As discussed in Section 2, some previous CPDP studies [58,59] aimed to use mixed data to construct the CPDP models. However, only one HDP study [35] considered this kind of solution and more efforts are needed in this direction. The key of this direction is to use active learning [60,61] to query informative and representative modules in the target project and resort to the human experts to label these modules.

Combining HDP and UDP methods. Based on the analysis results in RQ3, we find UDP methods and HDP methods are complementary to each other in identifying defective modules. Therefore, we believe combining HDP methods and UDP methods can make full use of their complementarity and maybe a possible way to improve the performance of HDP methods.

Learning semantic information by deep learning. In RQ3, we also find that there are a certain number of defective modules that cannot be correctly identified by any existing HDP methods and UDP methods. One of the potential reasons is that all the methods are based on hand-crafted metrics, and these metrics have limited ability to distinguish different types of modules. With the increasing successful applications of deep learning in defect prediction [45,62–64], whether the methods based on the semantic features automatically learned by deep learning are complementary to UDP methods is an interesting problem and needs more investigation.

Performing hyperparameter optimization for HDP methods. Hyperparameter optimization is a standard tool for software analytics. There are four common optimal hyperparameter search algorithms: grid search, random search, Bayesian optimization, and sequential model-based optimization [65]. There exist several successful applications of hyperparameter optimization on software defect prediction [46,66,67]. As pointed out by Hossein et al. [2], performing hyperparameter optimization should be considered in the future CPDP studies. However, for existing HDP studies, the values of hyperparameters in these methods are determined empirically. Therefore, as a special sub-problem of CPDP, hyperparameter optimization for HDP methods cannot be ignored. In future HDP studies, we should not only identify the hyperparameters in classifiers, but also identify hyperparameters for metric selection or metric transformation methods used by HDP methods. Then we should investigate whether using hyperparameter optimization can



(a) AUC (p-value: 6.86e-14 $|\delta|$: 0.20) (b) $F1$ (p-value: 1.11e-6 $|\delta|$: 0.02) (c) P_{opt} (p-value:1.31e-27 $|\delta|$:0.23)

(d) ACC (p-value: 1.91e-9 $|\delta|$: 0.11) (e) IFA (p-value: 0.28 $|\delta|$: 0.04) (f) $PMI@20\%$ (p-value: 1.00e-4 $|\delta|$: 0.13)

Fig. 7. The comparison results between HDP1 and HDP1-NFS.

significantly improve the performance of previous HDP methods.

7. Threats to validity

Threats to internal validity. The first threat is to reduce the potential errors in our code implementation. For some of HDP methods and UDP methods (such as HDP1 to HDP4, UDP1 to UDP2), we utilize the implementations shared by previous studies [7–9,12] to avoid re-implementation faults. For some methods (such as HDP5, UDP3, UDP4), we implement their proposed methods and have similar performance after comparing related studies. For each considered method, we use the experimental setting suggested by the corresponding original study to guarantee the fairness of our empirical studies. Moreover, we also use test cases to examine the implementation correctness of all the EPIs and NPIs. The second threat is for classifiers used by HDP methods. We find Logistic regression is sensitive to multicollinearity. The performance of this classifier may suffer if the variance inflation factor analysis is not performed. In this study, we follow the experimental setup in previous HDP studies for a fair comparison. In the future, we should analyze the influence of handling multicollinearity in a Logistic regression classifier. The third threat is that we do not apply hyper-parameter optimization for HDP methods. In our study, the hyper-parameter values used in our study is based on the experimental setup in previous corresponding studies, since it can make a fair comparison between UDP and HDP methods. Whether using hyperparameter optimization can significantly improve the performance of HDP methods needs investigation in future HDP studies.

Threats to external validity. To guarantee the representative of our empirical subjects, we chose data sets which have been widely used in previous HDP research studies[7–9,12]. Some of these data sets have been verified and corrected in a manual way [40]. Moreover, we choose five HDP methods and four UDP methods proposed from 2014 to 2019, which are recently proposed in five years and have been accepted by top-tier journals or conferences. Therefore, we can guarantee the selected methods can reflect the state of the art of existing studies. In the future, more experimental subjects from open-source projects and commercial projects and new HDP methods should be applied to reduce external validity.

Threats to conclusion validity. To avoid the threats on the used statistical analysis methods, in this article, we use p-Value and Cliff's δ to examine whether the performance difference between two considered methods is statistically significant. Both the Wilcoxon signed-rank test and Cliff's δ are non-parameter methods and do not need to check whether the results of the cosidered methods satisfy the normality assumption.

Threats to construct validity. To guarantee the performance indicators used in our empirical studies reflect the real-word situation, we not only investigate two NPIs but also consider four EPIs [19,21,47,48,68]. Therefore, we can have a holistic look at the performance comparison between HDP methods and UDP methods. Moreover, for EPIs, we use the lines of code to measure the effort required to review the module. However, this assumption is still an open question [18], and other measures of effort can be used to evaluate the generalization of our findings. Finally, we do not compare HDP methods to WPDP methods, since this issue has been widely investigated in previous HDP studies [4,5,8].

8. Conclusion

In this article, to our best knowledge, we are the first to perform a comparative study to have a holistic look at state-of-the-art HDP methods. In particular, we compare five HDP methods with four UDP methods in terms of three different perspectives: NPIs, EPIs, and diversity analysis on identifying defective modules.

After conducting empirical studies on five groups of datasets, including 34 different projects from the open-source community and commercial enterprises, we achieve the following findings: First, HDP methods do not perform significantly better than some of UDP methods in terms of two NPIs and four EPIs. Second, The diversity of prediction for defective modules across HDP *vs*. UDP methods are more than within HDP methods or UDP methods. Finally, the performance of existing HDP methods is still pessimistic when considering two satisfactory criteria. Our study shows we still have a long way to go for HDP, and we present valuable observations about the road ahead for HDP.

We share our reproducible scripts and data on the website of our project.[6]

CRediT authorship contribution statement

Xiang Chen: Methodology, Software, Writing - original draft, Supervision. **Yanzhou Mu:** Data curation, Software. **Ke Liu:** Data curation, Software, Validation. **Zhanqi Cui:** Conceptualization, Writing - review & editing. **Chao Ni:** Conceptualization, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

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 to this work and they are co-first authors. This work is supported in part by the National Natural Science Foundation of China (Grant nos. 61702041, 61872263 and 61202006), and Guangxi Key Laboratory of Trusted Software (Grant no. kx202012).

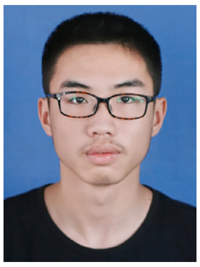
References

[1] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic literature review on fault prediction performance in software engineering, IEEE Trans. Softw. Eng. 38 (6) (2011) 1276–1304.
[2] S. Hosseini, B. Turhan, D. Gunarathna, A systematic literature review and meta-analysis on cross project defect prediction, IEEE Trans. Softw. Eng. 45 (2) (2019) 111–147.
[3] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[4] J. Nam, W. Fu, S. Kim, T. Menzies, L. Tan, Heterogeneous defect prediction, IEEE Trans. Softw. Eng. 44 (9) (2017) 874–896.
[5] J. Nam, S. Kim, Heterogeneous defect prediction. Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering, 2015, pp. 508–519.
[6] X. Jing, W. Fei, X. Dong, F. Qi, B. Xu, Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering, 2015, pp. 496–507.
[7] Z. Li, X.-Y. Jing, F. Wu, X. Zhu, B. Xu, S. Ying, Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction, Autom. Softw. Eng. 25 (2) (2018) 201–245.
[8] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, Heterogeneous defect prediction through multiple kernel learning and ensemble learning. Proceedings of the International Conference on Software Maintenance and Evolution, 2017, pp. 91–102.
[9] Z. Li, X.-Y. Jing, X. Zhu, H. Zhang, B. Xu, S. Ying, Heterogeneous defect prediction with two-stage ensemble learning, Autom. Softw. Eng. 26 (3) (2019) 599–651.
[10] Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, B. Xu, How far we have progressed in the journey? An examination of cross-project defect prediction, ACM Trans. Softw. Eng. and Methodol. 27 (1) (2018) 1:1–1:51.
[11] F. Zhang, Q. Zheng, Y. Zou, A.E. Hassan, Cross-project defect prediction using a connectivity-based unsupervised classifier. Proceedings of the International Conference on Software Engineering, 2016, pp. 309–320.
[12] J. Nam, S. Kim, CLAMI: Defect prediction on unlabeled datasets. Proceedings of International Conference on Automated Software Engineering, 2015, pp. 452–463.
[13] X. Chen, Y. Mu, Y. Qu, C. Ni, M. Liu, T. He, S. Liu, Do different cross-project defect prediction methods identify the same defective modules? J. Softw. Evol. Process 32 (5) (2020) e2234.
[14] P. He, B. Li, Y. Ma, Towards cross-project defect prediction with imbalanced feature sets, CoRR (2014). abs/1411.4228.
[15] Y. Benjamini, Y. Hochberg, Controlling the false discovery rate: a practical and powerful approach to multiple testing, J. R. Stat. Soc. Ser. B (Methodological) (1995) 289–300.
[16] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering, 2009, pp. 91–100.
[17] Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, An investigation on the feasibility of cross-project defect prediction, Autom. Softw. Eng. 19 (2) (2012) 167–199.
[18] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, N. Ubayashi, A large-scale empirical study of just-in-time quality assurance, IEEE Trans. Softw. Eng. 39 (6) (2013) 757–773.
[19] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, H. Leung, Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. Proceedings of the International Symposium on Foundations of Software Engineering, 2016, pp. 157–168.
[20] X. Chen, Y. Zhao, Q. Wang, Z. Yuan, MULTI: Multi-objective effort-aware just-in-time software defect prediction, Inf. Softw. Technol. 93 (2018) 1–13.
[21] Q. Huang, X. Xia, D. Lo, Supervised vs unsupervised models: a holistic look at effort-aware just-in-time defect prediction. Proceedings of the International Conference on Software Maintenance and Evolution, 2017, pp. 159–170.
[22] Q. Huang, X. Xia, D. Lo, Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction, Empir. Softw. Eng. (2018) 1–40.
[23] D. Bowes, T. Hall, J. Petrić, Software defect prediction: do different classifiers find the same defects? Softw. Qual. J. 26 (2) (2018) 525–552.
[24] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Comput. 10 (7) (1998) 1895–1923.
[25] Z.-H. Zhou, Ensemble Methods: Foundations and Algorithms, Chapman and Hall/CRC, 2012.
[26] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang, P. Yuan, Y. Tang, T. Zhang, Software defect prediction based on kernel PCA and weighted extreme learning machine, Inf. Softw. Technol. 106 (2019) 182–200.
[27] C. Ni, W.S. Liu, X. Chen, Q. Gu, D.X. Chen, Q.G. Huang, A cluster based feature selection method for cross-project software defect prediction, J. Comput. Sci. Technol. 32 (6) (2017) 1090–1107.
[28] C. Ni, X. Chen, F. Wu, Y. Shen, Q. Gu, An empirical study on pareto based multi-objective feature selection for software defect prediction, J. Syst. Softw. 152 (2019) 215–238.
[29] W. Liu, S. Liu, Q. Gu, J. Chen, X. Chen, D. Chen, Empirical studies of a two-stage data preprocessing approach for software fault prediction, IEEE Trans. Reliab. 65 (1) (2015) 38–53.
[30] Z.-W. Zhang, X.-Y. Jing, T.-J. Wang, Label propagation based semi-supervised learning for software defect prediction, Autom. Softw. Eng. 24 (1) (2017) 47–69.
[31] W. Dai, Q. Yang, G.-R. Xue, Y. Yu, Boosting for transfer learning. Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 193–200.
[32] Q. Yu, S. Jiang, Y. Zhang, A feature matching and transfer approach for cross-company defect prediction, J. Syst. Softw. 132 (2017) 366–378.
[33] M. Cheng, G. Wu, M. Jiang, H. Wan, G. You, M. Yuan, Heterogeneous defect prediction via exploiting correlation subspace. Proceedings of the International Conference on Software Engineering and Knowledge Engineering, 2016, pp. 171–176.
[34] Z. Li, X.Y. Jing, X. Zhu, H. Zhang, B. Xu, Y. Shi, On the multiple sources and privacy preservation issues for heterogeneous defect prediction, IEEE Transactions on Software Engineering PP (99) (2017).1–1
[35] Z. Li, X.Y. Jing, X. Zhu, Heterogeneous fault prediction with cost sensitive domain adaptation, Software Test. Verification Reliab. 28 (1) (2018) 1–22.
[36] S.v. Vaerenbergh, et al., Kernel Methods for Nonlinear Identification, Equalization and Separation of Signals, Universidad de Cantabria, 2010.
[37] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, A. Bener, Defect prediction from static code features: current results, limitations, new approaches, Autom. Softw. Eng. 17 (4) (2010) 375–407.
[38] A.G. Koru, K. El Emam, D. Zhang, H. Liu, D. Mathew, Theory of relative defect proneness, Empir. Softw. Eng. 13 (5) (2008) 473.
[39] M. D'Ambros, M. Lanza, R. Robbes, An extensive comparison of bug prediction approaches. Proceedings of the Working Conference on Mining Software Repositories, 2010, pp. 31–41.
[40] R. Wu, H. Zhang, S. Kim, S.-C. Cheung, Relink: recovering links between bugs and changes. Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering, 2011, pp. 15–25.
[41] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction. Proceedings of the International Conference on Predictive Models in Software Engineering, 2010.pp. 1–9:10
[42] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (1) (2007) 2–13.
[43] M. Shepperd, Q. Song, Z. Sun, C. Mair, Data quality: Some comments on the nasa software defect datasets, IEEE Trans. Softw. Eng. 39 (9) (2013) 1208–1215.

[6] https://github.com/EzioQR/HDPRevisit

[44] B. Turhan, T. Menzies, A.B. Bener, J. Di Stefano, On the relative value of cross-company and within-company data for defect prediction, Empir. Softw. Eng. 14 (5) (2009) 540–578.
[45] S. Wang, T. Liu, J. Nam, L. Tan, Deep semantic feature learning for software defect prediction, IEEE Transactions on Software Engineering (2018).1–1
[46] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto, Automated parameter optimization of classification techniques for defect prediction models. Proceedings of the International Conference on Software Engineering, 2016, pp. 321–332.
[47] Z. Xu, S. Li, X. Luo, J. Liu, T. Zhang, Y. Tang, J. Xu, P. Yuan, J.W. Keung, TSTSS: A two-stage training subset selection framework for cross version defect prediction, J. Syst. Softw. 154 (2019) 59–78.
[48] C. Ni, X. Xia, D. Lo, X. Chen, Q. Gu, Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction, IEEE Trans. Softw. Eng. (2020).
[49] A.N. Meyer, T. Fritz, G.C. Murphy, T. Zimmermann, Software developers' perceptions of productivity. Proceedings of the International Symposium on Foundations of Software Engineering, 2014, pp. 19–29.
[50] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, A.E. Hassan, Revisiting common bug prediction findings using effort-aware models. Proceedings of the International Conference on Software Maintenance, 2010, pp. 1–10.
[51] W.E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, A survey on software fault localization, IEEE Trans. Softw. Eng. 42 (8) (2016) 707–740.
[52] P.S. Kochhar, X. Xia, D. Lo, S. Li, Practitioners' expectations on automated fault localization. Proceedings of the International Symposium on Software Testing and Analysis, 2016, pp. 165–176.
[53] W. Fu, T. Menzies, Revisiting unsupervised learning for defect prediction. Proceedings of Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering, ACM, 2017, pp. 72–83.
[54] F. Zhang, I. Keivanloo, Y. Zou, Data transformation in cross-project defect prediction, Empir. Softw. Eng. 22 (6) (2017) 3186–3218.
[55] J. Śliwerski, T. Zimmermann, A. Zeller, When do changes induce fixes?. Proceedings of the International Workshop on Mining Software Repositories, 2005, pp. 1–5.
[56] Q. Song, Y. Guo, M. Shepperd, A comprehensive investigation of the role of imbalanced learning for software defect prediction, IEEE Trans. Softw. Eng. 45 (12) (2018) 1253–1269.
[57] K.E. Bennin, J. Keung, P. Phannachitta, A. Monden, S. Mensah, MAHAKIL: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction, IEEE Trans. Softw. Eng. 44 (6) (2017) 534–550.
[58] X. Xia, L. David, S.J. Pan, N. Nagappan, X. Wang, HYDRA: Massively compositional model for cross-project defect prediction, IEEE Trans. Softw. Eng. 42 (10) (2016) 977–998.
[59] L. Chen, B. Fang, Z. Shang, Y. Tang, Negative samples reduction in cross-company software defects prediction, Inf. Softw. Technol. 62 (2015) 67–77.
[60] Z. Xu, J. Liu, X. Luo, T. Zhang, Cross-version defect prediction via hybrid active learning with kernel principal component analysis. Proceedings of the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering, IEEE, 2018, pp. 209–220.
[61] Z. Yuan, X. Chen, Z. Cui, Y. Mu, ALTRA: Cross-project software defect prediction via active learning and tradaboost, IEEE Access 8 (2020) 30037–30049.
[62] J. Li, P. He, J. Zhu, M.R. Lyu, Software defect prediction via convolutional neural network. Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security, IEEE, 2017, pp. 318–328.
[63] Z. Xu, S. Li, J. Xu, J. Liu, X. Luo, Y. Zhang, T. Zhang, J. Keung, Y. Tang, LDFR: Learning deep feature representation for software defect prediction, J. Syst. Softw. 158 (2019) 110402.
[64] D. Chen, X. Chen, H. Li, J. Xie, Y. Mu, DeepCPDP: Deep learning based cross-project defect prediction, IEEE Access 7 (2019) 184832–184848.
[65] J. Chakraborty, T. Xia, F.M. Fahid, T. Menzies, Software engineering for fairness: a case study with hyperparameter optimization, arXiv preprint arXiv:1905.05786 (2019).
[66] A. Agrawal, T. Menzies, Is better data better than better data miners?: On the benefits of tuning smote for defect prediction. Proceedings of the 40th International Conference on Software Engineering, ACM, 2018, pp. 1050–1061.
[67] X. Chen, D. Zhang, Y. Zhao, Z. Cui, C. Ni, Software defect number prediction: unsupervised vs supervised methods, Inf. Softw. Technol. 106 (2019) 161–181.
[68] X. Chen, Y. Zhao, Z. Cui, G. Meng, Y. Liu, Z. Wang, Large-scale empirical studies on effort-aware security vulnerability prediction methods, IEEE Trans. Reliab. 69 (1) (2019) 70–87.

Xiang Chen received the B.Sc. degree in information management and system from Xi'an Jiaotong University, China in 2002. Then he received the M.Sc., and Ph.D. degrees in computer software and theory from Nanjing University, China in 2008 and 2011 respectively. He is with the School of Information Science and Technology at Nantong University as an associate professor. His research interests are mainly in software engineering. In particular, he is interested in empirical software engineering, mining software repositories, software maintenance and software testing. In these areas, he has published over 60 papers in refereed journals or conferences, such as IEEE Transactions on Software Engineering, Information and Software Technology, Journal of Systems and Software, IEEE Transactions on Reliability, Journal of Software: Evolution and Process, Software Quality Journal, Journal of Computer Science and Technology, ASE, ICSME, SANER and COMPSAC. He is a senior member of CCF, China, and a member of IEEE and ACM.



Yanzhou Mu is currently pursuing a master's degree with the College of Intelligence and Computing, Tianjin University. His research interests include software defect prediction and data mining.

Ke Liu is currently pursuing a bachelor's degree with the College of Information Science and Technology, Nantong University. Her research interests include software defect prediction and data mining.

Zhanqi Cui received the B.S. and Ph.D. degrees in software engineering and computer science from Nanjing University, in 2005 and 2011, respectively. He is currently an Associate Professor at the Beijing Information Science and Technology University. His research interest includes software analysis and testing.

Chao Ni received his B.Sc. degree in computer science from Nantong University, Nantong, in 2014. Then he received his M. Sc., and Ph.D. degrees in computer software and theory from Nanjing University, China in 2017 and 2020 respectively. He is currently a lecturer at the School of Software Technology, Zhejiang University. His research interests are mainly in software engineering. In particular, he is interested in mining software repositories, empirical software engineering, software maintenance and software defect prediction.