Cross-Project Change-Proneness Prediction

Chao Liu†, Dan Yang†, Xin Xia‡, Meng Yan§, Xiaohong Zhang†
†*School of Big Data & Software Engineering, Chongqing University, Chongqing, China*
‡*Faculty of Information Technology, Monash University, Melbourne, Australia*
§*College of Computer Science and Technology, Zhejiang University, Hangzhou, China*
Email: {liu.chao, dyang, xhongz}@cqu.edu.cn, xin.xia@monash.edu, mengy@zju.edu.cn

***Abstract*—Software change-proneness prediction (whether or not class files in a project will be changed in the next release) can help software developers to focus on preventive actions to reduce maintenance costs, and managers to allocate resources more effectively. Prior studies found that change-proneness prediction works well if there is sufficient amount of training data to build a model. However, it is not feasible for projects with limited historical data especially for new projects. To address this issue, cross-project change-proneness prediction, which builds a prediction model by using data in another project (*i.e.*, source project), and predicts the change-proneness in a target project, is proposed. Considering there are a large number of source projects, one challenge for cross-project change-proneness prediction is that given a target project, how to automatically select a source project which could show good prediction accuracy on it.**

In this paper, we propose a selective cross-project (SCP) model for change-proneness prediction. SCP automatically finds the source project which has the similar data distribution with the target project by measuring distribution similarity between source and target projects. We evaluate SCP by conducting an empirical study on 14 open source projects. We compare it with 2 most related change-proneness models, including RCP (Random Cross-Project prediction) proposed by Malhotra and Bansal, and CLAMI+ developed by Yan *et al.* Experiment results show that SCP improves RCP and CLAMI+ by 25.34% and 4.30% in terms of AUC respectively; and by 171.42% and 172.31% in terms of cost-effectiveness, respectively.

***Keywords*-Project Selection, Cross-project Prediction, Change-Proneness, Maintainability**

I. Introduction

In the whole life cycle of software development, software maintenance consumes most of the cost, both in terms of time and money [1]. In the maintenance phase, developers spend significant efforts to modify source files to fix bugs, refactor code, and improve functional or nonfunctional attributes [2], [3], [4]. Hence, for timely release of a project, classes files that are going to be changed in next release indicate that the development requirements are not clear or the classes require most tests [5]. Precisely predicting the change-proneness of classes (whether or not each class file will be changed in the near future, *i.e.*, next release) can help software developers focus on preventive actions to reduce maintenance costs, and software managers allocate resources more effectively [6], [7].

Existing Work and Challenge. In the past two decades, researchers have spent significant efforts on predicting change-prone classes. They found that some object-oriented metrics [7], [6] (*e.g.,* depth in inheritance tree, and number of children) and internal quality attributes [8] (*e.g.,* lines of code) strongly correlate to the change-proneness of classes. Based on these metrics and attributes, machine learning (ML) models have been successfully used to improve the prediction performance [9], [5], *e.g.,* Bayesian network [10], neural network [8], support vector machine (SVM) [5], and ensemble methods [11].

Most of the above-mentioned models based on ML build the prediction model by learning from historical labeled data within a project (*i.e.*, within-project prediction). However, in practice, it is often time-consuming and expensive to collect labeled data. Moreover, it is also difficult to apply these models on projects with limited historical data, especially for new projects [12].

To address the drawback of within-project prediction, Malhotra and Bansal [13] investigated the cross-project prediction, and they built a change-proneness model by using data in another project (*i.e.*, source project), and predicted the change-proneness of classes in a target project. However, considering there are a large number of source projects, one challenge for cross-project change-proneness prediction is that given a target project, how to automatically select a source project which could show good prediction accuracy on it. We refer to this problem as *source project selection*.

The Proposed Approach. To address the problem of source project selection, we propose a Selective Cross-Project (SCP) model to automatically find a better source project from candidates for improving the performance of cross-project change-proneness prediction. To the best of our knowledge, this is the first work of proposing selective cross-project model for change-proneness prediction. In detail, SCP works in two steps: 1) SCP first estimates the unknown labels (change-prone or not) of classes in target project by an unsupervised approach; 2) SCP then finds the source project with best-matched distribution, compared with the target project with previously estimated label information, to train a classifier. The final predicted labels by the classifier are evaluated by measured ones (the ground-truth is described in Section III).

To evaluate the effectiveness of our model, we conduct an empirical study on 14 open source projects, containing a total of 10,739 class files. We evaluate the performance of SCP in terms of AUC and cost-effectiveness. AUC measures the discriminatory power of a prediction model. Cost-effectiveness measures the percentage of real change-prone classes that developers can identify, when developers spend code inspection efforts (*e.g.,* reading 20% lines of code) on change-prone classes recommended by a model. A model with better cost-effectiveness saves more developers' efforts in software maintenance. Cost-effectiveness has been widely used for evaluating software prediction models [14], [15], [16], [17].

In the experiment, we compare the SCP with 2 most related models for change-proneness prediction. One is the Random Cross-Project (RCP) prediction by randomly selecting source project [13]. The other is a state-of-the-art model for change-proneness prediction, namely CLAMI+ (Clustering, Labeling, Metric selection, and Instance Selection Plus) [12]. CLAMI+ outperforms RCP in discriminatory power, but it does not consider the cost-effectiveness [12]. The empirical results show that, on average across 14 datasets, our model achieves the best performance in terms of AUC (0.687) and cost-effectiveness score (38.40%). SCP outperforms RCP and CLAMI+ by 25.34% and 4.30% in terms of AUC respectively; by 171.42% and 172.31% in terms of cost-effectiveness, respectively.

In addition, we also compare SCP with 2 similar cross-project models studied in defect prediction, *i.e.*, TCA+ (Transfer Component Analysis Plus) [18], and TDS (Training Data Selection) [19]. TCA+ improves the performance of cross-project prediction by reducing distribution difference between source and target projects [20], [21]. TDS is to select source projects by using a distribution similarity measure between a set of candidate source projects and a target project. The main difference between TDS and our approach is the way to measure project similarity. The similarity measure of TDS is based on the distribution characteristics of software metrics between two projects, while our similarity measure uses the label information between two projects, and we use an unsupervised learning approach to estimate the labels for the classes in the target project. We estimate labels in the target projects since a prior study found that it can help to identify a source project with a similar data distribution as the target project [22]. Experimental results show that SCP outperforms TDS and TCA+ by 21.26% and 17.62% in terms of AUC; by 131.75% and 5.70% in terms of cost-effectiveness, respectively.

Contributions. The main contributions of this study are:

- We propose a novel selective cross-project model (SCP) for software change-proneness prediction, which can train a classifier better by using the selected source project with the best-matched distribution to the target project.

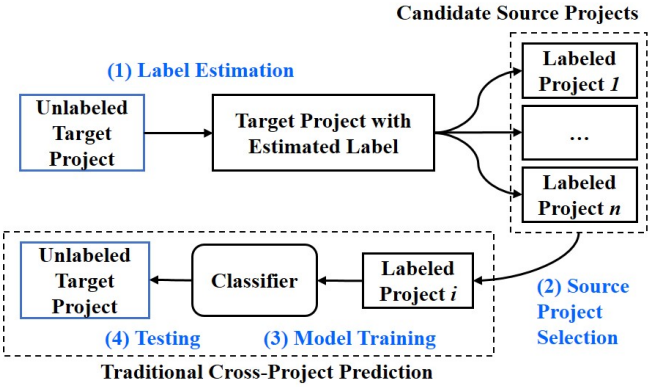

Figure 1. Overall architecture of SCP.

- We conduct an empirical study on 14 open source projects to evaluate the effectiveness of our model. Results show that SCP outperforms the most related two models RCP and CLAMI+ considering AUC and cost effectiveness. And SCP also outperforms 2 similar cross-project prediction techniques (TDS and TCA+) in a substantial range.
- We find that using more source projects has negative effect on the performance of SCP. And if SCP is trained by multiple source projects, its performance is sensitive to the training sequence of used source projects.

Paper Organization. The remainder of this paper is organized as follows. Section II presents the details of the proposed model. Section III describes experimental setup and evaluation criteria. Section IV presents experiment results and threats to validity. Section V briefly mentions related work. Finally, Section VI summarizes our findings.

II. Methodology

In this section, we first describe the overall architecture of our SCP model in Section II-A. Next, we present the details of each component in SCP.

A. Overall Architecture

Cross-project prediction is unstable due to the mismatched distribution between source and target projects [13], [23]. Our model aims to find the best-matched source project from a set of candidate source projects, instead of randomly choosing one project, to train a classifier. The trained classifier thus can identify change-prone classes in the target project with better performance. In this study, each class in a source or target project is represented by 7 code metrics, following prior studies [13], [6], [24], such as depth in inheritance tree, and lines of code (details in Section III-A).

Fig. 1 presents the overall architecture of SCP. Unlike the traditional cross-project prediction (Steps 3-4) that trains a classifier with a randomly selected source project, SCP automatically chooses a candidate source project that has a high distribution similarity with the target project (Steps 1-2). Specifically, SCP first estimates the missed labels of the target project by an unsupervised approach (Step-1), which

is elaborated in Section II-B. It then selects a source project from candidates by comparing the similarity between a set of candidate source projects and the target project (Step-2), where we leverage 3 similarity strategies in Section II-C. The strategies are based on the label information of source and target projects, because researchers observed that label information can help the source project selection [22]. As the labels of classes in target project are unknown in practices, so we use the estimated labels by Step-1 as a substitute. Afterwards, SCP trains an underlying classifier with the selected source project (Step-3). We use Bayesian network as the classifier. It will be described in Section II-D. The trained classifier finally predicts the labels of the target project (Step-4), which are estimated by the measured labels (the ground-truth is described in Section III-A). The following 3 subsections respectively describe the implementation details of label estimation, source project selection, and the classifier.

B. Step-1: Label Estimation (LE)

To estimate the unknown labels of classes in a target project, we use an unsupervised label estimation (LE) approach that is also used in prior study [12]. The basic idea of this estimation is to cluster classes with different levels of complexity, and label the classes with higher complexity to be change-prone. One main concern that the LE step has already estimated the labels of target project. Why do we still need to build a cross-project prediction model? The reason is that we observe that unsupervised model has a poor performance in terms of cost-effectiveness. We investigate the effectiveness of LE and compare it with our proposed model in Section IV-A. Thus, we only use the LE step to help select source project.

Generally, LE works in two phases: 1) clustering classes in the target project into groups; 2) labeling some groups of classes with higher complexity to be change-prone. Fig. 2 shows an example to estimate the labels of a target project with 5 classes (C_1-C_5) represented by 3 code metrics (M_1-M_3). The two phases of LE are detailed as below.

Clustering Phase. The goal of this phase is to cluster classes in the target project into groups, according to the complexity of classes. As researchers found that a class whose metric values exceed the median of that metric is more likely to be change-prone [25], [12], we can represent the complexity of a class by counting the number of metrics whose values larger than corresponding median values.

Fig. 2 illustrates the way we cluster classes in a target project. Specifically, we first calculate median values for 3 code metrics, $Median([M_1, M_2, M_3]) = [1, 2, 2]$. We then identify whether a metric value exceeds the median value of that metric. Afterwards, we transform the difference from metric values to their median values to a complexity table, which indicates whether each metric value exceeds corresponding median (exceeded values are highlighted). Finally, by counting the number of higher valued metrics,

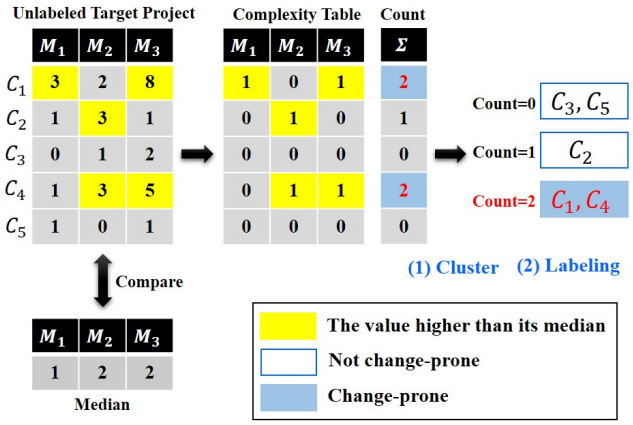

Figure 2. Illustration of label estimation for a target project with 5 classes ($C_1 - C_5$) represented by 3 code metrics ($M_1 - M_3$).

we can cluster classes into 3 groups according to their complexity, *i.e.* $count = 0, 1, 2$.

Labeling Phase. As a class with higher complexity tends to be change-prone [25], [26], [12], therefore the clustered groups can be divided by two halves, where the half with larger count value is likely to be change-prone. We use the median count as the division coefficient following researchers' studies [12], [27]. Therefore, the classes in the groups with count value larger than the division coefficient are labeled as change-prone.

In specific, as exemplified in Fig. 2, 3 clustered groups with count 0-3 respectively have 2, 1, and 2 classes. By dividing these counts with their median (equaling to 1), two classes (C_1 and C_4) in the group with count = 2 are labeled as change-prone, while other classes are labeled as not change-prone.

C. Step-2: Source Project Selection

To find the best source project from candidates, we design 3 strategies to measure the similarity between source and target projects. Since label information is beneficial to the source project selection [22], all the designed strategies thus involve label information, where the unknown labels of target projects are estimated by the first phase. Note that we only use the estimated labels in step 1 to measure the similarity between candidate source project and target project. The strategies are:

Strategy 1: Similarity on Change-Proneness Rate. It is shown that classes can be clustered based on their change rates, because clustered classes have similarity in importance, underlying architecture, or chronic problems [28]. Similarly, we assume that projects can be also clustered based on their percentages of change-prone classes, where the clustered projects have similarity in underlying architecture, development phase, or *etc.* As the source project selection aims to find a source project similar to the target project, this strategy therefore represents the similarity between source and target project by calculating the absolute

difference of change-proneness rates between two projects. For example, Fig. 2 shows that 2 of 5 classes are estimated to be change-prone, so that its change-proneness rate is 0.4.

Strategy 2: Similarity on Metric Distribution of Not Change-Prone Classes. Researchers have found that a source project with high similarity to the target project in terms of their metric distribution is beneficial to the classifier training [19], and incorporating label information to the similarity measure can select a better source project for training [22]. We therefore use the label information as a factor to distinguish the type of classes (change-prone or not). This strategy measures the metric distribution of not change-prone classes between source and target project, and the next strategy focuses on the change-prone classes.

Specifically, the metric distribution of one project can be represented by a characteristics vector [19]. This vector contains the mean and variance values of each code metric among not change-prone classes. Hence, the characteristics vector of the exemplified target project in Fig. 2 has 3 means (0.67, 1.33, 1.33) and 3 variances (0.33, 2.33, 0.33). Then, the metric distribution similarity of two projects can be calculated by the Euclidean distribution on their characteristics vectors [19]. In this way, a shorter distance indicates higher similarity between two projects.

Strategy 3: Similarity on Metric Distribution of Change-Prone Classes. Similar to the strategy 2, this strategy measures the metric distribution similarity between source and target projects, but focusing on change-prone classes.

Section IV-C compares the performance of SCP applied with 3 designed strategies (respectively called SCP1-3), and it also investigates the prediction performance of SPC with linear combined strategies.

D. Step-3: Classifier Building

After the step 2 for source project selection, we train a classifier by learning from the selected source project. We use Bayesian network, a powerful classifier for the change-proneness prediction [29], as our default classifier. It is a probabilistic graphical model that represents a set of random variables and their dependencies via a directed acyclic graph (DAG). In the graph, a directed edge from node N_i to node N_j indicates that the variable N_j is conditionally dependent on the variable N_i. Each node keeps a conditional probability distribution (CPD) $p(N_j|N_i)$. The graph together with the CPD describes a joint distribution of all random variables. In this study, we use the Bayesian network for SCP due to its good performance in cross-project predictions [30], [31]. We also investigate the impact of varying other common classifiers in Section VII. The Bayesian network is implemented by invoking Weka[1] with default settings [32].

[1]Weka contains a collection of common machine learning models written in Java. http://www.cs.waikato.ac.nz/ml/weka/

III. Experimental Setup

A. Experimental Dataset

Dataset. We evaluate our model on 14 open source projects from a public dataset Qualitas Corpus [33] same as Yan *et al.* [12] These projects are written in Java and have multiple evolution versions. For a project, we choose its latest two versions to study the change-proneness prediction, as the studied version and next version listed in Table II. We aim to predict whether class files of a project in the studied version will be changed in the next version. Table II shows the number of classes (#Class) and lines of code (#LOC) in the studied versions.

Dependent Variable. We measure the labels of studied projects by tracking the version control system following previous studies [34], [29], [12], where a change-prone class in the studied version is the one modified in the next version. Table II provides the change-proneness rate (%Changed) of projects in their studied versions with a substantial range, which can validate the model ability among a wide range.

Independent Variables. To predict the change-proneness of a class, we represent it by 7 typical object-oriented code metrics, following previous studies [13], [6], [24]. It is shown that these metrics are strongly correlated with the change-proneness of classes [6], [24]. Table I lists the definition of these metrics.

Source and Target Projects in Cross-Project Prediction. To simulate the practical usage of our model and follow the cross-project setting used in previous studies [13], [6], when we consider a project (*e.g.,* ant-1.8.3) as the target project, we use other 13 projects (antlr-3.2, ..., weka-3.5.7) as the candidate source projects. In total, we perform 14 cross-project change-proneness predictions.

B. Baseline Models

We compare the proposed model with two most related change-proneness models, CLAMI+ [12] and RCP [13]. RCP is based on the logistic regression classifier. We re-implemented RCP by invoking Weka [32] with default settings. CLAMI+ is a state-of-the-art model, which has already proved its advantages over RCP. We used the source code of CLAMI+ provided by the author. Moreover, we also compared SCP with two similar successful cross-project models for defect prediction, TDS [19] and TCA+ [18]. TDS is the most related work to SCP for source project selection, we re-implemented it according to the pseudo-code written in the original paper. TCA+ is an extended version of TCA with some data preprocessing. We developed TCA part according to [20] and the extended part following the algorithm in Nam *et al.* [18].

C. Performance Metrics

We adopt two widely used metrics to evaluate our model:

AUC. AUC measures a model's discriminatory power, which represents the area under receiver operating characteristic

Table I
SUMMARY OF ADOPTED CODE METRICS.

Name	Full Name	Description
DIT	Depth in Inheritance Tree	The level for a class file within its class inheritance hierarchy.
NOC	Number Of Children	Number of immediate subclasses inherited to the given class file.
CBO	Coupling Between Object class files	Number of class files perform interaction with a given class file.
WMC	Weighted Methods per Class	Count of methods implemented within a class file.
RFC	Response For a Class	Number of different executive methods when received message as an object.
LCOM	Lack of Cohesion in Methods	Count of the set of methods in a class file that are not sharing its fields.
LOC	Lines Of Code	Number of lines of one source code class file.

Table II
STATISTICS OF THE DATASETS.

Studied Version	Next Version	#Class	%Changed	#LOC
ant-1.8.3	ant-1.8.4	846	12.20%	105,153
antlr-3.2	antlr-3.3	226	84.07%	32,227
argouml-0.32.2	argouml-0.32.3	1505	27.51%	114,707
azureus-4.1.0.4	azureus-4.2.0.0	3150	07.17%	447,704
freecol-0.10.6	freecol-0.10.7	630	09.52%	99,937
freemind-0.6.7	freemind-0.7.1	74	89.19%	9,978
hibernate-3.1.2	hibernate-3.1.3	925	93.62%	86,438
jgraph-5.12.1.0	jgraph-5.12.2.1	53	20.75%	13,588
jmeter-2.8.0	jmeter-2.9.0	830	58.07%	82,872
jstock-1.0.7.2	jstock-1.0.7.3	276	10.14%	48,128
jung-1.7.5	jung-1.7.6	467	02.78%	33,657
junit-4.8.2	junit-4.9.0	143	20.98%	6,608
lucene-4.0.0	lucene-4.1.0	620	34.19%	74,863
weka-3.5.7	weka-3.5.8	994	68.61%	238,144

(ROC) curve [12]. We use the AUC as the previous studies *et al.* [12], [35], [36], [37]. The advantages of AUC over other metrics (*e.g.,* F1-score) are that: 1) AUC is a threshold independent measure [38]. A threshold represents the likelihood threshold for deciding an instance is classified as positive or negative. Usually, the threshold is set as 0.5 and other performance measures (such as F1-score) for a classifier rely on the determination of a threshold. 2) AUC is robust towards imbalanced dataset [30], [39]. Other performance measures such as precision, recall, and F1-score are highly affected by imbalanced dataset, which are difficult to fairly compare models [40], [27].

PofB20. PofB20 is a widely used cost-effectiveness measure for evaluating prediction models [41], [42], [43], [17], [44], [45], [46]. PofB20 measures the percentage of real change-prone classes that developers can identify, when developers spend code inspection efforts (*i.e.* reading 20% lines of code) on change-prone classes recommended by a model. Therefore, a model with higher PofB20 saves more developers' efforts in software maintenance. Specifically, it first sorts classes in a project by a confidence level (*i.e.* the probability to be change-prone predicted by a classifier). A class with higher confidence level has higher probability to be change-prone. Then, we simulate developers' behaviors to inspect predicted change-prone files one by one from the highest confidence level. We finally calculate the recall for the classes that hold 20% of total lines of code.

D. Research Questions

In the experiment, we design 5 research questions (RQs) to evaluate our model as follows.

RQ1. How effective is SCP? How much improvement can it achieve over two existing cross-project models for change-proneness prediction?

In this RQ, we investigate the extent SCP outperforms existing cross-project change-proneness prediction models. We answer this question by comparing SCP with a state-of-the-art model CLAMI+ [12], and a cross-project model RCP [13]. We use AUC and PofB20 to evaluate the performance of these models on 14 datasets (described in Table II). For each dataset, as RCP trains its classifier with one randomly selected source project. To suppress the randomness, we thus run RCP 20 times for each dataset, and report the mean value of its performance.

Moreover, we also compare SCP with its component LE, an unsupervised approach described in Section II-B, to investigate the effectiveness of LE. Note that LE has no value in PofB20, because it cannot provide confidence level as other supervised models, so that LE cannot guide developers identify change-prone classes one by one as other models.

To analyze the statistical difference between two models across 14 datasets, we apply the Wilcoxcon signed-rank test [47] at 95% significance level on 14 paired values that corresponds to a performance metric (AUC or PofB20). Since the study conducts several statistical tests at the same time (one test per dataset per evaluation metric), we thus use the Bonferroni correction [48] to counteract the results of multiple comparisons.

To quantify the amount of difference between two models, we utilize the Cliff's delta (δ) [49], a non-parametric effect size measure. δ ranges from -1 to 1, where δ = -1 or 1 indicates that one model outperforms another model on all datasets in terms of a performance metric (such as AUC), while δ = 0 means that results of two models are completely overlapping without any difference. Table III presents interpretations for different delta values [49].

Table III
CLIFF'S DELTA AND THE EFFECTIVENESS LEVEL [49].

No.	Cliff's Delta ($\lvert\delta\rvert$)	Effectiveness Level
1	$0.000 \leq \lvert\delta\rvert < 0.147$	Negligible
2	$0.147 \leq \lvert\delta\rvert < 0.330$	Small
3	$0.330 \leq \lvert\delta\rvert < 0.474$	Medium
4	$0.474 \leq \lvert\delta\rvert \leq 1.000$	Large

RQ2. How effectively can SCP perform as compared to similar cross-project defect prediction models?

Although the problem of cross-project setting has not

been solved in the change-proneness prediction, there are some viable solutions for the cross-project setting in defect prediction. We thus compare SCP with 2 previously successful and most related defect models, including a source selection model TDS [19], and a data preprocessing model TCA+ [18]. Notice that TCA+ trains its classifier with randomly selected one source project, we thus repeat TCA+ 20 times for one cross-project prediction to suppress its randomness as the setting of RCP model in RQ1. The prediction performance between SCP and other 2 models are also analyzed by Wilcoxon signed-rank test [47] (also corrected by the Bonferroni correction [48]), and Cliff's delta [49] as the settings in RQ1.

RQ3. How different source project similarity strategies affect the performance of SCP?

Section II provides 3 similarity strategies to help SCP find better source project: *1) SCP1*, an absolute difference of the change-proneness rates between source and target projects; *2) SCP2*, a metric distribution similarity of not change-prone classes in source and target projects; *3) SCP3*, a similarity strategy also likes *SCP2* but measures the metric distribution on change-prone classes. Due to the limited space in this paper, we only report the average performance of SCP with different strategies across 14 datasets, and analyze which strategy is the best choice for SCP. Additionally, to test if above 3 strategies are complementary, we also explore the prediction performance of SCP with different linearly combined strategies.

RQ4. How the classifier affects the performance of SCP?

SCP helps a classifier find better source project, and the classifier finally determines the performance of cross-project change-proneness prediction. Thus, this RQ investigates how different classifiers affect the prediction performance of SCP. Table VII shows 7 commonly used classifiers, which are implemented by invoking Weka [32] with default settings.

RQ5. How the number of source projects influences the performance of SCP?

In default, SCP trains its classifier with only one selected source project. To investigate whether the prediction can be further improved by using more source projects, we hence compare the performance of SCP with top-n and bottom-n selected source projects, where $n \in [1, 13]$.

IV. Results and Discussion

A. RQ1: SCP vs. Two Related Change-Proneness Models.

Table IV presents the AUC and PofB20 values of our SCP model compared with 2 related change-proneness prediction models, a cross-project prediction model RCP and a state-of-the-art model CLAMI+. From the table, we can find that the improvements of our SCP over 2 baselines are substantial. Specifically, across 14 datasets, the AUC scores of SCP vary from 0.581 to 0.870 with average 0.687, and SCP improves RCP and ClAMI+ in average AUC by 25.34% and 4.30%. Meanwhile, the PofB20 of SCP within the range of 03.95% and 64,59% (average = 38.40%), and SCP improves two baselines in mean PofB20 by 171.42% and 172.31% respectively.

Table IV lists the number of projects where each baseline model obtains better (W, win), equal (T, tie), or worse (L, lose) performance comparing to SCP. We can find that SCP achieves better AUC and PofB20 on more than 10 projects compared with 2 baselines, which indicates that SCP achieves better performance in most of cases.

Table IV also provides the p-value and Cliff's delta, when we compare SCP with 2 baseline models. And we consider a comparison is significantly different when the p-value is less than 0.05. We can observe that, in terms of AUC, SCP shows significant increases for RCP (p<0.001) with large effect size ($\delta = 0.938$); meanwhile, SCP shows little improvement over CLAMI+ on AUC (p>0.05, δ=0.122). Comparing PofB20 of 3 models, we can find out that SCP is significantly better than other two models (both $p < 0.001$ and δ>0.7).

Additionally, when comparing LE (the component of SCP) with other models in Table IV, we can find that its mean AUC (0.633) is close to the state-of-the-art model CLAMI+, outperforming RCP. Thus the lightweight unsupervised method LE can effectively estimate the label information of target project. We can also observe that with the help of the estimated labels by LE, SCP can further improve the prediction performance in terms of AUC and PofB20.

> **Result 1:** *The proposed model SCP outperforms RCP and CLAMI+ on AUC and PofB20 for most of datasets with larger mean values, where the improvements on PofB20 are significant with large effect size.*

B. RQ2: SCP vs. Two Similar Cross-Project Defect Models.

Table V compares the performance of SCP with 2 similar defect models (TCA+ and TDS) succeeded in handling cross-project prediction, in terms of AUC and PofB20. Compared with these 2 models, SCP shows better performance. Specifically, the average AUC and PofB20 of TCA+ are respectively enhanced by 21.26%, and 131.75%. And all of these improvements are significant (p<0.001) with large effect size (δ>0.78). For the TDS, it has a high average PofB20 (36.34%), where SCP only makes 5.70% improvements without statistical difference (p>0.05, δ=0.122). However, TDS has low average AUC (0.584) across 14 projects, where SCP improves it by 17.62% in statistical significance (p<0.001) with large effect size (δ>0.64). These results produced by TDS imply that source project selection can help developers save efforts for identifying change-prone classes. And a better selection method as SCP can further improve the prediction performance.

Table IV
PERFORMANCE COMPARISON (AUC AND POFB20) OF 3 CROSS-PROJECT CHANGE-PRONENESS MODELS (RCP, CLAMI+, SCP) AND THE COMPONENT OF SCP (LE). 'W/T/L' INDICATES THE NUMBER OF PROJECT THAT A BASELINE IS BETTER, EQUAL TO, WORSE THAN SCP.

Dataset	AUC				PofB20			
	RCP	LE	CLAMI+	SCP	RCP	LE	CLAMI+	SCP
ant-1.8.3	0.533	0.646	0.750	**0.811**	05.00%	-	00.00%	**46.94%**
antlr-3.2	0.554	0.641	0.653	**0.654**	05.21%	-	12.04%	**64.59%**
argouml-0.32.2	0.533	0.562	**0.577**	0.566	07.79%	-	11.22%	**49.06%**
azureus-4.1.0.4	0.573	0.662	0.654	**0.714**	15.15%	-	18.52%	**24.07%**
freecol-0.10.6	0.572	0.695	0.699	**0.723**	21.50%	-	**23.33%**	17.39%
freemind-0.6.7	0.538	0.642	0.625	**0.689**	09.70%	-	19.70%	**50.77%**
hibernate-3.1.2	0.530	0.573	0.597	**0.601**	07.23%	-	05.66%	**61.06%**
jgraph-5.12.1.0	0.630	0.764	0.752	**0.798**	25.45%	-	18.18%	**31.20%**
jmeter-2.8.0	0.532	0.637	**0.667**	0.646	14.90%	-	13.07%	**45.44%**
jstock-1.0.7.2	0.552	0.608	0.608	**0.618**	13.39%	-	15.63%	**30.10%**
jung-1.7.5	0.545	0.681	0.756	**0.870**	**30.38%**	-	14.29%	03.95%
junit-4.8.2	0.525	0.560	0.538	**0.581**	13.50%	-	10.00%	**36.80%**
lucene-4.0.0	0.526	0.608	**0.666**	0.641	13.77%	-	21.13%	**34.89%**
weka-3.5.7	0.524	0.588	0.670	**0.700**	15.09%	-	14.66%	**41.35%**
Average	0.547	0.633	0.658	**0.687**	14.15%	-	14.10%	**38.40%**
Improved%	+25.34	+08.53	+04.41	-	+171.42	-	+172.31	-
W/T/L	0/0/14	0/0/14	3/0/11	-	2/0/12	-	2/0/12	-
p-Value	<0.001	>0.05	>0.05	-	<0.001	-	<0.001	-
δ	0.938	0.389	0.122	-	0.796	-	0.816	-

Table V
PERFORMANCE COMPARISON (AUC AND POFB20) OF SCP WITH 2 CROSS-PROJECT DEFECT MODELS (TCA+ AND TDS). THE BEST VALUES ARE IN BOLD. 'W/T/L' INDICATES THE NUMBER OF PROJECT THAT A BASELINE MODEL IS BETTER, EQUAL TO, WORSE THAN SCP.

Dataset	AUC			PofB20		
	TCA+	TDS	SCP	TCA+	TDS	SCP
ant-1.8.3	0.709	0.681	**0.811**	30.00%	29.16%	**46.94%**
antlr-3.2	0.511	0.544	**0.654**	10.47%	61.26%	**64.59%**
argouml-0.32.2	0.520	**0.600**	0.566	11.67%	**49.57%**	49.06%
azureus-4.1.0.4	0.600	0.650	**0.714**	16.17%	**28.03%**	24.07%
freecol-0.10.6	0.565	0.500	**0.723**	20.50%	**17.95%**	17.39%
freemind-0.6.7	0.564	0.652	**0.689**	13.94%	**66.61%**	50.77%
hibernate-3.1.2	0.562	0.542	**0.601**	09.57%	59.88%	**61.06%**
jgraph-5.12.1.0	0.621	0.500	**0.798**	25.55%	23.15%	**31.20%**
jmeter-2.8.0.0	0.524	0.500	**0.646**	13.18%	22.57%	**45.44%**
jstock-1.0.7.2	0.523	0.500	**0.618**	14.46%	**35.07%**	30.10%
jung-1.7.5	0.619	0.776	**0.870**	**25.38%**	06.61%	03.95%
junit-4.8.2	0.521	0.563	**0.581**	12.33%	**38.51%**	36.80%
lucene-4.0.0	0.549	0.576	**0.641**	15.31%	**40.38%**	34.89%
weka-3.5.7	0.540	0.589	**0.700**	14.44%	30.04%	**41.35%**
Average	0.566	0.584	**0.687**	16.57%	36.34%	**38.40%**
Improved%	+21.26	+17.62	-	+131.75	+05.70	-
W/T/L	0/0/14	1/0/13	-	2/0/12	8/0/6	-
p-Value	<0.001	<0.001	-	<0.001	>0.05	-
δ	0.806	0.643	-	0.786	0.122	-

Result 2: *TCA+ is not applicable for change-proneness prediction, while TDS can help developers identify more change-prone classes when they inspect source code line by line. And SCP outperforms TDS for its better source project selection method.*

C. RQ3: Impact of the Similarity Strategies on SCP

Table VI lists the average performance (AUC and PofB20) of SCP with 3 source project selection strategies designed in Section II and their linear combinations (*i.e.* choosing training data according to the addition result of two or more selection strategies), across 14 datasets. We can find that SCP1 achieves the best AUC (0.687) with the second high in PofB20 (38.40%). Although SCP3 and SCP1+SCP3 obtain

Table VI
PERFORMANCE COMPARISON (AVERAGE AUC AND POFB20) OF SCP MODEL WITH DIFFERENT SIMILARITY STRATEGIES FOR SOURCE PROJECT SELECTION ACROSS 14 PROJECTS, WHERE SCP1-3 ARE BASIC STRATEGIES, AND THE REST ARE LINEAR COMBINED STRATEGIES.

Strategy	AUC	PofB20
SCP1	**0.687**	38.40%
SCP2	0.575	27.72%
SCP3	0.617	**39.43%**
SCP1 + SCP2	0.575	27.72%
SCP1 + SCP3	0.617	**39.43%**
SCP1 + SCP2 + SCP3	0.594	36.67%

the best PofB20 (39.43%), this improvement over SCP1 is negligible while their AUC (0.617) are much lower than SCP1. Therefore, the SCP1 strategy is the best one to choose source project for building a classifier in SCP.

Result 3: *To find a better source project for cross-project prediction, the absolute difference of change-proneness rates between source and target projects is the best strategy for SCP.*

D. RQ4: Impact of the Classifier on SCP

To investigate the impact of classifier in SCP, we replace the default classifier Bayesian network with 7 other commonly used classifiers in Table VII. The table shows the average performance of these settings in AUC and PofB20 across 14 datasets. Results indicate that the default classifier, Bayesian Network, achieves the best AUC and PofB20. Therefore, we suggest using the Bayesian network when applying our model.

Result 4: *Compared with 7 commonly used classifiers, Bayesian network is the best one for SCP.*

E. RQ5: Impact of the Number of Source Projects on SCP

Fig. 3 shows the average performance (AUC and PofB20) on SCP trained with different numbers of source projects,

Table VII
PERFORMANCE COMPARISON (AVERAGE AUC AND POFB20) OF SCP WITH DIFFERENT COMMON CLASSIFIERS.

Classifier	AUC	PofB20
LogitBoost	0.622	13.06%
Adaptive Boosting	0.646	34.39%
Alternating Decision Tree	0.650	33.06%
Back Propagation Network	0.628	19.78%
Bayesian Network	**0.687**	**38.40%**
Radial Basis Function Network	0.652	20.11%
Simple Logistic Regression	0.638	17.11%
Support Vector Machine	0.645	16.19%

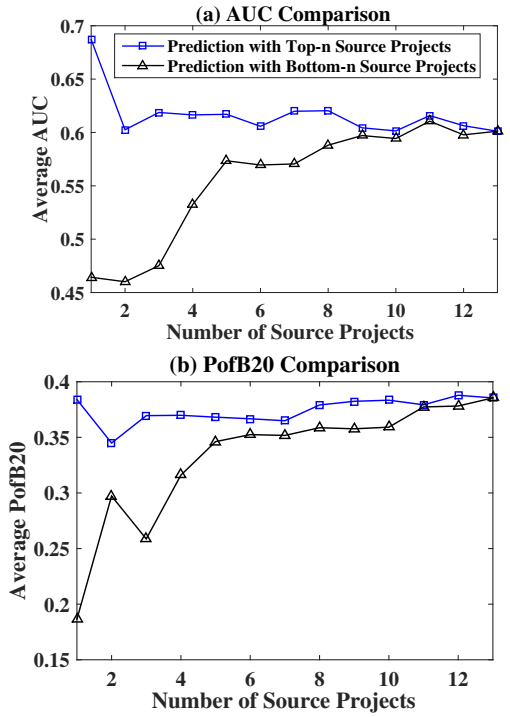

Figure 3. Performance comparison (average AUC and PofB20) of SCP with different number of source projects. For one prediction, we combine the top-n or bottom-n source projects recommended by source project selection method in SCP sequentially.

across 14 datasets. The blue fold line with squares in the figure shows the performance of SCP using top-n source projects recommended by our model, $n \in [1, 13]$. We can observe that AUC and PofB20 have some decreases as we use more source projects, which implies that combining source projects does not have a higher matched distribution compared with the target project. In contrast, the black fold line with triangles in the figure shows a large decrease for AUC and PofB20, when we use the bottom-n, especially $n \in [1, 6]$, source projects recommended by our model. These results suggest that the training sequence of source projects largely affects the performance of SCP. These results may be caused by the project difference in domain, development phase, developers' expertise, and *etc.*

Result 5: *SCP trained with more source projects does not have a better performance. If a classifier is trained with multiple source projects, its performance is sensitive to the training sequence of used source projects.*

F. Threats to Validity

The SCP uses 7 code metrics for model inputs by following [13], [6], [24]. We assume that they are sufficient for change-proneness prediction. However, using more code metrics may achieve better performance, but it is reasonable to simplify the modeling complexity while maintaining acceptable performance. On the other hand, we use linear additivity to combine different similarity strategies to choose source project. But experiment result indicates the linear additivity cannot draw advantages from different strategies. In the future, we will explore a better way to combine similarity measures to improve the model performance.

In the experiment, we evaluate our model by investigating 14 open source projects as [12], but the experiment results on more datasets may be different as all the other empirical evaluations. Moreover, the collected software systems are all open source and written in Java, thus the conclusions may be different for closed source projects, and the projects written in other programming languages. However, the prediction is mainly based on the object-oriented metrics. Therefore, if the programming language is object-oriented and its code metrics are extracted in the same way, this threat could be minimized.

V. RELATED WORK

A. Change-Proneness Prediction

In terms of the used code metrics, the relationship between change-prone class files and code metrics has been studied since decades ago. It is shown that larger sized classes are likely to be change-prone [50]. Later, researchers concluded that a number of Object-Oriented (OO) metrics are strongly correlated with the change-proneness [51], [28], [52] including coupling, cohesion, inheritance, complexity, etc. And the size of code file is a primary indicator, such as the lines of source code [9], [26]. However, the highest value of those OO metrics does not always indicate the change-proneness of a class [26], and the size of code file overestimates the change-proneness [53]. Therefore, researchers started to research on the combination of the code metrics, and validated that the Chidamber and Kemerer (CK) metric suite associated with the size metric is an effective combination to predict change-proneness [54], [55]. Following their research, we use the CK metric suite and size metric as our independent variables.

In terms of the techniques, a few studies first used the traditional statistical approaches for developing change-proneness prediction models [6], [53]. To improve their performance, some other studies found it is effective by using ML techniques for building change-proneness models, such as tree-based models, artificial neural network, support vector machine, and ensemble methods [26], [9], [5], [56],

[57], [10], [11], [58]. However, most of them conduct the prediction by learning from historical labeled data from the target project (*i.e.*, within-project prediction). It is not feasible for new projects or projects with limited historical data. To address this limitation, Malhotra and Bansal [13] proposed cross-project change-proneness prediction that uses labeled data of other projects to conduct the prediction on target data. However, the performance is not stable due to the data distribution difference between source and target projects. The difference of our model and the work of Malhotra and Bansal [13] is that we provide an automatically source project selection method, instead of their random selection. Yan *et al.* [12] proposed a model (CLAMI+) that outperforms the cross-project work of Malhotra and Bansal [13] in terms of AUC. However, we found that it is not feasible in terms of cost-effectiveness. Therefore, we aim to enhance the performance of cross-project prediction performance, in terms of both AUC and cost-effectiveness.

B. Selective Prediction Models

The challenge of cross-project prediction is that the prediction performance is unstable if we randomly choose one alternative project as the source project. To the best of our knowledge, the selective prediction has not been explored in the change-proneness prediction. But a different research problem, defect prediction, supplies some useful ideas.

Previously, Watanabe *et al.* [59] verified the possibility of cross-project prediction by using 2 projects. And they attributed such possibility to the similarity of the domain, programming language, metrics between the projects for model training and testing. Later, Zimmermann *et al.* [23] conducted a large scale cross-project prediction among 12 real-world applications. They found that the cross-project prediction is challenging (only 3.4% predictions worked). To solve this issue, Herbold [19] proposed a training data selection (TDS) method to find a better source project. Specifically, the author uses a nearest neighbor based model to choose the source project closest to the target project from alternative projects for defect prediction. The results indicated the TDS can improve the prediction performance (the success rate improved to 18%).

The TDS is the most similar work to our model SCP. The difference is that our similarity measure considers the label information (change-proneness) of files, where the missed labels of target project are estimated by an unsupervised approach in advance. We make this difference motivated by the empirical finding of He *et al.* [22]. They found that it is possible to achieve much higher performance (success rate can be over 50%), if selecting the source project via a similarity measure on data distribution on condition of their label information.

VI. Conclusion

Predicting change-prone class files for software systems has long been a challenging problem. The cross-project prediction performance is usually unstable because the distribution between a randomly selected source project and a target project is not matching. To tackle this issue, we propose a model named SCP to find the best-matched source project from alternatives. We conduct an empirical study on 14 open source projects to evaluate the effectiveness of SCP, the results show that:

- The SCP is an effective model for cross-project change-proneness prediction compared with 2 related change-proneness prediction models, including a cross-project model RCP and a state-of-the-art model CLAMI+. On average across 14 projects, our SCP improves RCP and CLAMI+ by 25.34% and 4.30% in terms of AUC respectively, and by 171.42% and 172.31% in terms of cost-effectiveness respectively.
- Compared with 2 similar cross-project defect prediction models (TCA+ and TDS), our SCP also achieves significant improvements. On average across 14 projects, SCP improves TCA+ and TDS by 21.26% and 17.62% in terms of AUC respectively, and by 131.75% and 5.70% in terms of cost-effectiveness respectively.
- Among 3 designed strategies for source project selection, the best one is to choose the source project with close rate of change-prone classes to a target project.
- Training a classifier with more selected source projects has negative effect on SCP. And if training a classifier with multiple source projects, and the performance of classifier is sensitive to the training sequence of used source projects. Thus, it is recommended to use the top-1 source project selected by SCP to train a classifier.

Acknowledgment

The work described in this paper was partially supported by the Fundamental Research Funds for the Central Universities of China (No. 106112017CDJXSYY002), the National Natural Science Foundation of China (No. 61772093), Chongqing Research Program of Basic Science & Frontier Technology (No. cstc2017jcyjB0305), and China Postdoctoral Science Foundation (No. 2017M621931).

References

[1] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *JSS*, 2007.

[2] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in *ESEM*, 2009.

[3] Y. Fu, M. Yan, X. Zhang, L. Xu, D. Yang, and J. D. Kymer, "Automated classification of software change messages by semi-supervised latent dirichlet allocation," *IST*, 2015.

[4] M. Yan, Y. Fu, X. Zhang, D. Yang, L. Xu, and J. D. Kymer, "Automatically classifying software changes via discriminative topic model: Supporting multi-category and cross-project," *JSS*, 2016.

[5] R. Malhotra and M. Khanna, "Examining the effectiveness of machine learning algorithms for prediction of change prone classes," in *HPCS*, 2014.

[6] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," *EMSE*, 2012.

[7] J. Al Dallal, "Object-oriented class maintainability prediction using internal quality attributes," *IST*, 2013.

[8] M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," *JSEP*, 2013.

[9] A. G. Koru and H. Liu, "Identifying and characterizing change-prone classes in two large-scale open-source products," *JSS*, 2007.

[10] C. Van Koten and A. Gray, "An application of bayesian network for predicting object-oriented software maintainability," *IST*, 2006.

[11] M. O. Elish, H. Aljamaan, and I. Ahmad, "Three empirical studies on predicting software maintainability using ensemble methods," *Soft Computing*, 2015.

[12] M. Yan, X. Zhang, C. Liu, L. Xu, M. Yang, and D. Yang, "Automated change-prone class prediction on unlabeled dataset using unsupervised method," *IST*, 2017.

[13] R. Malhotra and A. J. Bansal, "Cross project change prediction using open source projects," in *ICACCI*, 2014.

[14] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *TSE*, 2013.

[15] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *ASE*, 2013.

[16] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models," in *FSE*, 2016.

[17] M. Yan, Y. Fang, D. Lo, X. Xia, and X. Zhang, "File-level defect prediction: Unsupervised vs. supervised models," in *ESEM*, 2017.

[18] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *ICSE*, 2013.

[19] S. Herbold, "Training data selection for cross-project defect prediction," in *PROMISE*, 2013.

[20] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *TNN*, 2011.

[21] S. J. Pan and Q. Yang, "A survey on transfer learning," *TKDE*, 2010.

[22] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *ASE*, 2012.

[23] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *FSE*, 2009.

[24] D. Spinellis, "ckjm chidamber and kemerer metrics software," v 1.6. Technical report, Athens University of Economics and Business, 2005. http://-www. spinellis. gr/sw/ckjm, Tech. Rep., 2005.

[25] R. Malhotra and A. Bansal, "Prediction of change prone classes using threshold methodology," *ACSIT*, 2015.

[26] A. G. Koru and J. Tian, "Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products," *TSE*, 2005.

[27] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets (t)," in *ASE*, 2015.

[28] J. M. Bieman, A. A. Andrews, and H. J. Yang, "Understanding change-proneness in oo software through visualization," in *ICPC workshop*, 2003.

[29] R. Malhotra and R. Jangra, "Prediction & assessment of change prone classes using statistical & machine learning techniques." *JIPS*, 2017.

[30] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *TSE*, 2008.

[31] A. Okutan and O. T. Yıldız, "Software defect prediction using bayesian networks," *EMSE*, 2014.

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, 2009.

[33] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble, "The qualitas corpus: A curated collection of java code for empirical studies," in *APSEC*, 2010.

[34] R. Malhotra and M. Khanna, "An empirical study for software change prediction using imbalanced data," *EMSE*, 2017.

[35] X. Xia, E. Shihab, Y. Kamei, D. Lo, and X. Wang, "Predicting crashing releases of mobile applications," in *ESEM*, 2016.

[36] M. Yan, X. Xia, E. Shihab, D. Lo, J. Yin, and X. Yang, "Automating change-level self-admitted technical debt determination," *TSE*, 2018.

[37] X. Yang, D. Lo, X. Xia, and J. Sun, "Condensing class diagrams with minimal manual labeling cost," in *COMPSAC*, 2016.

[38] A. P. Bradley, "The use of the area under the roc curve in the evaluation of machine learning algorithms," *PR*, 1997.

[39] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction," *TSE*, 2017.

[40] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in *FSE*, 2012.

[41] E. Arisholm, L. C. Briand, and M. Fuglerud, "Data mining techniques for building fault-proneness models in telecom java software," in *ISSRE*, 2007.

[42] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *ICSE*, 2013.

[43] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *TSE*, 2016.

[44] X. Yang, D. Lo, X. Xia, and J. Sun, "Tlel: A two-layer ensemble learning approach for just-in-time defect prediction," *IST*, 2017.

[45] Y. Zhang, D. Lo, X. Xia, and J. Sun, "Combined classifier for cross-project defect prediction: an extended empirical study," *FCS*, 2018.

[46] Q. Huang, X. Xia, and D. Lo, "Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction," in *ICSME*, 2017.

[47] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, 1945.

[48] H. Abdi, "Bonferroni and sidak corrections for multiple comparisons. encyclopedia of measurement and statistics. edited by: Salkind nj. 2007."

[49] N. Cliff, *Ordinal methods for behavioral data analysis*, 2014.

[50] M. Lindvall, "Are large c++ classes change-prone? an empirical investigation," *SPE*, 1998.

[51] J. M. Bieman, D. Jain, and H. J. Yang, "Oo design patterns, design structure, and program changes: an industrial case study," in *ICSM*, 2001.

[52] E. Arisholm, L. C. Briand, and A. Foyen, "Dynamic coupling measurement for object-oriented software," *TSE*, 2004.

[53] Y. Zhou, H. Leung, and B. Xu, "Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness," *TSE*, 2009.

[54] S. Eski and F. Buzluca, "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes," in *ICSTW*, 2011.

[55] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness," *IJMLC*, 2013.

[56] E. Giger, M. Pinzger, and H. C. Gall, "Can we predict types of code changes? an empirical analysis," in *MSR*, 2012.

[57] D. Romano and M. Pinzger, "Using source code metrics to predict change-prone java interfaces," in *ICSM*, 2011.

[58] M. Yan, M. Yang, C. Liu, and X. Zhang, "Self-learning change-prone class prediction." in *SEKE*, 2016.

[59] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *PROMISE workshop*, 2008.