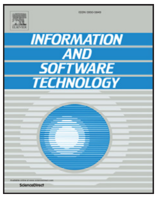
Contents lists available at ScienceDirect

Information and Software Technology

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



A two-phase transfer learning model for cross-project defect prediction

Chao Liu[a,b,*], Dan Yang[b], Xin Xia[c], Meng Yan[d], Xiaohong Zhang[b]

[a] *Key Laboratory of Dependable Service Computing in Cyber Physical Society Ministry of Education, Chongqing University, Chongqing, China*
[b] *School of Big Data & Software Engineering, Chongqing University, Chongqing, China*
[c] *Faculty of Information Technology, Monash University, Melbourne, Australia*
[d] *College of Computer Science and Technology, Zhejiang University, Hangzhou, China*

ARTICLE INFO

Keywords:
Cross-Project prediction
Defect prediction
Transfer learning
Source project selection

ABSTRACT

Context: Previous studies have shown that a transfer learning model, TCA+ proposed by Nam et al., can significantly improve the performance of cross-project defect prediction (CPDP). TCA+ achieves the improvement by reducing data distribution difference between source (training data) and target (testing data) projects. However, TCA+ is unstable, i.e., its performance varies largely when using different source projects to build prediction models. In practice, it is hard to choose a suitable source project to build the prediction model.

Objective: To address the limitation of TCA+, we propose a two-phase transfer learning model (TPTL) for CPDP.

Method: In the first phase, we propose a source project estimator (SPE) to automatically choose two source projects with the highest distribution similarity to a target project from candidates. Next, two source projects that are estimated to achieve the highest values of F1-score and cost-effectiveness are selected. In the second phase, we leverage TCA+ to build two prediction models based on the two selected projects and combine their prediction results to further improve the prediction performance.

Results: We evaluate TPTL on 42 defect datasets from PROMISE repository, and compare it with two versions of TCA+ (TCA+_Rnd, randomly selecting one source project; TCA+_All, using all alternative source projects), a related source project selection model TDS proposed by Herbold, a state-of-the-art CPDP model leveraging a log transformation (LT) method, and a transfer learning model Dycom with better form of TCA. Experiment results show that, on average across 42 datasets, TPTL respectively improves these baseline models by 19%, 5%, 36%, 27%, and 11% in terms of F1-score; by 64%, 92%, 71%, 11%, and 66% in terms of cost-effectiveness.

Conclusion: The proposed TPTL model can solve the instability problem of TCA+, showing substantial improvements over the state-of-the-art and related CPDP models.

1. Introduction

Software defect prediction can help software assurance teams effectively allocate testing resources by predicting defect-prone class files in projects prior to the testing phase [25,35]. Recent studies have shown that machine learning (ML) models can successfully identify defective classes/files/modules (aka., instances) under a within-project defect prediction (WPDP) setting [4,8,20,38], i.e., training and testing data are extracted from the same project. However, in practice, WPDP cannot work well to a new project, since a new project has no or not sufficient training data to train an ML model, and the cost of manually identifying whether the classes/files/modules are defective is high [39].

An alternative and promising solution is the cross-project defect prediction (CPDP), which trains an ML model by using plenty of training data from other projects (aka., source projects) [18,23,28,29,39]. For example, we can obtain training data from the PROMISE repository that provides many publicly released defect prediction datasets [16]. CPDP, however, is a challenging task as its prediction performance is usually not high. This is because an ML model trained by one or a set of projects (i.e., source projects) cannot generalize well to other projects (i.e., target projects) [23], due to domain difference phenomenon between the source and target projects, e.g., architecture, programming language, coding styles, and developer experience.

To conquer the challenge existed in CPDP, many transfer learning approaches, which aim to adapt the domain difference between the source and target projects, have been proposed [18,23,28]. TCA+ is a state-of-the-art transfer learning algorithm for CPDP [23], which can minimize the data distribution difference between the source and target projects. Specifically, TCA+ maps source and target projects into a shared latent

[*] Corresponding author.
E-mail addresses: liu.chao@cqu.edu.cn (C. Liu), dyang@cqu.edu.cn (D. Yang), xin.xia@monash.edu (X. Xia), mengy@zju.edu.cn (M. Yan), xhongz@cqu.edu.cn (X. Zhang).

https://doi.org/10.1016/j.infsof.2018.11.005
Received 10 December 2017; Received in revised form 13 September 2018; Accepted 19 November 2018
Available online xxx

space where their data distribution distance is close [27]. In this way, an ML model trained by transformed source project can better identify defective instances on transformed target project. However, TCA+ is unstable and its performance is largely determined by the selection of source project. Our preliminary investigation shows that carefully choosing one source project, instead of randomly using one or a set of source projects, can substantially improve the performance of TCA+ (Section 2).

To address the limitation of TCA+, in this paper, we propose a two-phase transfer learning model (TPTL) for CPDP. In the first phase, we propose a source project estimator (SPE) to automatically choose two source projects with the highest distribution similarity to a target project from a set of candidate source projects. SPE is inspired by a Training Data Selection (TDS) method proposed by Herbold [11]. TDS is an unsupervised approach, which directly measures the Euclidean distance between the source and target projects. Different from TDS, SPE is a supervised approach, and we build SPE as two regression models, which predict the values of F1-score and cost-effectiveness on the target project based on the data distribution characteristics collected from the source and target projects, respectively. In this study, F1-score represents the discriminatory power of a model, which is a harmonic average of precision and recall; and PofB20, a widely used cost-effectiveness metric, measures the percentage of bugs that a developer can identify by inspecting the top 20% lines of code as sorted by the defective likelihood score of our model [15] (details show in Section 4.2). Next, two source projects which are estimated to achieve the highest values of F1-score and PofB20 on target project are selected. In the second phase, we leverage TCA+ to build two prediction models based on the two selected projects, and combine their prediction results to further improve the prediction performance.

We evaluate our TPTL model against two related CPDP models, TCA+ [23] and TDS [11] by investigating 42 defect datasets, including a total of 16,443 instances (classes), from the PROMISE data repository [21]. We implement TCA+ in two versions: TCA+ trained with one randomly selected source project (TCA+_Rnd), and TCA+ trained with all alternative source projects (TCA+_All). We also compare TLTP with a state-of-the-art CPDP model that leverages a log transformation (LT) method [3]. Furthermore, TPTL is compared with a transfer learning model Dycom [22] with a better form of TCA that previously shows excellent results in the web effort estimation dataset.

Experiment results show that TPTL achieves the best performance in F1-score (0.481) and PofB20 (0.197). On average across 42 datasets, TPTL respectively improves TCA+_Rnd, TCA+_All, TDS, LT, and Dycom by 19.28%, 4.98%, 36.12%, 27.13%, and 11.08% in terms of F1-score; by 64.15%, 91.60%, 71.01%, 11.27%, and 65.73% in terms of PofB20, respectively. Results indicate that TPTL can solve the instability problem of TCA+, showing substantial improvements over the state-of-the-art and related models. Moreover, the experiment shows that TPTL needs about 55 min and 0.001 s for model building and predicting respectively, on average across 42 datasets, where 99.88% of building time are used for collecting training data for SPE. As SPE does not need to be updated all the time, TPTL thus has an acceptable execution time for practical use.

The main contributions of this article are:

- We propose an improved CPDP model called TPTL, which utilizes the advantages of source project selection and transfer learning algorithm to build a two-phase transfer learning model.
- We evaluate our TPTL model with previously succeeded CPDP model TCA+ [23], a related source selection model TDS [11], a state-of-the-art CPDP model LT [3], and an excellent transfer learning model Dycom [22] originally used for web effort estimation. Across 42 defect datasets, the experiment results show that the proposed model can substantially improve the performance of CPDP comparing to the baseline models.

Table 1
Experiment results using TCA+ with different source project, where 'All' indicates a prediction using a combination of all other 25 source projects. We use poi-2.5 as the target project.

Source	F1-Score	Source	F1-Score
ant-1.3	0.301	synapse-1.1	0.255
ant-1.4	0.415	synapse-1.2	0.713
ant-1.5	0.467	velocity-1.4	0.312
ant-1.6	0.424	velocity-1.6	0.612
ant-1.7	0.376	tomcat	0.460
log4j-1.0	0.454	xalan-2.4	0.759
log4j-1.1	0.507	xalan-2.5	0.683
log4j-1.2	0.488	xalan-2.6	0.514
lucene-2.0	0.622	xalan-2.7	0.716
lucene-2.2	0.731	xerces-1.2	0.625
lucene-2.4	0.785	xerces-1.3	0.642
redaktor	0.706	xerces-1.4	0.766
synapse-1.0	0.505	**All**	0.285

In the remainder of this article, we describe preliminary work on transfer learning and the motivation of building our TPTL model in Section 2. We present the implementation details of TPTL in Section 3. We describe the experiment settings in Section 4. We discuss our experiment results in Section 5, and threats to validity in Section 6. We briefly review related work in Section 7. We finally present the summary and outlook of this work in Section 8.

2. Preliminary and motivation

In this section, we describe the preliminary on transfer learning and the motivation to build TPTL.

2.1. Preliminary on transfer learning

The challenge of CPDP mainly results from the difference of data distribution between the source and target projects [23,39]. For example, a newly created project may have many defects in large-sized classes due to its complexity and importance, while a stable project may have more defects in small-sized classes for some recently updated functions. In this way, a model trained by one project cannot correctly identify defective files in the other project.

To overcome the data distribution difference between source and target projects, TCA+ (transfer component analysis plus) is developed [23]. TCA+ is a previously succeeded CPDP model, an extended model of TCA [27]. To reduce the distribution difference between source and target projects, TCA maps them into a shared latent space, where their data distribution distance is close [27]. In this way, a classifier trained by the transformed source project can better identify defective classes on transformed target project. To further minimize data distribution difference and suppress the effect of outliers, Nam et al. [23] incorporate TCA with a data pre-processing method. Details can be found in their work by Nam et al. [23].

2.2. Why two-phase transfer learning?

Although TCA+ can improve the performance of CPDP, it encounters a new problem: how should TCA+ chooses a suitable source project from candidates, randomly selecting one or combining all available source projects?

To investigate this question, we evaluate the performance, F1-score[1], of TCA+ on 26 projects[2] from the PROMISE repository [21] as listed in Table 1. In the prediction, we use poi-2.5 as the target project. Other

[1] The definition of F1-score is described in Section 4.2.
[2] Details of studied projects are discussed in Section 4.1.

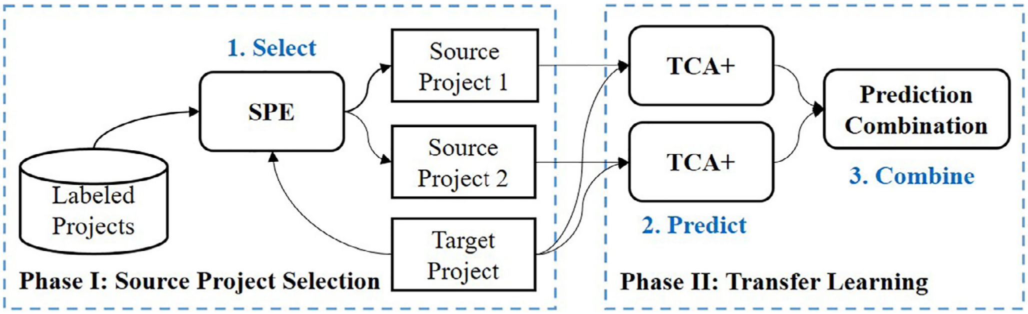


Fig. 1. Overall architecture of TPTL.

25 projects are separately used or combined together (All) as a source project. We choose logistic regression[3] as the classifier for TCA+.

From Table 1, we can observe that the predictions by TCA+ are unstable if we train TCA+ with only one source project, F1-scores vary from 0.255 to 0.785. Thus, randomly choosing one source project may lead to poor performance for TCA+ [26]. However, using all candidate source projects does not necessarily have better performance (F1-score = 0.285). These phenomena are also discovered by Turhan [34], which is called the dataset shift problem. To solve this problem, we present a two-phase model TPTL: it first chooses two source projects with similar data distribution to a target project; and then leverage TCA+ to build two prediction models based on the two selected projects, and combine their prediction results to further improve the prediction performance.

3. Proposed model

In this section, we first describe the overall architecture of TPTL, then present the implementation details of its components.

3.1. Overall architecture

Fig. 1 illustrates the overall architecture of the proposed model TPTL, which works in two phases: a source project selection phase, and a transfer learning phase.

Phase-I: Source Project Selection Phase. Specifically, the input of TPTL is a collection of candidate source projects, and a source project contains many class files. In the source projects, each class is attached with a known label (i.e., defective or clean), and a set of software metrics as shown in Table 2. In this paper, we use the metrics proposed by Jureczko and Madeyski [16]. In the target project, we extract the same metrics as in the source projects, but without the label. In this phase, SPE selects two candidate source projects (Step 1), and they are estimated to achieve the highest F1 and PofB20 scores on the target project. The details of SPE is presented in Section 3.2.

Phase-II: Transfer Learning Phase. In the second phase, we build two TCA+ models for two pairs of the selected source and target projects (Step 2). These two TCA+ models respectively aim to achieve high F1 and PofB20 scores on the target project. We further combine these two TCA+ models to improve the prediction results (Step 3).

3.2. Source project estimator (SPE)

Herbold [11] found that if data distributions between the source and target projects are close, a CPDP model can achieve better prediction performance. Thus, the goal of SPE is to select the source projects which have similar data distributions as the target project.

In this paper, SPE learns two regression models by leveraging a number of independent variables (i.e., median values of all software metrics in a project) extracted from the source projects – one is used to predict the F1-score on the target project, and another is used to predict the PofB20 score on the target project. Next, we select two source projects which respectively achieve the highest predicted scores of F1 and PofB20 on the target project. Fig. 2 presents the details of the learning process of the two regression models. They are learned by using independent and dependent variables (in our case, F1 and PofB20 scores) extracted from source projects.

Table 2
Metrics for defect prediction used by Jureczko and Madeyski [16].

Metric	Description
wmc	the number of methods declared in a given class.
dit	the depth of inheritance hierarchy for a given class.
noc	the number of children inherited to a given class.
cbo	the number of classes coupled to a given class.
rfc	the number of distinct methods called in a given class.
lcom	the number of method pairs in a class that do not share access to any class attributes.
lcom3	another type of lcom metric proposed by Henderson [10].
npm	the number of public methods in a given class.
loc	the number of lines of code in a given class.
dam	the ratio of the number of private/protected attributes to the total number of attributes in a given class.
moa	the number of attributes in a given class which are of user-defined types.
mfa	the number of methods inherited by a given class divided by the total number of methods that can be accessed by the member methods of the given class.
cam	the ratio of the sum of the number of different parameter types of every method in a given class to the product of the number of methods in the given class and the number of different method parameter types in the class.
ic	the number of parent classes coupled to a given class.
cbm	the total number of new or overwritten methods that all inherited methods in a given class are coupled to.
amc	the average size of methods in a given class.
ca	the number of other classes that invoke a given class.
ce	the number of classes invoked in a given class.
max_cc	the maximum McCabe's cyclomatic complexity (CC) score of methods in a given class.
avg_cc	the arithmetic mean of McCabe's cyclomatic complexity (CC) score of methods in a given class.

3.2.1. Regression model construction

As shown in Fig. 2, to learn two regression models, we need pairs of source and target projects to generate values for independent and dependent variables. But in practices, we only have one target project, and it does not have labels to evaluate the prediction performance. To solve this problem, we use different combinations of labeled source projects as the hypothetical source and target projects. For example, if we have 2 labeled source projects (A and B), then we have 2 combinations of training data (AB and BA) for regression models.

Independent Variables. The independent variables of SPE include not only a hypothetical source project but also a hypothetical target project, because their data distribution similarity affects the prediction performance of CPDP [9,11,23]. We represent the features of a hypothetical source and target projects by a characteristics vector. Elements

[3] Description on the implementation is provided in Section 3.3.

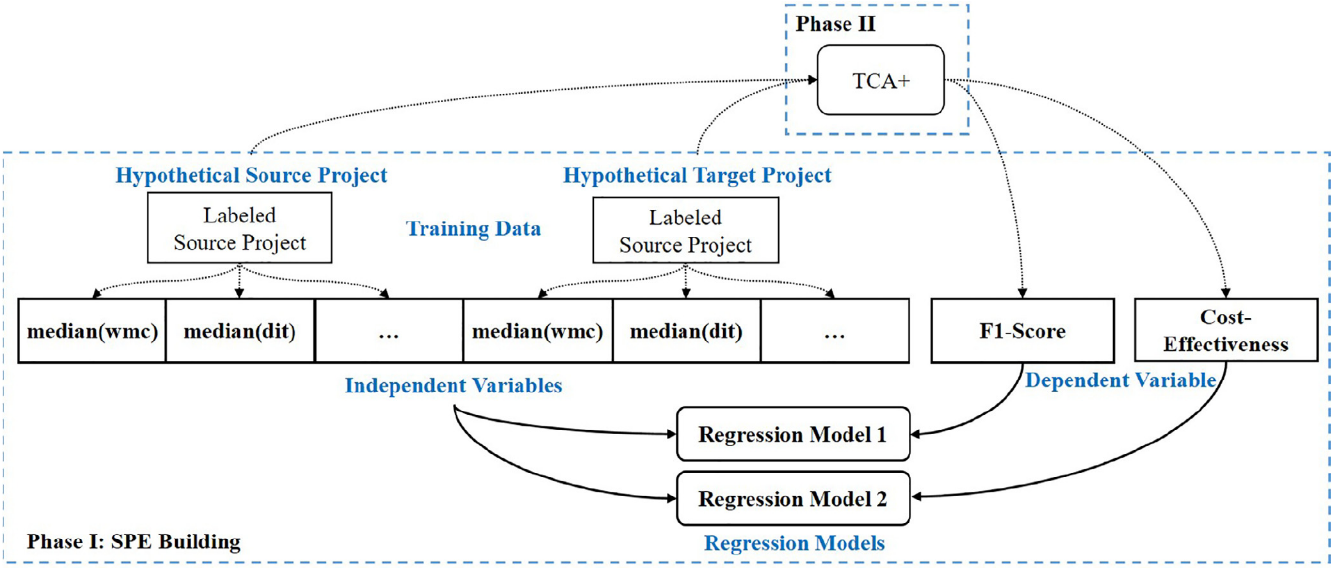


Fig. 2. Working flow to build SPE.

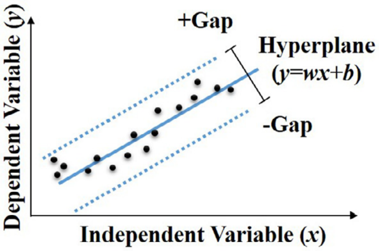


Fig. 3. Illustration of the SMOreg.

in the vector are median values of all software metrics in both projects as illustrated in Fig. 2. As each project is represented by 20 metrics as listed in Table 2, there are a total of 40 elements in the vector totally. Experiment results in Section 5.2 shows that the median values are enough to capture the features of source and target projects, and using more distribution characteristics like Herbold [11], such as mean and variance, does not help SPE find a better source project.

Dependent Variable. The dependent variable of SPE is the prediction performance of TCA+ using a pair of hypothetical source and target projects in Phase-II, as shown in Fig. 2. We build two regression models in this study because we have two types of dependent variables, i.e. two evaluation metrics (F1-score or PofB20).

Regression Models. In default, we use the well-known supervised algorithm SMOreg (Sequential Minimal Optimization for regression) as the two underlying regression models. The SMOreg is a kind of support vector machine (SVM) model but designed for the regression problem, which is trained by the sequential minimal optimization method [32]. For our research, as illustrated in Fig. 3, SMOreg aims to learn a hyperplane with two short gaps ($\pm Gap$) to capture the relationship from independent variables to the dependent variable, so that the learned hyperplane can be used to estimate the distribution similarity between two projects. We implement SMOreg by invoking Weka library [7] with default settings.

3.2.2. Source project selection

After the two regression models are trained, given a source and a target project, SPE first extracts the same independent variables as shown in the regression model construction step. Next, SPE inputs these independent variables into two developed regression models, and each model will get a result (i.e., F1 or PofB20 score). Finally, SPE selects two source projects which are estimated to achieve the highest F1 and PofB20 scores on the target project, respectively.

3.3. TCA+

TCA (Transfer Component Analysis). To reduce the data distribution distinction between a pair of source and target projects, TCA learns a nonlinear function ϕ to map the source project $x_i \in \mathbf{X}_{train}$ and the target project $x_j \in \mathbf{X}_{test}$ into a latent space. In the latent space, TCA requires the distribution distance (Dist) between two projects to be close [27]. Thus, the objective of TCA is to minimize the distance represented in Eq. (1), where p and q indicate the total number of class files in the source and target projects, respectively. Pan et al. [27] provide an implicit way to obtain the transformed source and target projects in details.

$$Dist(\mathbf{X}_{train}, \mathbf{X}_{test}) = \left\| \frac{1}{p}\sum_{i=1}^{p} \phi(x_i) - \frac{1}{q}\sum_{i=1}^{q} \phi(x_j) \right\|_{\mathcal{H}}^{2} \quad (1)$$

Data Preprocessing. We first process the metric values for a pair of source and target projects via a log transformation method following [3]. Moreover, Nam et al. [23] observed that incorporating TCA with normalization method (e.g., min-max or z-score normalization) can improve the performance of CPDP, but the performance is sensitive to the type of normalization approaches and the distribution characteristics of the source and target projects. Therefore, they propose a heuristic method to choose a suitable normalization method for TCA, according to the distribution features of the source and target projects. Implementation details can be found in their work [23]. They call TCA extended with their heuristic method as TCA+.

Classifier. TCA+ also involves an underlying classifier, which finally determines the prediction performance of TPTL. Following Nam et al. [23], we use the logistic regression as the classifier for its better CPDP. The logistic regression is used to estimate the probability of a binary response (i.e., defective or clean) based on multiple independent variables (i.e., 20 used software metrics in Table 2) [13]. We adopt the same implementation library (LIBLINEAR[4]) and parameters settings ("-S0", use logistic regression; "-B -1", use no bias term) following Nam et al. [23] For each class in the target project, the classifier will output the confidence score that the class is buggy.

[4] LIBLINEAR library: http://www.csie.ntu.edu.tw/~cjlin/liblinear/

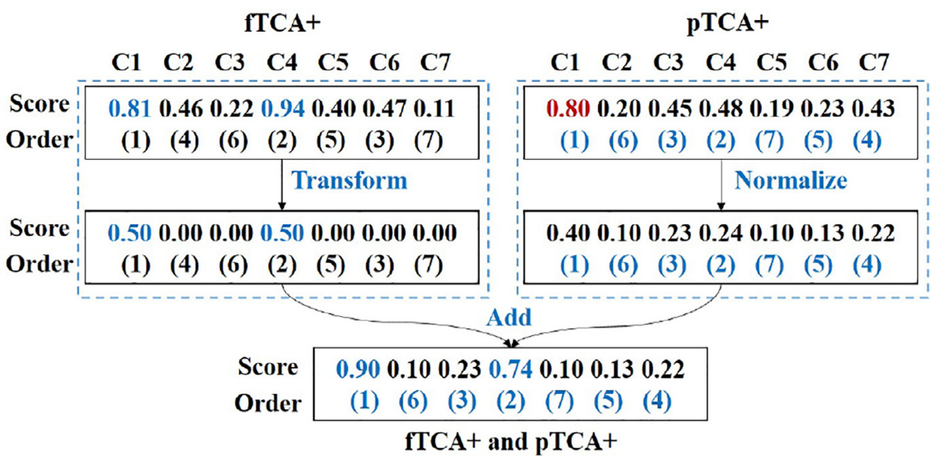


Fig. 4. Example of the prediction combination with 7 classes (C1–C7). The confidence scores predicted by fTCA+ are transformed, and added with the normalized scores (divided by 2) by pTCA+, to maintain the discriminability of fTCA+ and the order of pTCA+.

3.4. Prediction combination

Our approach builds two TCA+ models for the two selected source projects, and our experiments find that each TCA+ model only achieves a high F1 or PofB20 score on the target project (we respectively call them fTCA+ and pTCA+ hereafter)[5]. Thus, we need to combine these two TCA+ models to improve both of the F1 and PofB20 scores, since they are the most important evaluation metrics in defect prediction.

By default, the fTCA+ model predicts a class as buggy if the confidence score is more than a threshold of 0.5, else it predicts the class as clean. A fTCA+ model with a high F1-score indicates that it could identify the buggy classes accurately, i.e., a class with a confidence score of more than 0.5 is more likely to be buggy. The pTCA+ model with a high PofB20 score implies that the top-ranked classes are more likely to be buggy, and the ranking list is built based on the confidence scores output by the pTCA+ model.

To take the advantages of two TCA+, we combine their results in two steps: (1) to maintain the discriminability advantage of fTCA+, we first turn its predicted values higher than 0.5 to be 0.5, otherwise to be 0; (2) to incorporate the sequence generated by pTCA+ to fTCA+, we normalize the values predicted by cTCA+ to the range of 0 to 0.5 (dividing all values by 2), and then add the normalized values by pTCA+ to the transformed values by fTCA+. Fig. 4 exemplifies the way we combine confidence scores of 7 classes (C1-C7) predicted by fTCA+ and pTCA+, and how we maintain the features of two lists. In the example, the combination method cannot only predict two classes are defective as fTCA+ but also reorder classes as pTCA+.

4. Experiment

In this section, we present the experiment setup, evaluation metrics, and research questions for model evaluation.

4.1. Experiment setup

Datasets. We evaluate TPTL using defect datasets collected by Jureczko and Madeyski [16] from the PROMISE data repository [21], as listed in Table 3. The datasets contain 42 releases from 14 distinct open-source Java projects. Each instance in a dataset corresponds to a Java class. And one instance is represented by 20 static code metrics as shown in Table 2, and is also attached by a label, i.e. defective or clean. Table 3 shows the total number of classes and defects in these 42 defect datasets.

Baseline Models. In the study, we first compare the proposed TPTL model with 2 previously succeeded CPDP models including TCA+ [23], TDS [11]. TDS is the most related work for source project selection but without the process of TCA+, and we re-implement TDS according to the pseudo-code written in the original paper. TCA+ is an extended algorithm of TCA with some data preprocessing work. We develop TCA part following Pan et al. [27] and the extended part as Nam et al. [23].

Moreover, we compare TPTL with the model LT [3], a state-of-the-art multi-source CPDP model leveraging a median based metric transformation [12]. In specific, for n combined source projects X and a target project Y, LT first preprocesses those data by log transformation, $\widehat{X} = log(1+X)$, $\widehat{Y} = log(1+Y)$. To mitigate the difficulty of cross-project prediction, LT aligns the median of i-th metric of the transformed source project $\widehat{X}_j$ to the median of transformed target project by $\widehat{X}_i = \widehat{X}_i + median(\widehat{X}_i) - median(\widehat{Y}_i)$. In this way, a cross-project model built on all transformed source project $\widehat{X}$ can effectively predict the labels of the transformed target project $\widehat{Y}$. In the prediction, LT uses the decision tree C4.5 as its classifier following [12], which is implemented by invoking the Weka tool [7] with default settings.

In addition, we compare TPTL with the transfer learning model Dycom with a more advanced form of TCA, which achieves excellent results in web effort estimation [22]. Dycom is a weighted sum of n cross-project regression models (f_i, $i \in [1, n]$) and a within-project model (f_{n+1}) built on 10% labeled data x from target project. For a cross-project model f_i, it is transformed with a linear function g_i pre-trained by target data x, so that the transferred cross-project model $g_i(f_i)$ is close to a within-project model, i.e., $g_i(f_i(x)) \approx f_{n+1}(x)$. We implement Dycom as the same model learning strategy as described in Minku et al. [22]. Additionally, due to the difference of web effort estimation (i.e., regression problem) and software defect prediction (i.e., classification problem), we use our underlying classifier (logistic regression) to replace the regression model (regression trees) used in Dycom.

Note that, following Nam et al. [23], all of the baseline models except LT use logistic regression as their underlying classifiers. The logistic regression is implemented as our TPTL model, which is described in Section 3.3.

Source and Target Projects. To simulate the practical model usage, we perform the cross-project prediction as previous studies [23,30,31]. Specifically, for one defect prediction, we pick up one release of a project as a target project (e.g., ant-1.3), and use all releases of other projects as the candidate source projects (i.e., 37 releases not belonged to the ant in the example). We thus have 42 cross-project predictions in total.

4.2. Evaluation metrics

In this study, we adopt two widely used performance metrics, F1-score and cost-effectiveness, to evaluate the proposed model.

F1-Score. When predicting defective classes in a target project, a model may succeed (True Positive, *TP*) or fail (False Positive, *FP*) to

[5] For more details, please refer to Table 11.

Table 3
Studied software systems. For each system, #Class means the number of class files; #Defect indicates the number of defective classes; %Defect indicates the percentage of defective classes.

Dataset	#Class	#Defect	%Defect	Dataset	#Class	#Defect	%Defect
ant-1.3	125	20	16.00	lucene-2.0	195	91	46.67
ant-1.4	178	40	22.47	lucene-2.2	247	144	58.30
ant-1.5	293	32	10.92	lucene-2.4	340	203	59.71
ant-1.6	351	92	26.21	poi-1.5	237	141	59.49
ant-1.7	745	166	22.28	poi-2.0	314	37	11.78
camel-1.0	339	13	03.83	poi-2.5	385	248	64.42
camel-1.2	608	216	35.53	poi-3.0	442	281	63.57
camel-1.4	872	145	16.63	redaktor	176	27	15.34
camel-1.6	965	188	19.48	synapse-1.0	157	16	10.19
ckjm	10	5	50.00	synapse-1.1	222	60	27.03
ivy-1.1	111	63	56.76	synapse-1.2	256	86	33.59
ivy-1.4	241	16	06.64	tomcat	858	77	08.97
ivy-2.0	352	40	11.36	velocity-1.4	196	147	75.00
jedit-3.2	272	90	33.09	velocity-1.6	229	78	34.06
jedit-4.0	306	75	24.51	xalan-2.4	723	110	15.21
jedit-4.1	312	79	25.32	xalan-2.5	803	387	48.19
jedit-4.2	367	48	13.08	xalan-2.6	885	411	46.44
jedit-4.3	492	11	02.24	xalan-2.7	909	898	98.79
log4j-1.0	135	34	25.19	xerces-1.2	440	71	16.14
log4j-1.1	109	37	33.94	xerces-1.3	453	69	15.23
log4j-1.2	205	189	92.20	xerces-1.4	588	437	74.32

Table 4
Cliff's delta and the effectiveness level [5].

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

predict a defective class, truly (True Negative, *TN*) or wrongly (False Negative, *FN*) identify a clean class. Based on these 4 possible results, precision and recall can measure two aspects of the prediction performance. Specifically, precision equals to the proportion of correctly predicted instances that are defective, $Precision = TP/(TP + FP)$; recall is the proportion of defective instances that are correctly predicted, $Recall = TP/(TP + FN)$. To comprehensively evaluate a CPDP model, the F1-score is widely used [15,17,23], which is a harmonic mean of the precision and the recall, $F1 - score = (2 \times Precision \times Recall)/(Precision + Recall)$.

Cost-Effectiveness. Cost-effectiveness is important because after a model predicts defective classes in a project, developers need to inspect buggy classes one by one, and they expect to find more bugs by reading fewer lines of code [15]. PofB20 is a widely used cost-effectiveness measure proposed by Jiang et al. [2,15,30,31]. It measures the percentage of bugs that a developer can identify by inspecting the top 20% lines of code [15]. A CPDP model with high cost-effectiveness can save developers' efforts for code inspection.

We calculate PofB20 as Jiang et al. [15]. We first sort instances in a target project based on their confidence scores (i.e., the probability to be defective predicted by a CPDP model), where an instance with higher confidence score is more likely to be defective. We then inspect one instance at a time from the highest confidence score to the lowest, and count the total lines of code that have been reviewed. When we inspect 20% of total lines of code in testing data, we stop the inspection, and the PofB20 is equal to the final percentage of bugs that are identified.

4.3. Research questions

The core objective of this study is to investigate whether our TPTL model can solve the instability problem of TCA+. In the experiment, we address the following research questions.

4.3.1. RQ1: how effective is our TPTL model? how much improvement can TPTL achieve over the related models?

We investigate the extent that TPTL model advances 4 related models, TCA+, TDS, LT, and Dycom. In this RQ, we develop two versions of TCA+ with different source project selection approaches: TCA+_All, combining all candidate source projects as an input; TCA+_Rnd, randomly selecting one source project as the input. Furthermore, as Dycom needs to be adjusted by 10% labeled data from the target project, it is also run by 10 times to suppress the effects of randomness. Note that, to simplify the comparison of Dycom with other models, it performs predictions on all data in the target project (including the 10% labeled data). Although this may overestimate the performance of Dycom, it would not affect the conclusion of the comparison.

To check if the performance difference between a baseline model and TPTL is statistically significant, we apply the Wilcoxon signed-rank test [37] at a 95% significance level on their performance (F1-score or PofB20) of 42 defect datasets. To counteract the results of multiple comparisons among baselines, we also use Bonferroni correction [1] to correct the p-values generated by the Wilcoxon signed-rank test. Moreover, we use Cliff's delta (δ, a non-parametric effect size measure) [5] to quantify the amount of difference between two models. As shown in Table 4, δ ranges from -1 to 1, and its value range is divided into four effectiveness levels, where a higher level indicates that two model has a greater difference on a performance [5].

4.3.2. RQ2: how TPTL components affect its prediction performance?

What kinds of statistical metrics can better capture the distribution similarity between source and target projects in SPE? By default, we adopt the median values of all software metrics from source and target projects as the independent variables of SPE in Section 3.2. Apart from the median, other statistical metrics (e.g., mean) can be also used [11]. Hence, we investigate whether other statistical metrics (i.e., mean, variance, skewness, and kurtosis) affect the performance of TPTL individually. We also analyze whether TPTL can be improved by using combined characteristics.

Does the choice of different regression models in SPE affect the performance of TPTL? SPE consists of two regression models. We use SMOreg as the underlying regression models in default, but using a different algorithm may improve the performance of TPTL. Therefore, we apply 8 common regression models to TPTL, as shown in Table 5, to in-

Table 5
Overview of 8 regression models for SPE.

No.	Model name	Abbreviation
1	Sequential Minimal Optimization for regression	SMOreg
2	Additive Regression	AR
3	Decision Table	DT
4	Least Squares Regression	LSR
5	Linear Regression	LR
6	Radical Basis Function network	RBF
7	Simple Linear Regression	SLR
8	ZeroR	ZR

vestigate which algorithm is the best choice for two regression models. We implement them by invoking Weka library [7] with default settings.

Does the number of source projects in SPE affect the performance of TPTL? SPE uses different combinations of labeled source projects to train two regression models. We aim to analyze if the performance of TPTL is proportional to the number of training data. If so, our TPTL models can be further improved by training SPE with more data. In specific, we train SPE in TPTL with m randomly selected training data, where m ranges from 100 to 1300 with step 200. To suppress the effect of randomness, we run each experiment 10 times, and report their mean and standard deviation values.

What is the benefit of the prediction combination? As referred to in Section 3.4, we combine prediction results of two TCA+ (pTCA+ and fTCA+), and we aim to investigate whether the combined prediction is better than the results generated by pTCA+ and fTCA+ respectively, and why we need two TCA+.

4.3.3. RQ3: how efficient is the TPTL model? how much time does it take for TPTL model to run compared to the baseline models?

This RQ investigates the time efficiency of TPTL comparing to the 5 baseline models including TCA+_Rnd, TCA+_All, TDS, LT, and Dycom as referred to in RQ1. We run all the models on a 64-bit, Ubuntu 14, server with 2.10 GHz CPU and 64 GB RAM. As TCA+_Rnd, and Dycom incorporate randomness in model training, we therefore run them 10 times, and record the average model building and predicting time for each dataset. And we finally compare average model building and predicting time across 42 datasets for TPTL and other baselines.

5. Results and discussion

5.1. RQ1: Effectiveness of TPTL

Prediction Performance. Tables 6 and 7 show the F1-score and PofB20 values of TPTL versus 5 baseline models. TPTL gains the best performance on F1-score, whose F1-scores range from 0.050 to 0.787 with average 0.481 across 42 datasets. Table 6 shows that our model TPTL achieves substantial improvements over baseline models. On average across the 42 defect datasets, TPTL outperforms TCA+_Rnd, TCA_All, TDS, LT, and Dycom by 19.28%, 4.98%, 36.12%, 27.13%, and 11.08% in terms of F1-score, respectively. We notice that four of the studied projects (xerces, xalan, synapse, and poi) show larger improvements in the last version. One possible reason is the latest version contains more classes and defective classes as shown in Table 3, so that the defective pattern on a larger-size version is easier to be captured.

TPTL also achieves the best cost-effectiveness in terms of PofB20. The PofB20 scores of TPTL vary from 0.027 to 0.450 with average 0.197. Table 7 shows that the improvements of our model TPTL over the baseline models are substantial. Across 42 defect datasets, the average PofB20 of TPTL outperforms those of TCA+_Rnd, TCA+_All, TDS, TL, and Dycom by 64.15%, 91.60%, 71.01%, 11.27%, and 65.73%, respectively.

We can observe that for our model PofB20 shows larger improvements than F1-score. A prediction model with a high F1-score means that the model has the ability to distinguish defective classes from clean classes, while the model with a high PofB20 score means that there will be more defective classes in the top-ranked classes. In practice, due to a tight schedule and limited resources, developers might check the top-ranked classes to identify as many defects as possible. Meanwhile, we can notice that the best average F1-score (0.443) and PofB20 (0.119) among baselines are not high enough, thus TPTL's relative improvements over them (+11.08% and +11.27%) are high. Note that the high relative improvements are mainly due to the low performance of the baseline approaches.

Number of Improvements. Moreover, the "W/T/L" row in Tables 6 and 7 reports the number of datasets for which a baseline model obtains a better, equal, and worse performance than our TPTL model. Comparing with all baseline models, results show that TPTL achieves more than 24 improvements in terms of F1-score, and larger than 25 improvements in terms of PofB20.

Statistical Test. Tables 6 and 7 also present the p-values (tested by the Wilcoxon signed-rank test and corrected by the Bonferroni correction) and Cliff's delta (δ) when comparing TPTL with the baseline models in terms of F1-score and PofB20 scores. We can observe that comparing with baselines on F1-score, TPTL shows significant improvements (p-value < 0.05) over TDS and LT with medium size effect (δ=0.365 and 0.340 respectively), and has no significant improvements on TCA+_Rnd, TCA+_All, and Dycom. As to the PofB20, TPTL shows significantly improvements (p-value < 0.05) on all baselines with large size effect ($0.582 \le \delta \le 0.72$) except for the LT.

Above results imply that TCA+ trained with all candidate source projects (TCA+_All) obtains higher performance than that with randomly selected source projects (TCA+_Rnd), with largely enhanced F1-score and slightly reduced PofB20. But we cannot ensure that combining source projects at hand will always gain a better performance as referred in our case study shown in Section 2. And by carefully choosing source projects from candidates, TPTL achieves substantial improvements over TCA+_All and TCA+_Rnd. Thus, the proposed model TPTL can solve the instability issue of TCA+ with the well-chosen source project.

Besides, the TPTL's advantage over TDS indicates that although a suitable source project can be selected by measuring its metric difference with target project, a better substitute is to learn their relationship as our source project estimator (Section 3.2). Furthermore, TPTL shows significantly improved F1-score and competitive PofB20 over the state-of-the-art CPDP model LT [12], which suggests that TPTL has a substantially improved discriminability power for defect classification, and comparable cost to identify defects by inspecting top 20% lines of code.

Additionally, although the overestimated model Dycom shows a substantial advantage over other baselines on F1-score, TPTL still gains better performance with higher F1-score and significantly larger PofB20. Besides, TPTL only uses two selected source projects, but Dycom requires all candidate source projects plus 10% more labeled data from the target project. This condition implies the TPTL can potentially be further improved by using more selected source projects, and adjusting the model with more data from the target project.

Result 1: *The proposed TPTL model outperforms the previously succeeded CPDP models TCA+, TDS, LT, and Dycom in terms of F1-score and cost-effectiveness. TPTL can not only solve the instability problem of TCA+, but also largely save developers' efforts to find defective classes.*

5.2. RQ2: impact of SPE components on TPTL

Statistical metrics. Table 8 presents the F1-score and PofB20 of TPTL model that uses different statistical metrics (median, mean, variance, skewness, and kurtosis) of the source and target projects as the in-

Table 6
F1-score comparison of TPTL model versus 5 baselines (TCA+_Rnd, TCA+_All, TDS, LT, and Dycom) for 42 datasets. The results of TCA+_Rnd and Dycom are in the form of mean ± standard deviation because they run 10 times for each dataset, while other models run only once. "W/T/L" reports the number of datasets that a baseline is better, equal, worse than TPTL.

Dataset	TCA+_Rnd	TCA+_All	TDS	LT	Dycom	TPTL
ant-1.3	0.376 ± 0.090	0.359	0.394	0.239	0.411 ± 0.082	0.456
ant-1.4	0.327 ± 0.037	0.380	0.345	0.283	0.393 ± 0.048	0.377
ant-1.5	0.260 ± 0.096	0.322	0.262	0.252	0.439 ± 0.121	0.237
ant-1.6	0.541 ± 0.066	0.559	0.449	0.493	0.448 ± 0.028	0.595
ant-1.7	0.431 ± 0.114	0.505	0.454	0.434	0.429 ± 0.042	0.455
camel-1.0	0.092 ± 0.040	0.096	0.111	0.159	0.365 ± 0.114	0.093
camel-1.2	0.403 ± 0.011	0.453	0.378	0.379	0.412 ± 0.010	0.502
camel-1.4	0.334 ± 0.053	0.382	0.323	0.285	0.388 ± 0.062	0.339
camel-1.6	0.308 ± 0.047	0.328	0.190	0.273	0.383 ± 0.020	0.356
ckjm	0.620 ± 0.129	0.667	0.000	0.750	0.474 ± 0.084	0.714
ivy-1.1	0.533 ± 0.049	0.654	0.642	0.333	0.454 ± 0.046	0.738
ivy-1.4	0.131 ± 0.072	0.253	0.000	0.200	0.375 ± 0.026	0.160
ivy-2.0	0.324 ± 0.076	0.383	0.305	0.278	0.422 ± 0.024	0.349
jedit-3.2	0.442 ± 0.132	0.578	0.143	0.548	0.450 ± 0.027	0.536
jedit-4.0	0.458 ± 0.080	0.473	0.235	0.442	0.412 ± 0.033	0.447
jedit-4.1	0.423 ± 0.106	0.481	0.239	0.472	0.450 ± 0.209	0.522
jedit-4.2	0.358 ± 0.080	0.381	0.440	0.336	0.396 ± 0.067	0.370
jedit-4.3	0.065 ± 0.019	0.047	0.112	0.043	0.358 ± 0.147	0.050
log4j-1.0	0.516 ± 0.063	0.495	0.528	0.377	0.437 ± 0.036	0.637
log4j-1.1	0.524 ± 0.090	0.617	0.538	0.415	0.463 ± 0.033	0.699
log4j-1.2	0.562 ± 0.059	0.611	0.314	0.271	0.555 ± 0.030	0.606
lucene-2.0	0.569 ± 0.081	0.644	0.542	0.470	0.441 ± 0.037	0.677
lucene-2.2	0.515 ± 0.066	0.591	0.581	0.495	0.467 ± 0.066	0.621
lucene-2.4	0.458 ± 0.203	0.556	0.700	0.495	0.494 ± 0.053	0.633
poi-1.5	0.510 ± 0.135	0.689	0.600	0.745	0.411 ± 0.115	0.713
poi-2.0	0.264 ± 0.053	0.187	0.217	0.187	0.388 ± 0.041	0.218
poi-2.5	0.551 ± 0.141	0.701	0.554	0.754	0.507 ± 0.017	0.728
poi-3.0	0.513 ± 0.112	0.659	0.699	0.772	0.469 ± 0.127	0.787
redaktor	0.242 ± 0.063	0.318	0.304	0.252	0.364 ± 0.073	0.353
synapse-1.0	0.215 ± 0.097	0.289	0.526	0.444	0.399 ± 0.095	0.253
synapse-1.1	0.461 ± 0.079	0.460	0.487	0.488	0.414 ± 0.027	0.475
synapse-1.2	0.549 ± 0.095	0.596	0.510	0.444	0.447 ± 0.075	0.571
tomcat	0.240 ± 0.100	0.314	0.083	0.269	0.390 ± 0.074	0.287
velocity-1.4	0.458 ± 0.095	0.494	0.136	0.258	0.514 ± 0.039	0.734
velocity-1.6	0.461 ± 0.067	0.532	0.025	0.293	0.406 ± 0.100	0.568
xalan-2.4	0.322 ± 0.045	0.379	0.345	0.235	0.416 ± 0.029	0.403
xalan-2.5	0.466 ± 0.021	0.496	0.448	0.342	0.449 ± 0.055	0.533
xalan-2.6	0.486 ± 0.036	0.536	0.647	0.362	0.446 ± 0.038	0.512
xalan-2.7	0.493 ± 0.130	0.568	0.345	0.373	0.542 ± 0.059	0.616
xerces-1.2	0.209 ± 0.033	0.197	0.167	0.190	0.390 ± 0.021	0.192
xerces-1.3	0.331 ± 0.075	0.387	0.256	0.419	0.427 ± 0.019	0.377
xerces-1.4	0.576 ± 0.090	0.604	0.250	0.324	0.472 ± 0.021	0.690
Average	0.403 ± 0.079	0.458	0.353	0.378	0.433 ± 0.178	0.481
Improved	+19.28%	+04.98%	+36.12%	+27.13%	+11.08%	–
W/T/L	5/0/37	17/0/25	8/0/34	10/0/32	16/0/26	–
p-value	0.103	0.508	0.020	0.030	0.219	–
Cliff's δ	0.269	0.085	0.365	0.340	0.204	–

dependent variables of SPE. Numbers in the table are the average values across 42 datasets. We observe that TPTL with the median of software metrics obtains the best F1-score (0.481) and PofB20 (0.197) comparing with alternative statistical metrics (mean, variance, skewness, and kurtosis) and their 4 ways of combinations. Therefore, these results indicate that the median values of software metrics between the source and target projects are enough to capture their distribution similarity, i.e. a better way to represent the independent variables for SPE. This phenomenon implies that the source and target projects can be closely connected by their metric medians. This is also confirmed by Camargo and Ochimizu [3], and that is why their LT model that does standardization based on the median shows the best performance in a CPDP benchmark [12].

Regression models. Table 9 shows the average prediction results of TPTL with 8 different regression models (defined in Table 5), where SMOreg is the default approach for two regression models in the SPE. We can observe that replacing SMOreg with other alternatives has a negative effect on the prediction performance, both in terms of F1-score and PofB20. Therefore, SMOreg is the best algorithm for the two regression models in SPE.

Training data. Table 10 presents the F1-score and PofB20 of TPTL model with different number of training data (ranging from 100 to 1300 with step 200) in the SPE. Each training data represents a combination of hypothetical source and target project. In the table, we use 'All' to stand for the number of all possible combinations, instead of a digit, because each dataset has a different number of candidate source projects. Results show that SPE with more training data has positive effects on the prediction performance of TPTL. This case suggests that TPTL can be further improved by training SPE with more data.

Prediction Combination. Table 11 shows the prediction performance of TPTL with different settings on TCA+, building one (fTCA+ or pTCA+, as referred to in Section 3.4) or two TCA+ (using our combining method). We can notice that fTCA+ has a high F1-score (0.481) but low PofB20 (0.126), which suggests that the prediction accuracy is high, but the order of predicted confidence scores for classes cost much developers' efforts to find defective classes when inspecting code line by line.

Table 7
PofB20 comparison of TPTL model versus 5 baselines (TCA+_Rnd, TCA+_All, TDS, LT, and Dycom). The results of TCA+_Rnd and Dycom are in the form of mean ± standard deviation because they run 10 times for each dataset, while other models run only once. “W/T/L” reports the number of dataset that a baseline is better, equal, worse than TPTL.

Dataset	TCA+_Rnd	TCA+_All	TDS	LT	Dycom	TPTL
ant-1.3	0.170 ± 0.162	0.150	0.150	0.250	0.118 ± 0.074	0.450
ant-1.4	0.113 ± 0.184	0.075	0.075	0.075	0.094 ± 0.067	0.150
ant-1.5	0.194 ± 0.145	0.156	0.156	0.188	0.114 ± 0.072	0.219
ant-1.6	0.134 ± 0.158	0.163	0.141	0.174	0.104 ± 0.065	0.228
ant-1.7	0.176 ± 0.163	0.163	0.157	0.217	0.113 ± 0.071	0.169
camel-1.0	0.138 ± 0.208	0.077	0.154	0.154	0.122 ± 0.089	0.385
camel-1.2	0.072 ± 0.177	0.069	0.079	0.157	0.124 ± 0.118	0.171
camel-1.4	0.107 ± 0.182	0.055	0.090	0.159	0.106 ± 0.078	0.145
camel-1.6	0.080 ± 0.191	0.053	0.069	0.154	0.128 ± 0.113	0.170
ckjm	0.240 ± 0.165	0.200	0.200	0.400	0.146 ± 0.118	0.200
ivy-1.1	0.059 ± 0.174	0.032	0.032	0.111	0.088 ± 0.071	0.143
ivy-1.4	0.075 ± 0.198	0.188	0.188	0.188	0.128 ± 0.108	0.125
ivy-2.0	0.160 ± 0.163	0.150	0.125	0.150	0.120 ± 0.076	0.175
jedit-3.2	0.040 ± 0.161	0.000	0.044	0.189	0.091 ± 0.070	0.244
jedit-4.0	0.044 ± 0.173	0.013	0.027	0.173	0.100 ± 0.085	0.227
jedit-4.1	0.201 ± 0.156	0.013	0.025	0.190	0.101 ± 0.075	0.228
jedit-4.2	0.125 ± 0.190	0.125	0.104	0.167	0.110 ± 0.073	0.313
jedit-4.3	0.200 ± 0.217	0.364	0.182	0.182	0.118 ± 0.096	0.182
log4j-1.0	0.171 ± 0.160	0.147	0.176	0.324	0.115 ± 0.075	0.353
log4j-1.1	0.211 ± 0.149	0.135	0.216	0.243	0.131 ± 0.089	0.189
log4j-1.2	0.067 ± 0.240	0.058	0.079	0.143	0.148 ± 0.142	0.138
lucene-2.0	0.134 ± 0.155	0.066	0.066	0.132	0.110 ± 0.083	0.264
lucene-2.2	0.084 ± 0.160	0.049	0.028	0.160	0.118 ± 0.102	0.222
lucene-2.4	0.065 ± 0.165	0.044	0.094	0.138	0.116 ± 0.097	0.143
poi-1.5	0.104 ± 0.192	0.071	0.085	0.142	0.105 ± 0.078	0.106
poi-2.0	0.138 ± 0.184	0.081	0.108	0.108	0.137 ± 0.117	0.243
poi-2.5	0.043 ± 0.188	0.020	0.024	0.052	0.123 ± 0.120	0.093
poi-3.0	0.095 ± 0.185	0.025	0.028	0.075	0.114 ± 0.110	0.100
redaktor	0.115 ± 0.213	0.259	0.222	0.148	0.116 ± 0.085	0.037
synapse-1.0	0.194 ± 0.183	0.250	0.313	0.313	0.127 ± 0.088	0.250
synapse-1.1	0.170 ± 0.177	0.150	0.183	0.217	0.122 ± 0.086	0.267
synapse-1.2	0.169 ± 0.154	0.151	0.186	0.163	0.116 ± 0.078	0.198
tomcat	0.191 ± 0.188	0.169	0.130	0.234	0.114 ± 0.072	0.273
velocity-1.4	0.022 ± 0.204	0.000	0.136	0.068	0.193 ± 0.228	0.027
velocity-1.6	0.118 ± 0.174	0.026	0.051	0.231	0.137 ± 0.144	0.231
xalan-2.4	0.150 ± 0.155	0.145	0.200	0.264	0.119 ± 0.084	0.155
xalan-2.5	0.090 ± 0.165	0.088	0.121	0.212	0.106 ± 0.084	0.181
xalan-2.6	0.106 ± 0.160	0.127	0.080	0.224	0.128 ± 0.110	0.224
xalan-2.7	0.093 ± 0.235	0.073	0.137	0.198	0.154 ± 0.156	0.204
xerces-1.2	0.059 ± 0.179	0.056	0.070	0.056	0.123 ± 0.138	0.141
xerces-1.3	0.077 ± 0.155	0.058	0.087	0.232	0.102 ± 0.066	0.188
xerces-1.4	0.047 ± 0.183	0.025	0.021	0.082	0.094 ± 0.071	0.124
Average	0.120 ± 0.059	0.103	0.115	0.177	0.119 ± 0.096	0.197
Improved	+64.15%	+91.60%	+71.01%	+11.27%	+65.73%	–
W/T/L	5/0/37	3/2/37	6/2/34	13/3/26	6/0/36	–
p-value	0.000	0.000	0.000	0.260	0.000	–
Cliff’s δ	0.575	0.620	0.582	0.143	0.720	–

Table 8
Average performance (F1-score and PofB20) comparisons of TPTL with different statistical metrics (mean, variance, skewness, kurtosis, and median) of source and target projects as the independent variables of SPE.

Feature	F1-Score (Improved)	PofB20 (Improved)
median	**0.481**	**0.197**
mean	0.467 (−02.91%)	0.163 (−17.26%)
variance	0.458 (−04.78%)	0.177 (−10.15%)
skewness	0.475 (−01.25%)	0.161 (−18.27%)
kurtosis	0.471 (−02.08%)	0.167 (−15.23%)
median+mean	0.469 (−02.49%)	0.156 (−20.81%)
median+variance	0.467 (−02.91%)	0.170 (−13.71%)
median+skewness	0.472 (−01.87%)	0.188 (−04.57%)
median+kurtosis	0.464 (−03.53%)	0.191 (−03.05%)

Table 9
Average performance (F1-score and PofB20) comparisons of TPTL model with different regression models for their SPE.

Model	F1-Score (Improved)	PofB20 (Improved)
SMOreg	**0.481**	**0.197**
AR	0.444 (−07.70%)	0.196 (−00.61%)
DT	0.382 (−20.75%)	0.138 (−30.20%)
LSR	0.480 (−00.16%)	0.174 (−11.80%)
LR	0.480 (−00.16%)	0.197 (−00.17%)
RBF	0.383 (−20.23%)	0.161 (−18.46%)
SLR	0.365 (−24.04%)	0.132 (−33.06%)
ZR	0.365 (−24.04%)	0.132 (−33.06%)

In contrast, pTCA+ has low F1-score (0.413) but high PofB20 (0.222). This result implies that developers can find more defective classes by reading top 20% lines of code, but many of these classes are not actually defective (F1-score is low). Therefore, both fTCA+ and pTCA+ will waste developers’ efforts for code inspection in practical use. Moreover, we can find that, by combining fTCA+ and pTCA+, TPTL can maintain

Table 10
Performance comparison of TPTL model with different number of randomly selected training data for their SPE. To suppress the effect of the randomness, each experiment is repeated 10 times. Each experiment results is the average value across 42 datasets in terms of F1-score and PofB20 respectively. The ‘All’ in the last row indicates that TPTL uses all possible training data.

#Training	F1-Score (Improved)	PofB20 (Improved)
100	0.419 ± 0.009 (−12.97%)	0.161 ± 0.008 (−18.35%)
300	0.427 ± 0.018 (−11.30%)	0.162 ± 0.009 (−17.88%)
500	0.439 ± 0.014 (−08.76%)	0.166 ± 0.010 (−15.77%)
700	0.456 ± 0.012 (−05.20%)	0.185 ± 0.006 (−05.92%)
900	0.461 ± 0.009 (−04.11%)	0.184 ± 0.010 (−06.40%)
1100	0.472 ± 0.006 (−01.79%)	0.194 ± 0.005 (−01.41%)
1300	0.475 ± 0.003 (−01.19%)	0.188 ± 0.005 (−04.78%)
All	**0.481**	**0.197**

Table 11
Average performance (F1-score and PofB20) comparisons of TPTL model with different model settings.

Model Setting	F1-Score (Improved)	PofB20 (Improved)
fTCA+ and pTCA +	**0.481**	0.197
fTCA+	**0.481** (−00.00%)	0.126 (−36.04%)
pTCA+	0.413 (−14.14%)	**0.222** (+12.69%)

Table 12
Average time of model building and predicting.

No.	Model	Building	Predicting
1	TPTL	3300.606 s	0.001 s
2	TCA+_Rnd	0.116 s	0.001 s
3	TCA+_All	13547.193 s	0.001 s
4	TDS	0.034 s	0.001 s
5	LT	1.332 s	0.010 s
6	Dycom	0.073 s	0.002 s

a high F1-score (0.481) and substantially enhance the PofB20 of fTCA + from 0.126 to 0.197. Therefore, combining two TCA + by our method is reasonable and beneficial to TPTL.

> **Result 2:** *For SPE, the median is the best statistical metrics to capture the distribution similarity between source and target projects; SMOreg is the best approach for the two regression models in SPE; training SPE with more data can help TPTL achieve better performance; combining prediction results of two TCA + built in the second phase is reasonable and beneficial to TPTL.*

5.3. RQ3: time efficiency of TPTL

Table 12 lists the average time of model building and predicting, across 42 defect datasets. The table shows that TPTL needs about 55 min to build, and 0.001 s to predict the labels of classes in a target project. TCA + _Rnd works fast (building time = 0.116 s) because it randomly chooses a candidate source project for model building, instead of carefully selecting a better source project as TPTL. Moreover, building TCA + _All is very slow (about 3.8 h), this is because it inputs TCA + with all alternative source projects. Therefore, compared with TCA + _All, TPTL saves the model building time substantially. Besides, other models run fast for not using the TCA + .

Table 13
Average building time for TPTL components.

No.	Component	Execution Time
1	Collecting Training Data for SPE	3296.587 s
2	Training SPE	1.921 s
3	Executing TCA+	2.097 s
4	Building Classifier	0.001 s

To analyze why building TPTL needs 55 min, we provide the specific building time for each TPTL component in Table 13. We can find that building SPE, TCA + and classifier are fast (taking 1.921 s, 2.097 s and 0.001 s respectively), while 99.88% of the building time is used for collecting training data (independent and dependent variables) for SPE. As SPE does not need to be updated all the time in practices, TPTL thus has an acceptable building time. Even if SPE is required to be updated for better performance of TPTL, the two regression models in SPE can be efficiently updated with new data, instead of retraining the whole CPDP model from zero like TCA + _All.

> **Result 3:** *TPTL works efficiently with short predicting time. And TPTL has an acceptable building time, because most of the time is used to collect training data for SPE, which is unnecessary to be redone all the time in practices.*

6. Threats to validity

Internal validity relates to errors and the replication of baseline models. We re-implemented TCA + , TDS, LT, and Dycom by ourselves, and there could be errors that we did not notice. But we have double checked our code to minimize the possibility to make mistakes. And we evaluate our model only by using defeat datasets from the PROMISE repository, which may involve quality issues. Besides, the regression model SMOreg in TPTL is implemented by invoking the WEKA tool with default settings, and different parameters in SMOreg might cause different results. In practice, parameter tuning will require a lot of expertise, which will be hard for developers especially novices. Our proposed model shows that the default parameters in SMOreg achieve a good performance, which can make easy deployment of our tool in practice.

External validity relates to the generalizability of the experiment results. We have investigated 42 defect datasets from 14 distinct open source projects from the PROMISE repository. However, our experiment results may be different if we collect more datasets, and if we use datasets from closed software projects or different repositories. To minimize these threats, we plan to analyze our model on more and varied defect datasets in the near future.

Construct validity refers to the suitability of measures for our performance metrics. We adopt two commonly used evaluation metrics, F1-score and PofB20 (a cost-effectiveness measure) to evaluate our model. To investigate the statistical difference between TPTL and baseline models, we also run the Wilcoxon signed-rank test and corrected the results of multiple comparisons by the Bonferroni correction.

7. Related work

To effectively predict defective classes in software projects, researchers have developed a number of defect prediction models since decades ago [19,24,29,35]. Their models use software metrics, such as lines of code, to capture features of classes in projects. And they utilize software metrics of defective and clean classes to train a ML model, e.g. support vector machine (SVM) [17], under a WPDP setting where training and testing data are extracted from the same project [6,15,17,33,38]. However, WPDP models have applicability problem for a new project, since newly initiated projects rarely provide sufficient training data for WPDP models [39].

To address the limitation of WPDP models, a number of CPDP models have been proposed, which trains ML models using plenty of training data from other projects [18,23,28]. However, a CPDP model can hardly capture generalizable properties of defective classes in one or a set of source projects, so that its prediction on target project is usually unstable and unsatisfactory. To overcome the problem in CPDP models, researchers endeavor to build a bridge from source projects to target project. Generally, CPDP can be viewed as a specific case of transfer learning, which extracts knowledge from a set of source projects and transfers it to a target project.

Previously, researchers try to improve CPDP by selecting a suitable source project as the training data of a classifier. Watanabe et al. [36] verified the possibility of CPDP, and attributed such possibility to the similarity of the domain, programming language, metrics between the source and target projects. Later, Zimmermann et al. [39] conducted a large scale cross-project predictions among 12 real-world applications, and found that CPDP is challenging (only 3.4% predictions worked) because of the data distribution difference between the source and target projects. To solve this issue, Herbold et al. [11] proposed a training data selection (TDS) method to find better source projects by measuring the Euclidean distance between the data distributions of the source and target projects. The results indicated TDS can significantly improve the prediction performance (the success rate improved to 18%). Meanwhile, Nam et al. [23] successfully applied the Transfer Component Analysis (TCA) [27] method to CPDP. Instead of choosing source projects, TCA directly maps a given source project and a target project into a shared latent space, where their distribution distance is close. Nam et al. [23] further extended this approach to TCA+ with a data preprocessing method. Our proposed model TPTL is built upon TCA+ [23] and draws idea from TDS [11]. The main difference between our model and TDS is that we build two supervised regression models to estimate the data distribution similarity between source and target projects, instead of directly measuring their Euclidean distance based on an empirical assumption.

To understand the state-of-the-art in CPDP, Hosseini et al. [14] conducted a systematic literature review on 30 studies to synthesize the factors important to CPDP, with respect to models, datasets, and etc. They found that the CPDP model performance is influenced by the way it is built, and summarized frequently adopted configurations for each factor. Same as the results of this literature review, CPDP factor in our study are configured with frequently used ones: using the logistic regression model as the underlying classifier, inputting classifier with combined software metrics [10,16], verifying the model with Jureczko dataset [16], and evaluating the model with F1-score [15,17,23].

Apart from TCA+ and TDS, many other CPDP models have been proposed [12,14]. Herbold et al. [12] replicated 24 CPDP approaches, and found the best model is a log transformation (LT) based metric standardization model [3]. Generally, LT first preprocesses metric values of files in source and target projects by log transformation, and then aligns the median of transformed metrics between each source and target projects to mitigate the difficulty of cross-project prediction. We also compare this state-of-the-art model LT with our TPTL model.

Moreover, in the research of web effort estimation, a transfer learning model Dycom [22] with a better form of TCA was proposed and showed excellent experimental results. Dycom is a weighted sum of multiple transferred models respectively trained by different source projects and 10% data from the target project. In this study, we apply Dycom to the CPDP research and also compare it with the proposed model TPTL.

8. Conclusion and future work

In this article, we propose a two-phase transfer learning (TPTL) model for CPDP. TPTL first developed an SPE to automatically choose two source projects with highest distribution similarity to a target project from a set of candidate source projects. TPTL leverage TCA+ to build two prediction models based on the two selected projects, and combine their prediction results to further improve the prediction performance. We evaluate our TPTL model on 42 defect datasets and compare it with a related transfer learning model TCA+ with two versions (TCA+_Rnd and TCA+_All), a related source project selection model TDS, a state-of-the-art CPDP model LP, and a transfer learning model Dycom with better form of TCA showing excellent results in the web effort estimation. Experiment results show that:

- On average across 42 defect datasets, TPTL can respectively improve TCA+_Rnd, TCA+_All, TDS, LT, and Dycom by 19.28%, 4.98%, 36.12%, 27.13%, and 11.08% in terms of F1-score, respectively; by 64.15%, 91.60%, 71.01%, 11.27%, and 65.73% in terms of cost-effectiveness, separately. These results indicate that TPTL can solve the instability problem of TCA+ by choosing better candidate source projects, showing substantial improvements over the state-of-the-art and related models.
- We select two candidate source projects by two regression models in SPE. Results indicate that the median is the best statistical metrics to capture the distribution similarity between source and target projects; SMOreg is the best approach for the two regression models in SPE; TPTL can be further improved by training SPE with more data; and combining prediction results of two developed TCA+ is reasonable and beneficial to TPTL.
- When constructing a TPTL model, it respectively takes about 55 min and 0.001 s for model building and predicting on a dataset. As 99.88% of the building time is spent on collecting training data for SPE, which does not need to be redone all the time, therefore the working efficiency of TPTL is acceptable for practical use.

In the near future, we plan to evaluate the TPTL model with more software projects and develop a better model to further improve the performance of CPDP. We also plan to build a multi-source model to fully use knowledge recorded in all candidate source projects, instead of choosing only two source projects from candidates.

Acknowledgement

The work described in this paper was partially supported by the National Natural Science Foundation of China (Grant No. 61772093), Chongqing Research Program of Basic Science & Frontier Technology with Grant No. cstc2017jcyjB0305. The source code and datasets of TPTL can be downloaded from: https://bitbucket.org/ChaoLiuCQ/replication-kit-ist2018-tptl.

References

[1] H. Abdi, Bonferroni and šidák corrections for multiple comparisons, Encycl. Meas. Stat. 3 (2007) 103–107.
[2] E. Arisholm, L.C. Briand, M. Fuglerud, Data mining techniques for building fault-proneness models in telecom java software, in: Software Reliability, 2007. ISSRE'07. The 18th IEEE International Symposium on, IEEE, 2007, pp. 215–224.
[3] A.E. Camargo Cruz, K. Ochimizu, Towards logistic regression models for predicting fault-prone code across software projects, in: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society, 2009, pp. 460–463.
[4] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Inf. Sci. 179 (8) (2009) 1040–1058.
[5] N. Cliff, Ordinal methods for behavioral data analysis, Psychology Press, 2014.
[6] M. D'Ambros, M. Lanza, R. Robbes, An extensive comparison of bug prediction approaches, in: Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on, IEEE, 2010, pp. 31–41.
[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update, ACM SIGKDD Explorations Newsl. 11 (1) (2009) 10–18.
[8] A.E. Hassan, Predicting faults using the complexity of code changes, in: Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, 2009, pp. 78–88.
[9] 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.
[10] B. Henderson-Sellers, Object-oriented metrics: Measures of complexity, Prentice-Hall, Inc., 1995.
[11] S. Herbold, Training data selection for cross-project defect prediction, in: Proceedings of the 9th International Conference on Predictive Models in Software Engineering, ACM, 2013, p. 6.
[12] S. Herbold, A. Trautsch, J. Grabowski, A comparative study to benchmark cross-project defect prediction approaches, IEEE Trans. Softw. Eng. 44 (2017) 811–833.
[13] D.W. Hosmer Jr, S. Lemeshow, R.X. Sturdivant, Applied logistic regression, 398, John Wiley & Sons, 2013.
[14] S. Hosseini, B. Turhan, D. Gunarathna, A systematic literature review and meta-analysis on cross project defect prediction, IEEE Trans. Softw. Eng. (2017).

[15] T. Jiang, L. Tan, S. Kim, Personalized defect prediction, in: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, IEEE Press, 2013, pp. 279–289.

[16] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ACM, 2010, p. 9.

[17] S. Kim, E.J. Whitehead Jr, Y. Zhang, Classifying software changes: clean or buggy? IEEE Trans. Softw. Eng. 34 (2) (2008) 181–196.

[18] Y. Liu, T.M. Khoshgoftaar, N. Seliya, Evolutionary optimization of software quality modeling with multiple repositories, IEEE Trans. Softw. Eng. 36 (6) (2010) 852–864.

[19] Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Inf. Softw. Technol. 54 (3) (2012) 248–256.

[20] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (1) (2007) 2–13.

[21] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The promise repository of empirical software engineering data, 2012.

[22] L. Minku, F. Sarro, E. Mendes, F. Ferrucci, How to make best use of cross-company data for web effort estimation? in: Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on, IEEE, 2015, pp. 1–10.

[23] J. Nam, S.J. Pan, S. Kim, Transfer defect learning, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 382–391.

[24] N. Ohlsson, H. Alberg, Predicting fault-prone software modules in telephone switches, IEEE Trans. Softw. Eng. 22 (12) (1996) 886–894.

[25] T.J. Ostrand, E.J. Weyuker, R.M. Bell, Predicting the location and number of faults in large software systems, IEEE Trans. Softw. Eng. 31 (4) (2005) 340–355.

[26] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.

[27] S.J. Pan, I.W. Tsang, J.T. Kwok, Q. Yang, Domain adaptation via transfer component analysis, IEEE Trans. Neural Netw. 22 (2) (2011) 199–210.

[28] A. Panichella, R. Oliveto, A. De Lucia, Cross-project defect prediction models: L'union fait la force, in: Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on, IEEE, 2014, pp. 164–173.

[29] F. Peters, T. Menzies, A. Marcus, Better cross company defect prediction, in: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on, IEEE, 2013, pp. 409–418.

[30] F. Rahman, P. Devanbu, How, and why, process metrics are better, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 432–441.

[31] F. Rahman, D. Posnett, I. Herraiz, P. Devanbu, Sample size vs. bias in defect prediction, in: Proceedings of the 2013 9th joint meeting on foundations of software engineering, ACM, 2013, pp. 147–157.

[32] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy, Improvements to the smo algorithm for svm regression, IEEE Trans. Neural Netw. 11 (5) (2000) 1188–1193.

[33] S. Shivaji, J.E.J. Whitehead, R. Akella, S. Kim, Reducing features to improve bug prediction, in: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society, 2009, pp. 600–604.

[34] B. Turhan, On the dataset shift problem in software engineering prediction models, Empirical Softw. Eng. 17 (1–2) (2012) 62–74.

[35] B. Turhan, T. Menzies, A.B. Bener, J. Di Stefano, On the relative value of cross-company and within-company data for defect prediction, Empirical Softw. Eng. 14 (5) (2009) 540–578.

[36] S. Watanabe, H. Kaiya, K. Kaijiri, Adapting a fault prediction model to allow inter languagereuse, in: Proceedings of the 4th international workshop on Predictor models in software engineering, ACM, 2008, pp. 19–24.

[37] F. Wilcoxon, Individual comparisons by ranking methods, Biom. Bull. 1 (6) (1945) 80–83.

[38] T. Zimmermann, N. Nagappan, Predicting defects using network analysis on dependency graphs, in: Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on, IEEE, 2008, pp. 531–540.

[39] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, in: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM, 2009, pp. 91–100.