Improving and further analysis of eCPDP (EarlyCross-Project Defect Prediction)

Sunjae Kwon (cadet6465@kaist.ac.kr)
Korea Advanced Institute of Science and Technology
Duksan Ryu
Jeonbuk National University
Jongmoon Baik
Korea Advanced Institute of Science and Technology

Research Article

Keywords: CPDP, Transfer learning, SVD, Unit testing phase

Posted Date: April 8th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1520493/v1

Improving and further analysis of eCPDP (EarlyCross-Project Defect Prediction)

Sunjae Kwon[1] · **Duksan Ryu**[2] · **Jongmoon Baik**[1]

Abstract Context Cross-project Defect Prediction (CPDP) utilizes other finished project (i.e., source project) data to predict defects of the current working project. Transfer Learning (TL) has been mainly applied at CPDP to improve prediction performance by alleviating the data distribution discrepancy between different projects. However, existing TL-based CPDP techniques are not applicable at the unit testing phase since they require the entire historical target project data. As a result, they lose the chance to increase the product's reliability in the early phase by applying the prediction results.

Objective To increase the product's reliability in the early phase by proposing a novel TL-based CPDP technique applicable at the unit testing phase (i.e., eCPDP).

Method We utilize Singular Value Decomposition (SVD) that only requires source project data for TL.

Result eCPDP shows similar or better performance than the 5 state-of-the-art TL-based CPDP techniques on 9 different performance metrics over 24 projects.

Conclusion 1) We show that eCPDP is an applicable CPDP model at the unit testing phase. 2) It can help practitioners find and fix defects in an earlierphase than other TL-based CPDP techniques.

Keywords: CPDP · Transfer learning · SVD · Unit testing phase

Author's Contribution

All authors contributed equally to this work

Sunjae Kwon (Country: South Korea)
E-mail: cadet6465@kaist.ac.kr (Corresponding author)

Duksan Ryu (Country: South Korea)
E-mail: duksan.ryu@jbnu.ac.kr

Jongmoon Baik (Country: South Korea)
E-mail: jbaik@kaist.ac.kr

1. Korea Advanced Institute of Science and Technology, Daejeon, South Korea

2. Jeonbuk National University, Jeonju, South Korea

1 Introduction

A defect is derived from developer's errors in a software system, and it might cause failures that bring misbehavior and unexpected results to the system. Given that software systems have increasingly intertwined with our daily lives,defects are highly likely to damage our business and society. On the other hand, it is impossible to inspect entire modules to uncover defects in the software system due to the limited resources for software quality assurance (SQA). Thus, many researchers have proposed Software Defect Prediction (SDP) that predicts defect-prone modules and helps practitioners allocate or prioritize thelimited SQA resources to the defect-prone modules.

SDP usually utilizes machine learning algorithms that require training datafor building a prediction model. However, it is uncommon to acquire sufficient historical data of a newly started project to build an accurate predictionmodel (i.e., within-project environment). A solution for the challenge mentioned above is Cross-Project Defect Prediction (CPDP) that utilizes different finished project (i.e., source project) data to build a prediction model for the current working project (i.e., target project). Unfortunately, Zimmermannet al. (2009), Nam et al. (2013), and Xu et al. (2019) mention that naive CPDP yields poor prediction performance due to the data distribution discrepancy between source and target project. Researchers have applied Transfer Learning (TL) that transfers knowledge across different domain, and TL have shownto be available to alleviate the discrepancy problem (e.g., Turhan et al. 2009; Kawata et al.2015; Nam et al. 2013; Xu et al.2019)

However, there is an applicable time latency problem in the existing TL-based CPDP techniques. The existing techniques are only applicable when alltarget project data are collected because they require entire historical target project data to fit source and target project into similar distribution. In other words, they are inapplicable at the unit testing phase, where the target projectis under development, and all target project data have not been collected. Given that the sooner defects are discovered and fixed, the less effort and cost put into rework, existing TL-based CPDP techniques lose a chance of increasing the product's reliability in an earlier phase.

Therefore, we propose a novel approach called eCPDP (i.e., TL-basedCPDP technique using Singular Value Decomposition applicable at the unittesting phase) to address the latency problem of the existing TL-based CPDPtechniques. We utilize Singular Value Decomposition (SVD), one of the popular matrix factorization techniques. Unlike existing TL techniques, it providesa simple way to extract and transfer knowledge only using source project dataso that eCPDP offers practitioners prediction results at the unit testing phase.We compared the performance of eCPDP with 5 state-of-the-art CPDP techniques over 3 widely used SDP benchmark datasets including 24 differentprojects. Unlike baselines, only eCPDP is applicable at the unit testing phase;we compared a collection of eCPDP's prediction results executed during theunit testing phase with baseline's prediction results executed when all targetproject data have been collected. The evaluation results show that eCPDP has similar or better performance than baselines on 9 different performance metrics over 24 different projects. It indicates that applying eCPDP at the unit testing phase is viable.

This study enhances the previous work (Kwon et al. 2021). First, we enhance eCPDP by applying data distribution transformation and parameter optimization to improve prediction performance. Second, we extend baselinemodels with 5 state-of-the-art CPDP techniques. Also, we add 5 more conventional performance metrics and one cost-sensitive metric and append one moredataset to mitigate threats to validity. Finally, we further analyze eCPDP with ablation studies and Maximum Mean Discrepancy (MMD) tests on each process of it to confirm how each process of eCPDP affects prediction performanceand distribution discrepancy.

The main contributions of our study are summarized as the following 3 items. 1) we propose a novel TL-based CPDP technique applicable at the unittesting phase; 2) shows that eCPDP executed during the unit testing stage hassimilar to or outperform the baselines executed before the release; 3) further enhance the previous work in Kwon et al. (2021) in terms of methodology andanalysis method.

The rest of this paper is organized as follows. Section 2 introduces the related work and motivation of this study. Section 3 describes the frameworkand algorithm of the eCPDP. Section 4 shows the experimental settings for evaluating the performance of eCPDP. Section 5 shows the results and discusstheir implication. Section 6 shows the results of an ablation study on eCPDP to analyze the effect of each eCPDP's process on the prediction performance. The threats to validity is summarized in Section 7. We conclude this paper and provide future work in Section 8.

2 Background and Related Work

2.1 Transfer Learning (TL) for CPDP

SDP usually utilizes Machine learning (ML) classifiers to automate the processof defect prediction. The ML classifiers are designed to properly learn and classify, assuming that the training and testing data have the same distribution(Zimmermann et al. 2009; Nam et al. 2013). However, the assumption usually holds in a within-project environment but might not in a Cross-Projectenvironment, where programming language, coding style, and developer experience differ between source and target project. These differences cause thedistribution-discrepancy problem, and it is the major challenge of the CPDP.Transfer Learning (TL) has been applied to CPDP as an indispensable approach for alleviating the problem above. Figure 1 describes the data distribution discrepancy and the result of TL. Figure 1-(a) shows the distributiondiscrepancy between a source and a target project derived from their uniquecharacteristics. However, the TL makes source and target project data similar by filtering or transforming data as shown in Figure 1-(b).

Fig. 1 Distribution discrepancy problem and transfer learning

Among various TL techniques, the earliest and the most common technique is the instance-filtering technique. Using various filtering algorithms, it selectsinstances from source project data similar to the target project instances. Turhan et al. (2009) use Nearest Neighborhood (NN) filter to select close to the target project. Kawata et al. (2015) devise DSBSCANfilter as a variant of NN filter (i.e., Density-Based Spatial Clustering). In addition, Chen et al. (2015) and Limsettho et al. (2018) combine filtering algorithm and synthetic samplingtechniques to consider the class-imbalance problem additionally. Hosseini et al. (2018) combine NN filter with genetic algorithm to generate the best trainingdata.

There are some other techniques called as the data-transformation technique. The key idea is to align source and target data according to common latent factors to make the source and target data into a similar distribution. Nam et al. (2013) and Xu et al. (2019) adopt Transfer Component Analysis (TCA) and Balanced Distribution Adaptation (BDA) respectively to minimizethe distance between source and target project data. Gong et al. (2020) devise Stratification embedded in Nearest Neighbor (STr-NN) approach that is the combination of transformation and filtering method.

Lastly, Herbold (2013) proposed source project selection technique usingmeta-feature (e.g., average and standard deviation of each feature). It recommends a source project similar to a target project by comparing their meta- feature. Liu et al. (2019) and Sun et al. (2021) enhance Herbold (2013) combining TCA and collaborative filtering technique respectively.

While there are various studies on TL-based CPDP techniques, their applicable timing has not received any attention. Since the existing TL-based techniques require all module information of a developing target project, they can predict each module's defectiveness only when the entire target modules are implemented. Thus, this paper proposes novel TL-based CPDP techniques (i.e., eCPDP) that independently predict the module's defectiveness at theunit testing phase. The detail of the differences between eCPDP and existing TL-based CPDP will be covered in Section 2.2.

2.2 CPDP and unit testing phase

The unit testing phase is the earliest where practitioners identify defects in the source code of the proceeding project by allocating SQA resources. In addition, the NIST (National Institute of Standards and Technology) suggeststhat the relative cost of repairing bugs at the unit testing phase is half of the fixing cost at the integration testing phase in Planning (2002). Misra et al. (2017) also mentioned that unit testing could prevent additional defects that may have occurred due to the identified defect and reduce the cost of reworkand budget that might be put caused by the defects.

However, since the existing TL-based CPDP techniques are inapplicable atthe unit testing phase, this paper proposes a novel TL-based CPDP technique(i.e., eCPDP) applicable at the unit testing phase. Figure 2 clearly shows the differences between eCPDP and the existing TL-based CPDP techniques. Asshown in Figure 2-(a), existing TL-based CPDP techniques require the entiremodule information of a developing project for TL. In addition, they predict the proneness of the entire module at one time. On the other hand, as shown inFigure 2-(b), eCPDP only requires one module information of the developing project, so it can predict defectiveness of the module independently. In other words, eCDPD is applicable at the unit testing phase, where the coding process of each module has been finished.

In conclusion, eCPDP can provide practitioners with prediction results at the unit testing phase and help them effectively allocate limited SQA resources. For devising the novel TL techniques, we utilize Singular Value Decomposition (SVD), and the details on SVD will be described in the Section 2.4.

2.3 eCPDP and Just-In-Time (JIT) SDP

JIT defect prediction is a new type of SDP, and it is different from eCPDP in terms of prediction granularity. eCPDP predicts defective modules utilizingmodule-level product metrics. On the other hand, JIT SDP predicts defective commits using commit-level process metrics. In addition, since the defect-prone commits can be inspected while new commits are still being submitted, JIT SDP is also applicable at the unit testing phase.

Pascarella et al., 2019 mentioned that the commit-level prediction has two relative advantages over the module- level prediction. First, it can inform authors of defect-prone commits while the authors have fresh memory about the commits. Second, it is possible to minimize the impact of defect-prone commits on the following commits. Despite the strong points of the JIT SDP, it also has limitations. A commit, which is the granularity of the JIT SDP, depends on each developer's habit so that prediction results are likely correlated to the developers, not to the commit itself. In addition, when a defect-prone commit involves many modules, all the modules modified by the commit should be inspected.

However, eCPDP can mitigate the limitations of the JIT SDP. First, eCPDPuses module-level product metrics that only depend on the software module itself. Second, it reduces the amount of inspection resources compared to JIT SDP, since it allows engineers to concentrate on a single defect-prone module only. In conclusion, eCPDP and JIT SDP have pros and cons, and eCPDP can be an alternative that practitioners can choose depending on the development situation.

2.4 Singular Value Decomposition (SVD)

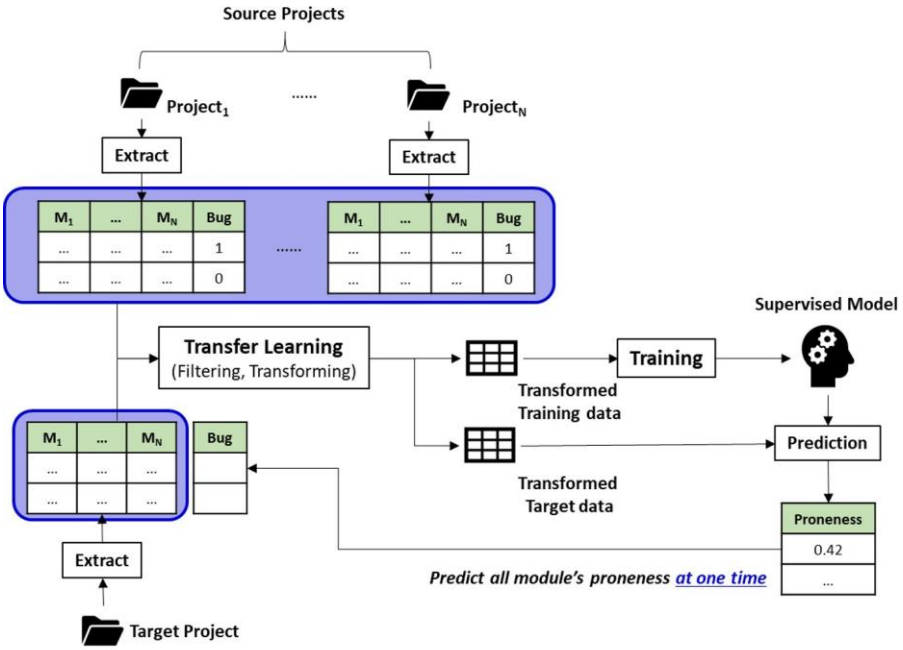
SVD is a well-known matrix factorization technique that provides a systematicway to decompose the 2D matrix form of high-dimensional data in terms of dominant patterns of the data. Brunton and Kutz (2019) mention that SVDis numerically stable and is guaranteed to exist for any matrix. Above all,it can discover dominant patterns purely from the data without additional knowledge or intuition. Due to the merits mentioned above, Ba et al. (2013), Reddy and Adilakshmi (2014), and Yuan et al. (2019) utilize SVD to extract eigenvectors in high dimensional data and use the extracted eigenvectors to make recommendations based on newly collected data.

Suppose we have a matrix $X \in \mathbb{C}^{n\times m}$ with n $\leq$ m, then SVD decomposes the matrix X into unique matrices as described in (1). The $U \in \mathbb{C}^{n\times n}$ and $V \in \mathbb{C}^{\mathrm{m}\times\mathrm{n}}$ are unitary matrices, and $\Sigma \in \mathbb{R}^{n\times m}$ is matrix with n non-negative values on diagonal and zeros off the diagonal. Since n $\leq$ m, Σ only have n none-zero values on the diagonal (i.e., $\Sigma = [\widehat{\Sigma}, 0]$, $\widehat{\Sigma} \in \mathbb{R}^{n\times n}$), so X can be economically represented as follows. This form is called as Economic- SVD, and this paper uses this form.

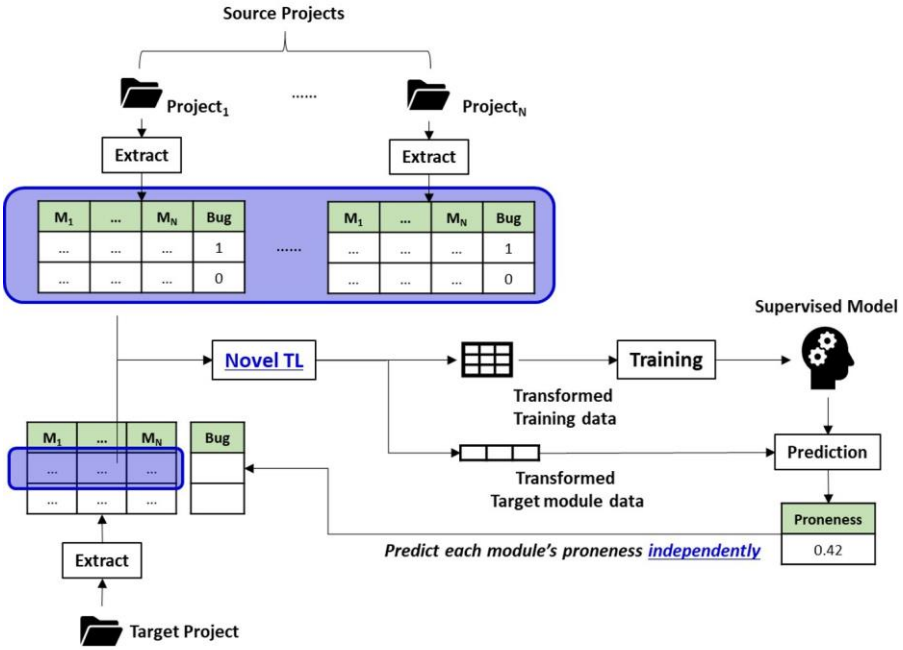
Economic-SVD

$$X = U\,\Sigma\,V^T = U\,[\widehat{\Sigma}, 0]\begin{bmatrix}\widehat{V}^T \\ \widehat{V}^{T\prime}\end{bmatrix} = U\,\widehat{\Sigma}\,\widehat{V}^T \tag{1}$$

The proposed technique utilizes SVD for principal component analysis (PCA), and Figure 3 describes how SVD extracts eigenvectors (i.e., the most dominant correlations between module information) from project data. The project data is matrix X including 300 module information with 20 features. SVD decom- poses the row-wise mean subtracted X (i.e., coordinate origin aligned) into $U, \widehat{\Sigma}$, and $\widehat{V}^T$. U represents the relationship between original features of the data and eigenvectors extracted from the project data, in other words, each column indicates how an eigenvector can be expressed with a combination of original features. $\widehat{\Sigma}$ is a diagonal matrix, and each element of the diagonal indicates importance of each eigenvector of U. Lastly, $\widehat{V}^T$ represents the relationship between module information and the extracted eigenvectors, in other words, each column indicates how a module information can be expressed with a com- bination of eigenvectors. In conclusion, since U is unitary matrix, it become the new coordination system of the project data and the rest $\widehat{\Sigma} * \widehat{V}^T$. become the coordinates of each module information corresponding to the U.



(a) Existing TL-based CPDP techniques



(b) eCPDP

Fig. 2 Differences between eCPDP and existing TL-based CPDP techniques

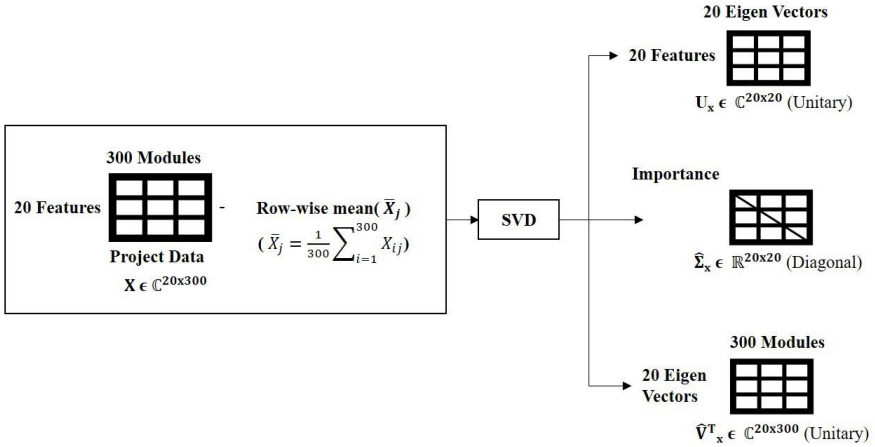


Fig. 3 Decomposition of project data using SVD

In conclusion, SVD can easily be used to decompose project data into a coordinate system (i.e., U) and corresponding coordinates (i.e., $\hat{\Sigma} * \hat{V}^T$) without additional information. The proposed technique utilizes SVD as TL by forcibly aligning a target module information into a coordinate system extracted from source project data. The detailed process is described in Section 3.1.

3 eCPDP

3.1 Transfer Learning using SVD

Transfer learning using SVD can be divided into 2 steps. 1) Extract a set of eigenvectors (i.e., coordination system) from Z-score normalized source project data, 2) Align target module data on the eigenvectors extracted from the source project data for putting them on the same coordination system. We apply Z-score normalization instead of row-wise mean subtraction because itcan align module information into the coordinate origin and, at the same time,reduce the differences of each feature's scale. The second step is called as Re- Alignment(R-Align) because target module data is re-aligned from the originalfeatures space to the eigenvector space extracted from the source project data.The detailed R-Align process is described in (2).

R-Align process

$$Y = U_X \hat{\Sigma}_X^Y \hat{V}_X^Y$$

$$U_X^T Y = U_X^T U_X \hat{\Sigma}_X^Y \hat{V}_X^Y \quad (2)$$

$$U_X^T Y = \hat{\Sigma}_X^Y \hat{V}_X^Y$$

Suppose there are a source project data $X \in \mathbb{C}^{n \times m}$ and a target project module information Y with the same number of features as X (i.e., $Y \in \mathbb{C}^{n \times 1}$). Y can be represented as $U_X \hat{\Sigma}_X^Y \hat{V}_X^Y$, where U_X is a set of eigenvectors extracted from X, and $\hat{\Sigma}_X^Y \hat{V}_X^Y$ is a new coordinate according to the U_X. Then, $\hat{\Sigma}_X^Y * \hat{V}_X^Y$ is easily calculated by multiplying the transpose of U_X (i.e., U_X^T), since U is unitary matrix (i.e., $U_X^T U_X = \mathrm{I}$). Now, $\hat{\Sigma}_X^Y \hat{V}_X^Y$ is new target module data forcibly aligned to U_X.

Table 1 describes part of Maximum Mean Discrepancy (MMD) calculation results for each step of the TL using SVD. MMD proposed by Gretton et al. (2012) is a kernel-based statistical test to calculate the difference between two data distributions, so the results of MMD indicate how much each step alleviates distribution discrepancy between source and target project data. The first row indicates MMD between the original source and target project data, the second row indicates MMD after Z-score normalization, and the last row indicates results after the R-Align process.

Table 1 Part of the MMD results of Ant v1.7 and Derby project

(1) Target project: Ant-1.7 Source Project	Poi-2.0	Camel-1.4	Ivy-2.0	Jedit-4.0	Log4j-1.0
Original data	580	28687	2781	49103	19543
After Z-score	1.78	2.66	2.37	1.65	4.24
After Z-score+R-Align	1.78	2.66	2.37	1.65	4.24
(2) Target project: Derby Source Project	ActiveMQ	Groovy	HBase	Hive	JRuby
Original data	3.06e+13	1.86e+13	2.80e+13	2.98e+13	3.06e+13
After Z-score	5.63e+64	1.10e+01	3.02e+01	5.62	1.63e+07
After Z-score+R-Align	5.63e+64	1.10e+01	3.02e+01	5.62	1.63e+07

As shown in the Table 1, Z-score normalization alleviates distribution discrepancy between source and target project data in most cases. As a result ofthe experiments, the distribution difference increased in 20 out of 194 possible pairs between source and target project (i.e., 10.3% out of possible pairs).However, since it is impossible to compare distribution differences at the unittesting phase, we devise an ensemble learning model that considers multiple source projects simultaneously to minimize bias of the special cases. In addition, Xia et al. (2016), Li et al. (2017), and Gong et al. (2020) also apply ensemble learning to overcome the weakness of a single classifier.

Note that, R-Align process does not affect the distribution discrepancy. The reason is that R-Align process is multiplying the unitary matrix to source and target project data. In geometric interpretation, multiplying unitary matrix is to rotate data based on the coordinate origin, so it does not change the distribution of them. However, it has positive effect on prediction performance; the detailed analysis will be presented in RQ4.

In conclusion, we can easily extract eigenvectors from source project data and simply align a new target project module according to the extracted eigen- vectors. However, in some cases, since these processes do not alleviate distribution discrepancy between source and target project data, we apply ensemble model to minimize the bias of the special cases. The detailed framework of the proposed technique is described in the section 3.3.

3.2 Box-cox transformation and Parameter optimization

Zhang et al. (2017) suggests that data distribution transformation, making software metrics follow a normal distribution (e.g., Log, Rank transformation), affects prediction performance in the CPDP environment. Box-cox transformation that subsumes log and other power transformation shows better performance than other transformation techniques. Li et al. (2020b) suggest that parameter optimization substantially improves the prediction performance in the TL-based CPDP environment with manageable computational cost.

Thus, we enhance eCPDP proposed in the previous study (Kwon et al. 2021) by applying Box-cox transformation and parameter optimization techniques to improve prediction performance. We utilize '*sklearn. prepossessing*'[1] and '*GridsearchCV*'[2] modules that are popular library of Python to implement Box-cox transformation and parameter optimization respectively.

3.3 Overall framework

The overall framework of eCPDP is described in Figure 5. It is composed of4 stages (i.e., Pre-possessing, Alignment, Training, and Prediction). eCPDP requires *n* independent source project data as a training dataset to predict the defectiveness of a completed module. In the Pre-possessing stage, each of *n* source project data and target module data are independently aligned to the same coordinate origin using Box-cox transformation and Z-score normalization. In the Alignment stage, each *n* source project data and target module data are independently aligned to the eigenvector space extracted from each source project using R-Align process. In the Training stage, *n* different pre- diction models are independently trained and parameter optimized using each aligned source project data. In the prediction stage, *n* different prediction models independently predict the target module's defectiveness, and the final prediction result can be obtained through soft-voting of *n* results.

[1] https://github.com/scikit-learn/scikit-learn/blob/7e1e6d09b/sklearn/preprocessing/
[2] https://github.com/scikit-learn/scikit-learn/blob/7e1e6d09b/sklearn/model selection/

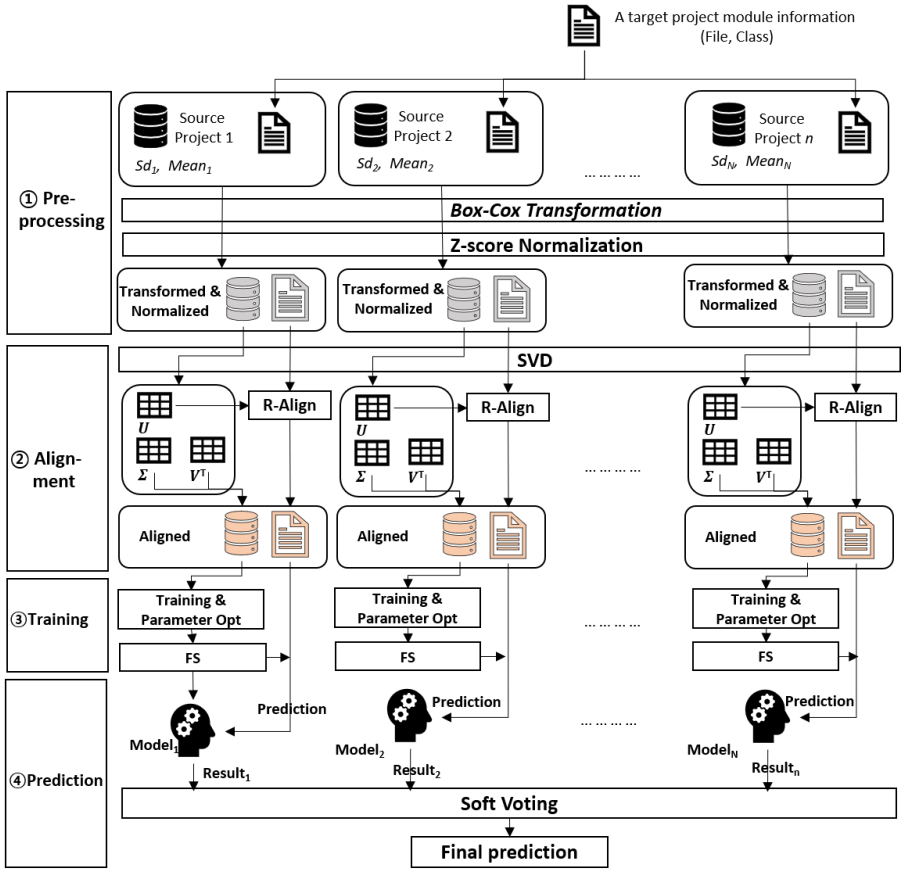


Fig. 4 Overall framework of eCPDP

Algorithm 1 describes the detailed step of eCPDP. First, each source project data is Box-cox transformed, and Z-score normalized. A module data of the target project is also transformed and normalized with each source project data's information (i.e., mean, sd, lamda) (Step1-2). Second, SVD extracts eigenvectors from transformed and normalized source project data. After that, target module data is aligned on the extracted eigenvectors from each source project data (Step 3-5). Third, n independent prediction models are trained and parameter-optimized using each aligned source project data (Step 6). After that, Feature-Selection (FS) are executed using '*SelectFromModel*'[3] module from '*sklearn.feature_selection*' library that assigns importance to each latent factor and removes unimportant eigenvector with the heuristically found threshold value (Step 6). Finally, the final independent n LR models are re trained with the best parameter settings and the feature selected source project data (Step 7). Each trained prediction model predicts the defectiveness of the target module using feature-selected target module data (Step 8). Lastly, the final defectiveness of the target module is obtainable from soft-voting of n independent prediction results (Step 9).

[3] https://github.com/scikit-learn/scikit-learn/blob/7e1e6d09b/sklearn/feature_ selection/

Algorithm 1 Algorithm of the eCPDP

Input:

Number of source project data: $[P_S^1, P_S^2, \dots P_S^n]$

Target module data: M_T

Output: Defectiveness of the target module: $Prob_D$

a) Prepossessing

1: Do Box-Cox transformation and Z-score normalization on each source project data
$[BP_S^1, BP_S^2, \dots BP_S^n]$ = BoxCox($[P_S^1, P_S^2, \dots P_S^n]$)
$[NP_S^1, NP_S^2, \dots NP_S^n]$ = Z-score($[BP_S^1, BP_S^2, \dots BP_S^n]$)

2: Do Box-Cox transformation and Z-score normalization on a target module data
$[BM_T^1, BM_T^2, \dots BM_T^n]$ = BoxCox_ fit($[(M_T, P_S^1), (M_T, P_S^2), \dots (M_T, P_S^n)]$)
$[NM_T^1, NM_T^2, \dots NM_T^n]$ = Z-score_ fit($[(BM_T^1, BP_S^1), (BM_T^2, BP_S^2), \dots (BM_T^n, BP_S^n)]$)

b) Alignment

3: Do SVD on each source project data
$[U_S^1, \Sigma_S^1, \dots V_S^1]$, $[U_S^2, \Sigma_S^2, \dots V_S^2]$, ... $[U_S^n, \Sigma_S^n, \dots V_S^n]$ = SVD($[NP_S^1, NP_S^2, \dots NP_S^n]$)

4: Align each source project data based on extracted latent factors (U_S^T) from self.
$[AP_S^1, AP_S^2, \dots AP_S^n]$ = $[(\Sigma_S^1 * V_S^1), (\Sigma_S^2 * V_S^2), \dots (\Sigma_S^n * V_S^n)]$

5: Align each target project data based on the associated latent factors (R-Align)
$[AM_T^1, AM_T^2, \dots AM_T^n]$ = $[(U_S^{1^T} * NM_T^1), (U_S^{2^T} * NM_T^2), \dots (U_S^{n^T} * NM_T^n)]$

c) Training

6: Training and optimizing parameters logistic regression (LR) models
[*Best_param*^{1}, *Best_param*^{2}, ... *Best_param*^{n}], [*T_ LR*^{1}, *T_ LR*^{2}, ... *T_ LR*^{n}] =
Training(Gridsearch([*LR*^{1}, AP_S^1],[*LR*^{2}, AP_S^2], ... [*LR*^{n}, AP_S^n]))

7: Feature selecting using *SelectFromModel* module
[*FS_* AP_S^1, *FS_* AP_S^2, ... *FS_* AP_S^n], [*FS_*AM_T^1, *FS_* AM_T^2, ... *FS_* AM_T^n] =
SelectFromModel([*T_ LR*^{1}, AP_S^1],[*T_ LR*^{2}, AP_S^2], ... [*T_ LR*^{n}, AP_S^n]

8: Retraining LR model with best parameter setting and feature selected data
[*F_LR*^{1}, *F_LR*^{2}, ... *F_LR*^{n}] = Training([*LR*^{1}, *Best_param*^{1}, *FS_*AP_S^1],
[*LR*^{2}, *Best_param*^{2}, *FS_*AP_S^1], … [*LR*^{n}, *Best_param*^{n}, *FS_*AP_S^n])

d)Prediction

9: Predict defectiveness of the target module with LR models
$[prob_D^n, prob_D^n, \dots prob_D^n]$ = Predict[(*F_LR*^{1}, *FS_*AM_T^1), (*F_LR*^{2}, *FS_* AM_T^2),
… (*F_LR*^{n}, *FS_*AM_T^n)]

10: Get final defectiveness through soft-voting of the results
$Prob_D$ = *mean*$[prob_D^n, prob_D^n, \dots prob_D^n]$

4 Experimental setup

4.1 Research questions

This study establishes 3 research questions to confirm performance of eCPDP. First, to evaluate performance of eCPDP, we compared performance of eCPDP with state-of-the-art CPDP techniques on various metrics (RQ1, 2). Second, in order to find the best classifier with eCPDP, we conduct combination experiments with popular classifiers (RQ3). The research questions and motivations of them are described below.

1. **RQ1:** Does eCPDP outperform on prediction performance than baselines?
   - **Motivation:** Unlike existing TL-based techniques, eCPDP only requires source project data for executing TL. If the performance of eCPDP is inferior to the existing techniques due to the absence of tar-get project data, it can be said that eCPDP is inapplicable at the unit testing phase. Thus, in this question, we aim to confirm the applicability of eCPDP at the unit testing phase by comparing state-of-the-art TL-based CPDP techniques. If the performance of eCPDP is similar toor outperforms state-of-the-art techniques, it can be seen that eCPDP is applicable.

2. **RQ2:** Can eCPDP reduce more effort and cost than baselines?
   - **Motivation:** Mende and Koschke (2010), Bennin et al. (2016), Zhang and Cheung (2013), and Panichella et al. (2016) mention that when analyzing the performance of SDP model, effort and cost should be considered to evaluate the applicability of the model in the practice. Thus, we aim to confirm the applicability of eCPDP at the unit testing phase in terms of effort and cost through this research question.

3. **RQ3:** Which classifier works the best with eCPDP?
   - **Motivation:** Tantithamthavorn et al. (2018), Li et al. (2020a), and Kang et al. (2021) mention that prediction accuracy on various prediction models largely depend on the parameter setting. It also suggests that optimal selection of the combination of classifier and transfer learning is as crucial as parameter optimization. Thus, studying the combi- nation of classifier and eCPDP with parameter optimization is necessary. Since there is no adjustable parameter in the process of transfer learning using SVD, we focus on the parameters of various classifiers.

4.2 Experimental Dataset

In our empirical study, we choose three publicly available datasets such as JERIKO (Jureczko et al. 2010), AEEEM (D'Ambros et al. 2010), JIRA (Yatish et al. 2019), and Table 2 show the overview of the three datasets. They have been frequently used for CPDP literature, and their diversity can help our approach avoid bias from a dataset. In addition, each metric of them is code and process metrics that are module independently calculated using automaticanalytic tools (i.e., SciTools and Ckjm). So, we assume that all metric valuesare recognizable at the unit testing phase. Although eCPDP and baselinesuse the same dataset, they use different amount of module information of thetarget project. As shown in Figure 2, baselines utilize all module information of a target project for prediction; on the other hand, eCPDP only uses theinformation of one module to be predicted.

1. JERIKO: It was prepared by Jureczko and Madeyski (2010). It includes various versions of open source java projects with object-oriented metrics.Among many projects in the PROMISE dataset, we choose 10 projects with 20 product metrics in common, which have been mainly selected forother CPDP studies.

2. AEEEM: It was extracted by D'Ambros et al. (2010) to compare the effect of different sets of metrics on the performance. It includes 5 open source projects with 61 metrics. The metrics consist of static, process metrics andentropy-based metrics.

3. JIRA: It was collected by Yatish et al. (2019). It is a highly-curated defect dataset including 7 open source java projects. It contains 65 softwaremetrics (54 code metrics, 5 process metrics, and 6 ownership metrics).

Table 2 An overview of studied projects

Dataset	Project	Entities	Buggy(%)	Metrics
JERIKO	Ant-1.7	745	166 (22.28%)	21
	Poi-2.0	872	37 (4.24%)	21
	Camel-1.4	872	145 (16.62%)	21
	Ivy-2.0	352	40 (11.36%)	21
	Jedit-4.0	306	75 (24.50%)	21
	Log4j-1.0	135	34 (25.18%)	21
	Xalan-2.4	723	110 (15.21%)	21
	Velocity-1.6	229	78 (34.06%)	21
	Tomcat	858	77 (8.97%)	21
	Xerces-1.3	453	69 (15.23%)	21
	luccen-2.4	536	203 (37.87%)	21
	synapse-1.2	745	166 (22.28%)	21
AEEEM	Eclipse JDT Core	997	206 (21.96%)	61
	Equinox	324	129 (39.81%)	61
	Apache Lucene	691	64 (9.26%)	61
	Mylyn	1862	245 (13.15%)	61
	Eclipse PDE UI	1292	209 (16.17%)	61
JIRA	ActiveMQ	1884	293 (15.55%)	65
	Derby	2705	383 (14.15%)	65
	Groovy	821	70 (8.52%)	65
	HBase	1059	218 (20.58%)	65
	Hive	1416	283 (19.99%)	65
	JRuby	731	87 (11.9%)	65
	Wicket	1763	130 (7.37%)	65

4.3 Baseline models

This study compares performance of eCDPD with 5 state-of-the-art techniques including FesCH proposed by Ni et al. (2017), GIS proposed by Hosseini et al. (2018), CDE proposed by Limsettho et al. (2018), TPTL proposed by Liu et al. (2019), and CFPS proposed by Sun et al. (2021). Table 3 summarizes baseline models and their brief description. We utilize publicly opened version of TPTL, implement other approaches according to the pseudo-code written in the paper. We apply the best parameter setting of each baseline described in the papers.

Table 3 Short description of baseline models

Baseline	Description
FesCH	Feature selection using density-based clustering algorithm
GIS	Combine NN filter and Genetic algorithm to generate training data
CDE	Synthetic Minority Oversampling Technique (SMORE) with class distribution estimation
TPTL	Combine source project selection technique and TCA+(Nam et al. 2013)
CFPS	Source project selection using collaborative filtering technique

4.4 Evaluation metrics

In this study, we adopt 2 types of performance metrics. First, we compute 7 commonly used metrics (i.e., AUC, PD, PF, F-measure, G-mean, Balance, MCC) to confirm conventional prediction performance of the model. Second, for confirming cost and effort effectiveness of each model, we adopt FIR (File Inspection Reduction), and CIR (Cost Inspection Reduction). Except for MCC, eight other metrics range in [0,1]. In addition, PF and CIR need to be minimized (i.e., less is better), and seven other metrics (i.e., AUC, PD, F-measure, G-mean, Balance, MCC, FIR) need to be maximized (i.e., more is better). All 9 metrics can be calculated based on Table 4, and the definitions and equations for each metric are as follows.

Table 4 Confusion matrix

	Predict as Defective	Predict as Clean
Actual Defective	TP	FN
Actual Clean	FP	TN

1. AUC (Area Under the Receiver Operating Characteristic curve): It is areaunder the curve between the true positive rate and false-positive rate, andit is independent of the cut-off value.

2. PD(Sensitivity): It is the ratio of defective instances that are predicted as defective instances. It focuses on where actual defective instances are predicted as non-defective instances.

$$PD = \frac{TP}{TP + FN} \tag{3}$$

3. PF(Specificity): It is the ratio of non-defective instances that are predicted as defective instance. It focuses on where actual non-defective instances are predicted as non-defective instances.

$$PF = \frac{FP}{FP + TN} \tag{4}$$

4. F-measure: It is harmonic mean of precision ($prec = \frac{TP}{TP + FP}$) and PD. Precision can effectively measure the performance of models when the cost of false positive is high, and F-measure seeks a balance of precision and PD.

$$F - measure = \frac{2 * PD * prec}{PD + pre} \tag{5}$$

5. G-mean: It is the geometric mean of recall values from the non-defective instance and the defective instance. It is useful for assessing the performance of the model under an imbalanced dataset.

$$G - mean = \sqrt{PD * (1 - PF)} \tag{6}$$

6. Balance: It is a Euclidean distance from ideal point (i.e., (PD=1, PF=0)) to model's real point (i.e., (PD, PF)). Higher balance indicates a model have good balance of the performance between the defective and clean modules, so it is also useful for assessing the performance of the model under an imbalanced dataset.

$$Balance = 1 - \sqrt{\frac{(1-PD)^2 + (0-PF)^2}{2}} \tag{7}$$

7. MCC (Matthews correlation coefficient): It includes all parts of the con- fusion matrix, different from other performance metrics. MCC fall in the range [-1,1], and values +1, 0, and -1 represent the perfect prediction, a random prediction, and the perfect inverse prediction, respectively.

$$MCC = \sqrt{\frac{(TP * TN) - (FP * FN)}{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{8}$$

8. FIR (File Inspection Reduction): It was proposed in Shin et al. (2010). It is the ratio of the reduced number of files to inspect using a prediction model compared to a random selection to obtain the same PD. When FI (File Inspection) is $FI = \frac{TP + FP}{TP + TN + FP + FN}$, FIR is defined as follows.

$$FIR = \frac{PD - FI}{PD} \tag{9}$$

9. CIR (Cost Inspection Reduction): It is the modified version of the met ric proposed in Zhang and Cheung (2013) to calculate the cost of defect prediction. CIR is ratio of the cost to inspect using a prediction model com- pared to cost to inspect all modules at integration testing phase. Assuming average cost of inspection for the defective module is C_i and average cost of missing a defective module C_{fn}, CIR is defined as follows.

$$CIR = \frac{C_i * (TP+FP) + C_{fn} * FN}{C_i * (TP+FP+ FN + TN)} \tag{10}$$

In addition, as mentioned in section 2.2, we set twice the C_i, C_{fn} in integration testing phase compared to them of unit testing phase, also assume that C_i / C_{fn} = 1/3 on both phases. (i.e., unit testing phase of C_i, C_{fn} : α, 3α, integration testing phase of C_i, C_{fn} : 2α, 6α).

4.5 Validation setting and Statistical test

Given a raw dataset with *N* project data, we select a project as target project, and remaining *N-1* project data are used as source project data. We conduct each baseline and eCPDP 30 time to reduce the randomness of their performance.

After measurement, we conduct the Wilcoxon rank sum test proposed by Wilcoxon (1946) with the 95% confidence level (i.e., p-value < 0.05) and Cohen's d effect size test proposed by Cohen (1988) that have been frequently applied in SDP studies. Wilcoxon rank sum test is a non-parametric statistical test to check if two independent distributions are equal. In addition, it does not have an assumption about the underlying distribution of data, Arcuri and Briand (2011) recommend using it in the software engineering study. If there is a significant difference between two independent variables, we apply Cohen's d effect size test, a non-parametric effect size measure, to quantify the magnitude of difference between them. The difference includes 4 classes as follows: negligible ($|d| < 0.2$), small ($0.2 \leq |d| < 0.5$), medium ($0.5 \leq |d| < 0.8$), and large ($0.8 \leq |d|$).

5 Experimental Result

5.1 RQ1: Does eCPDP outperform on prediction performance than baselines?

Approach To evaluate the performance of eCPDP, we choose 5 state-of-the- art TL-based CPDP techniques and compare eCPDP with them on 7 conventional prediction performance metrics (i.e., AUC, PD, PF, G-mean, F-measure, Balance, MCC). We execute each technique 30 times per target project and compare the performance distribution using Wilcoxon rank sum test and Co- hen's d effect size test.

Findings Table 5 shows Cohen's d effect size between eCPDP and baselines on 7 different metrics. The characters between the parenthesis represent the effect size. 'S' represents small, 'M' represents medium, and 'L' represents large. The 'n.s' indicates that the pair of performance between baseline and eCPDP is not statistically significant according to the Wilcoxon rank sum test. In the case of AUC, G-mean, F-measure, Balance, and MCC, eCPDP outperforms the baselines with more than medium effect size in 21 of 25 cases (i.e., 5 metrics * 5 baselines). In addition, when there is no significant difference in the PF between eCPDP and baselines, PD outperforms baseline and vice versa. It indicates that eCPDP has a better balance between PF and PD than baselines, in other words, it can predict more actual defects while minimizing false alarm rate. In conclusion, eCPDP has similar or better performance than baselines on 7 performance metrics.

Table 5 Cohen's d and effect size between the performance measure of eCPDP and baselines on 7 metrics ('*' indicate p-value < 0.05, '**' for p-value < 0.01, '***' for p-value < 0.001)

	FesCH	GIS	CDE	TPTL	CFPS
AUC	0.65(M)***	2.04(L)***	0.74(M)***	0.69(M)***	0.54(M)***
G-mean	0.53(L)***	0.38(S)***	1.51(L)***	1.25(L)***	2.18(L)***
F-measure	0.37(S)***	n.s.	0.93(L)***	0.46(M)*	1.63(L)***
Balance	0.44(M)***	0.32(S)***	1.60(L)***	1.12(L)***	2.55(L)***
MCC	0.80(L) ***	0.48(M) **	0.52(L) ***	0.48(M)**	1.06(L)***
PD	n.s.	n.s.	1.76(L)***	1.16(L)***	2.16(L)***
PF	1.07(L)***	0.40(S)***	n.s.	n.s.	n.s.

Table 6 presents the average performance of 7 metrics over each dataset. We consider a model wins if it achieves the best performance among others. Baselines win from 0 to 5 cases, while eCPDP wins 10 out of 21 cases. Furthermore, Table 7 presents average performance over each target project on AUC and G-mean, which are cut-off independent and cut-off dependent metric, respectively. Similar to Table 6, baselines win from 0 to 5 cases, while eCPDP wins 13 and 10 out of 24 cases in terms of AUC and G-mean, respectively. This indicates that eCPDP has better prediction performance than baselines over various datasets and projects.

Table 6 Average performance measure of eCPDP and 5 baseline models on 7 metrics per each dataset

Dataset	Metrics	FesCH	GIS	CDE	TPTL	CFPS	eCPDP
JERIKO	AUC	0.734	0.648	0.729	0.737	0.734	**0.773**
	PD	**0.655**	0.699	0.306	0.359	0.276	0.615
	PF	0.295	0.402	**0.092**	0.128	0.109	0.239
	G-mean	0.672	0.626	0.510	0.540	0.427	**0.674**
	F-measure	0.445	0.431	0.330	0.425	0.268	**0.461**
	Balance	0.666	0.621	0.502	0.531	0.457	**0.663**
	MCC	0.292	0.245	0.246	0.271	0.205	**0.325**
AEEEM	AUC	0.706	0.656	0.718	0.716	0.752	**0.753**
	PD	**0.678**	0.548	0.412	0.378	0.104	0.568
	PF	0.341	0.236	0.128	0.132	**0.006**	0.261
	G-mean	**0.664**	0.639	0.582	0.549	0.292	0.632
	F-measure	0.407	**0.410**	0.387	0.390	0.172	0.393
	Balance	**0.661**	0.629	0.565	0.539	0.367	0.618
	MCC	0.259	0.265	**0.298**	0.268	0.223	0.260
JIRA	AUC	0.777	0.689	0.774	0.762	0.782	**0.825**
	PD	**0.893**	0.539	0.275	0.508	0.149	0.649
	PF	0.711	0.160	**0.044**	0.097	0.021	0.216
	G-mean	0.452	0.665	0.490	0.653	0.326	**0.687**
	F-measure	0.289	**0.419**	0.320	0.323	0.208	0.413
	Balance	0.483	0.647	0.486	0.640	0.397	**0.668**
	MCC	0.147	0.324	0.295	0.288	0.214	**0.348**
	#of wins	5	2	3	0	1	**10**

Answer to RQ1: eCPDP has similar or better performance than 5 state of the art TL-based CPDP techniques on 7 performance metrics. In addition, eCPDP shows better performance over various datasets and projects. Therefore, from the perspective of conventional prediction performance metrics, applying eCPDP at the unit testing phase is viable.

5.2 RQ2: Can eCPDP reduce more effort and cost than baselines?

Approach It is the same as RQ1, except for applied performance metrics. We utilize FIR (i.e., effort-aware performance metric) and CIR (i.e., cost-aware performance metric) for evaluating effort and cost of the model.

Findings Table 8 shows Cohen's d effect size between eCPDP and baselines on FIR and CIR. In the case of FIR, eCPDP is similar to (i.e., CDE, TPTL, CFPS) or more than medium effect size better than baselines (i.e., FesCH, GIS).

Table 7 Average performance measure of eCPDP and 5 baseline models on AUC andG-mean per each target project

Dataset	Project	FesCH	GIS	CDE	TPTL	CFPS	eCPDP
(a) AUC							
JERIKO	Ant-1.7	0.807	0.704	0.759	0.770	0.774	**0.825**
	Poi-2.0	**0.711**	0.610	0.669	0.669	0.691	0.674
	Camel-1.4	0.617	0.608	0.653	**0.803**	0.663	0.690
	Ivy-2.0	0.774	0.620	0.756	0.772	0.797	**0.823**
	Jedit-4.0	0.760	0.642	0.754	**0.824**	0.751	0.794
	Log4j-1.0	0.779	0.706	0.806	0.639	0.771	**0.876**
	Xalan-2.4	0.748	0.663	0.719	**0.801**	0.757	0.789
	Velocity-1.6	**0.720**	0.550	0.678	0.635	0.640	0.707
	Tomcat	0.815	0.691	0.773	0.802	0.795	**0.831**
	Xerces-1.3	0.724	0.681	0.713	0.699	0.689	**0.749**
	luccen-2.4	0.676	0.696	0.728	0.685	0.733	**0.749**
	synapse-1.2	0.759	0.609	0.738	0.739	0.743	**0.768**
AEEEM	Eclipse JDT	0.623	0.673	0.693	0.720	**0.803**	0.802
	Equinox	0.757	0.704	0.802	0.695	**0.810**	0.802
	Lucene	0.665	0.647	0.751	0.747	0.759	**0.809**
	Mylyn	**0.746**	0.613	0.617	0.695	0.655	0.621
	Eclipse PDE	0.737	0.643	0.724	0.725	**0.738**	0.726
JIRA	ActiveMQ	0.762	0.706	0.783	0.691	0.771	**0.811**
	Derby	0.788	0.673	0.739	0.766	0.754	**0.789**
	Groovy	0.677	0.632	0.763	0.700	0.801	**0.826**
	HBase	0.760	0.682	0.776	0.813	0.765	**0.828**
	Hive	**0.791**	0.704	0.786	0.713	0.753	0.786
	JRuby	0.804	0.649	0.703	**0.863**	0.757	0.823
	Wicket	0.860	0.778	0.864	0.787	0.873	**0.900**
Average		0.744	0.662	0.740	0.740	0.752	**0.786**
# of wins		3	0	0	4	3	**13**
(b) G-mean							
JERIKO	Ant-1.7	0.740	0.701	0.591	0.579	0.660	**0.743**
	Poi-2.0	**0.698**	0.582	0.503	0.525	0.636	0.608
	Camel-1.4	0.575	0.599	0.373	0.459	0.322	**0.619**
	Ivy-2.0	0.735	0.610	0.625	0.600	0.416	**0.748**
	Jedit-4.0	0.711	0.609	0.636	**0.747**	0.178	0.703
	Log4j-1.0	0.674	0.695	0.430	0.371	0.569	**0.721**
	Xalan-2.4	0.701	0.658	0.634	0.398	0.311	**0.734**
	Velocity-1.6	**0.606**	0.451	0.383	0.547	0.143	0.605
	Tomcat	**0.735**	0.683	0.613	0.508	0.402	0.734
	Xerces-1.3	0.626	**0.676**	0.512	0.423	0.524	0.586
	luccen-2.4	0.582	**0.694**	0.338	0.664	0.347	0.606
	synapse-1.2	0.626	**0.676**	0.512	0.423	0.524	0.586
AEEEM	Eclipse JDT	0.626	0.665	0.442	**0.722**	0.295	0.566
	Equinox	0.720	0.696	**0.738**	0.618	0.486	0.682
	Lucene	0.640	0.638	0.565	0.441	0.181	**0.668**
	Mylyn	**0.685**	0.574	0.579	0.468	0.371	0.595
	Eclipse PDE	0.650	0.621	0.588	0.498	0.125	**0.651**
JIRA	ActiveMQ	0.583	0.684	0.350	0.512	0.000	**0.746**
	Derby	0.055	**0.640**	0.343	0.477	0.094	0.537
	Groovy	0.577	0.600	0.532	**0.876**	0.396	0.697
	HBase	0.467	0.674	0.593	0.687	0.221	**0.773**
	Hive	**0.680**	0.675	0.362	0.673	0.342	0.564
	JRuby	0.294	0.606	0.534	**0.887**	0.503	0.751
	Wicket	0.508	0.776	0.717	0.457	0.392	**0.779**
Average		0.606	0.640	0.519	0.575	0.369	**0.669**
# of wins		5	4	1	4	0	**10**

Table 8 The Cohen's d and effect size between the performance measure of eCPDP and baselines on FIR and CIR. (Meaning of '*' is same as Table 5)

	FesCH	GIS	CDE	TPTL	CFPS
FIR	1.16(L)***	0.44(M)***	n.s.	n.s.	n.s.
CIR	2.78(L)***	2.01(L)***	1.43(L)***	4.85(L)***	1.43(L)***

However, as shown in Table 5, PD of eCPDP is similar to FesCH and GIS and significantly better than CDE, TPTL, and CFPS. Considering the PD performance results, eCPDP can reduce inspection effort to reach similar PD of FesCH and GIS. In addition, eCPDP can reach higher PD than CDE, TPTL, and CFPS with similar inspection effort. In the case of CIR, eCPDP is a large effect size better than every baseline. As shown in Section 5.1, eCPDP has better prediction performance than baselines, so eCPDP can recognize more defects than baselines with smaller cost. In conclusion, eCPDP is better than baselines on cost-effectiveness, and it is similar to or better than baselines on the effort-effectiveness.

Table 9 Average performance measure of eCPDP and 5 baseline models on FIR and CIR per each dataset

Dataset	Metric	FesCH	GIS	CDE	TPTL	CFPS	eCPDP
JERIKO	FIR	0.432	0.349	**0.554**	0.390	0.507	0.482
	CIR	0.601	0.636	0.596	1.00	0.613	**0.289**
AEEEM	FIR	0.402	0.460	0.574	0.495	**0.741**	0.451
	CIR	0.607	0.548	0.543	0.678	0.543	**0.295**
JIRA	FIR	0.180	0.603	**0.737**	0.650	0.670	0.604
	CIR	0.781	0.405	0.385	0.517	0.396	**0.215**

Table 9 and 10 present the average performance of FIR and CIR over each dataset and project, respectively. In the case of FIR, as shown in Table 9, eCPDP is not the best technique among baselines. In addition, as shown in Table 10, CDE, TPTL, and CFPS win in most cases over various projects. However, as shown in Table 8, CDE, TPTL, and CFPS are not significantly different from eCPDP in terms of the overall performance distribution, so CDE, TPTL, and CFPS are not better than eCPDP in statistics. In other words, CDE, TPTL, and CFPS outperform eCPDP in specific projects but perform much worse than eCPDP in others. On the other hand, as shown in Table 8, eCPDP significantly outperforms FesCH and GIS, and it shows better performance than FesCH and GIS in most cases (i.e., 7 of 8 cases in Table 9 and 15 of 24 cases in Table 10). In summary, the effort efficiency of eCPDP is similar to CDE, TPTL, and CFPS. Also, eCPDP has better effort efficiency than FesCH and GIS over various datasets and projects. In the case of CIR, eCPDP outperforms in every datasets and projects. In addition, on average, eCPDP can save cost $\frac{1}{2}$ to $\frac{1}{3}$ of the cost compared to the baselines. In summary, the cost efficiency of eCPDP is 2 or 3 times better than baselines.

Answer to RQ2: eCPDP has similar or better performance than 5 state-of-the-art TL-based CPDP techniques on FIR and CIR. There- fore, applying eCPDP at the unit testing phase is viable from the perspective of effort and cost aware metrics.

Table 10 Average performance measure of eCPDP and 5 baseline models on FIR and CIRper each project

Dataset	Project	FesCH	GIS	CDE	TPTL	CFPS	eCPDP
(a) FIR							
JERIKO	Ant-1.7	0.485	0.427	**0.604**	0.397	0.332	0.481
	Poi-2.0	0.481	0.351	**0.614**	0.454	0.373	0.339
	Camel-1.4	0.320	0.349	0.358	**0.480**	0.402	0.450
	Ivy-2.0	0.557	0.339	0.681	0.401	**0.768**	0.543
	Jedit-4.0	0.417	0.273	0.513	**0.610**	0.533	0.424
	Log4j-1.0	0.405	0.400	0.683	0.213	0.580	**0.687**
	Xalan-2.4	0.449	0.428	0.454	0.454	**0.730**	0.516
	Velocity-1.6	0.328	0.080	**0.454**	0.450	-0.07	0.374
	Tomcat	0.559	0.572	0.679	0.440	**0.779**	0.644
	Xerces-1.3	0.529	0.439	0.589	0.242	**0.616**	0.526
	luccen-2.4	0.351	0.345	0.537	0.341	**0.570**	0.461
	synapse-1.2	0.351	0.345	**0.537**	0.341	0.570	0.461
AEEEM	Eclipse JDT	0.275	0.334	0.537	0.441	**0.602**	0.421
	Equinox	0.501	0.533	0.569	0.551	**0.753**	0.375
	Lucene	0.397	0.443	0.814	0.546	**0.877**	0.678
	Mylyn	0.483	0.487	0.363	0.457	**0.768**	0.282
	Eclipse PDE	0.353	0.506	**0.586**	0.480	0.706	0.500
JIRA	ActiveMQ	0.251	0.645	**0.811**	0.449	0.000	0.568
	Derby	0.003	0.639	0.717	0.409	**0.851**	0.742
	Groovy	0.229	0.532	0.781	**0.868**	0.838	0.675
	HBase	0.145	0.481	0.595	**0.639**	0.579	0.440
	Hive	0.370	0.642	0.744	0.633	**0.756**	0.651
	JRuby	0.056	0.627	0.758	**0.863**	0.840	0.554
	Wicket	0.207	0.655	0.753	0.662	**0.828**	0.596
Average		0.352	0.446	**0.611**	0.488	0.603	0.511
# of wins		0	0	7	5	11	**1**
(b) CIR							
JERIKO	Ant-1.7	0.550	0.598	0.564	1.109	0.668	**0.275**
	Poi-2.0	0.475	0.537	0.361	0.865	0.528	**0.264**
	Camel-1.4	0.583	0.576	0.520	1.119	0.510	**0.262**
	Ivy-2.0	0.418	0.541	0.331	1.010	0.321	**0.218**
	Jedit-4.0	0.618	0.734	0.616	0.496	0.724	**0.310**
	Log4j-1.0	0.652	0.635	0.672	1.475	0.636	**0.257**
	Xalan-2.4	0.530	0.532	0.504	1.219	0.438	**0.239**
	Velocity-1.6	0.852	0.937	0.947	0.825	1.023	**0.420**
	Tomcat	0.401	0.366	0.290	1.152	0.261	**0.165**
	Xerces-1.3	0.464	0.529	0.444	1.285	0.435	**0.232**
	luccen-2.4	0.935	0.797	1.058	0.922	1.050	**0.446**
	synapse-1.2	0.740	0.845	0.851	0.621	0.767	**0.377**
AEEEM	Eclipse JDT	0.929	0.863	1.048	0.648	1.125	**0.486**
	Equinox	0.536	0.527	0.493	0.411	0.530	**0.312**
	Lucene	0.486	0.451	0.247	0.787	0.271	**0.151**
	Mylyn	0.476	0.443	0.507	0.681	0.372	**0.293**
	Eclipse PDE	0.606	0.456	0.417	0.861	0.415	**0.232**
JIRA	ActiveMQ	0.710	0.408	0.432	1.006	0.467	**0.221**
	Derby	0.997	0.396	0.407	1.236	0.422	**0.187**
	Groovy	0.602	0.358	0.244	0.089	0.241	**0.148**
	HBase	0.840	0.559	0.534	0.319	0.551	**0.283**
	Hive	0.626	0.477	0.554	0.321	0.558	**0.257**
	JRuby	0.922	0.288	0.227	0.131	0.205	**0.208**
	Wicket	0.773	0.349	0.257	0.520	0.328	**0.198**
Average		0.655	0.555	0.523	0.796	0.535	**0.268**
# of wins		0	0	0	0	0	**12**

5.3 RQ3: Which classifier works the best with eCPDP?

Approach We conduct combination experiments with 5 classifiers that have been popularly applied in various CPDP studies. In addition, we execute parameter optimization using the grid search method. Table 11 summarizes the classifiers and parameter settings that we use. In order to reduce the randomness of each classifier, we execute each classifier 30 times per target project.

Findings Table 12 summarizes the average results on 7 different performance metrics. The bold fonts are the highest performance among 5 classifiers. First, the performance fluctuates significantly depending on applied classifiers. In the case of AUC, the performance difference between the smallest (i.e., NB) and the largest (i.e., LR) is 7.5%, whereas, in the case of F-measure, it reached 63%. So, we can re-confirm the importance of the selection of classifier suggested by Tantithamthavorn et al. (2018), Li et al. (2020a), and Kang et al. (2021). Second, LR is the best classifier in terms of conventional prediction performance metrics (i.e., AUC, G-mean, F-measure, Balance, MCC). In addition, LR shows better FIR and CIR than SVC that shows slightly worse than LR in the conventional prediction performance metrics. It indicates that LR has the best performance among other classifiers and reduces effort and cost compared to a model with similar performance. Finally, as for why eCPDP worked best for LR, we presume it can consider the linear combinations of original features twice for prediction. First, in the alignment stage of eCPDP, SVD extracts eigenvectors, linear combinations of original features, as shown in Figure 3. In other words, each eigenvector has information of all original features with different weights. Second, LR finds the best linear combination of the eigenvectors for classification in the training stage. In other words, LR refers to the compressed eigenvector once again for prediction, and it can see the effect of looking at several original features at once. In summary, through the 2 stage, eCPCP with LR can predict defective modules considering several original features at the same time, and we presume that it leads to have better performance than other classifiers.

Table 11 5 Different classifiers and parameter settings

Classifier	Parameter
LR (Logistic Regression)	C: [0.001, 0.01, 0.1, 1, 10, 100] penalty: [l1, l2, elastinet] solver: [‘newton-cg’, ’lbfgs’, ’liblinear’, ’sag’, ’saga’
NB (Naive Bayes)	var smoothing: [0, log−9, num=50]
RF (Random Forest)	n estimators: [10, 30, 60, 100] criterion: [’gini’, ’entropy’] min samples split: [2 ,4, 6, 8, 16, 20]
KNN (K-Nearest Neighbors)	n_neighbors : [1,2, ... 20] weights: [’uniform’, ’distance’] algorithm: [’ball_tree’, ’kd_tree’, ’brute’]
SVC (Support Vector Classification)	C: [0.001, 0.01, 0.1, 1, 10, 100] penalty: [’l1’, ’l2’] loss: [’hinge’, ’squared hinge’]

Table 12 Average performance of the 5 classifiers on 7 metrics

Metrics	NB	KNN	RF	SVC	LR
AUC	0.730	0.741	0.769	0.781	**0.786**
G-mean	0.547	0.417	0.457	0.664	**0.667**
F-measure	0.329	0.266	0.308	0.431	**0.435**
Balance	0.546	0.429	0.492	0.651	**0.652**
MCC	0.237	0.246	0.252	0.319	**0.321**
FIR	0.479	**0.679**	0.570	0.506	0.519
CIR	0.296	0.260	**0.257**	0.269	0.265

Answer to RQ3: Among various classifier, Logistic regression (LR) is the best classifier for eCPDP both on prediction performance and effort and cost efficiency.

6 Discussion

We devise the novel TL using SVD and apply it to eCPDP. However, as de- scribed in Section 3.1, only Z-score normalization affects distribution betweensource and target project, and the R-Align process does not affect distribution between them. Thus, it is necessary to study the effect of the process that does not affect data distribution on the prediction performance. In addition, we enhance eCPDP by applying Box-cox transformation and parameter optimization with grid-search. Thus, it is also necessary to study the effect of the enhancement on the prediction performance of eCPDP.

We generate 4 different models by eliminating one or two processes inthe entire framework to confirm the effect of each stage. Table 13 summarizes the description of the 4 models. By comparing Model1 and Model2, we can recognize the effect of Z-score normalization that alleviates distribution discrepancy between source and target project. By comparing Model2 and Model3, we can confirm the effect of the R-align process that does not affectdistribution between source and target. Finally, we can confirm the effect of Box-Cox transformation and parameter optimization process that are addedin the previous study (i.e., Model3) by comparing Model3 and Model4. The entire execution method is the same as RQ3.

Table 13 Description of the 4 Different Models

Name	Description
Model1	Baseline (Without Prepossessing and Alignment stage)
Model2	Model1 + Prepossessing using Z-score
Model3	Model2 + Alignment
Model4	Model3 + Box-cox transformation + Parameter optimization

Table 14 shows the average performance of 4 models over 24 projects on 7 metrics. The values between the parenthesis are the improvement percentage compared to the Model1. First, Model2 outperforms Model1 on every performance metric from 0.4% to 12.4%. It indicates that Z-score normalization, which alleviates distribution discrepancy between source and target project, improves prediction performance as well as cost and effort effectiveness. Second, Model3 outperforms Model2 on every metric except FIR. Though the R-align process degrades FIR performance by around 2%, it improves other metrics from 0.2% to 4%. Although the performance improvement is not significant, the R-align process, which does not affect distribution between source and target project, improves all other metrics except for FIR. Thus, it can be said that R-align also has a positive effect on the performance improvement. Third, Model4, which includes all processes, it improves every performance metrics from 0.4% to 4.3%. In addition, they improve FIR performance that was degraded by the R-align process in model 3. Thus, they positively affectthe performance of eCPDP and especially lead eCPDP to be more effort efficient. In summary, Z-score normalization, R-align, Box-cox transformation, and parameter optimization with grid-search are essential processes of eCPDP.

Table 14 Average performance of 4 models on 7 metrics

Metrics	Model1	Model2	Model3	Model4
AUC	0.726 (-)	0.763 (5.0%)	0.773 (6.4%)	**0.786 (8.2%)**
G-mean	0.636 (-)	0.644 (1.2%)	0.665 (4.5%)	**0.667 (4.8%)**
F-measure	0.403 (-)	0.415 (3.0%)	0.431 (7.0%)	**0.435 (8.0%)**
Balance	0.630 (-)	0.632 (0.4%)	0.650 (3.2%)	**0.653 (3.6%)**
MCC	0.272 (-)	0.305 (12.4%)	0.315 (16.2%)	**0.322 (18.4%)**
FIR	0.455 (-)	0.509 (11.9%)	0.500 (10.0%)	**0.520 (14.3%)**
CIR	0.279 (-)	0.274 (1.9%)	0.273 (2.1%)	**0.265 (5.0%)**

7 Threats to Validity

7.1 Internal Validity

This study includes 5 state of the art TL-based CPDP techniques. However, except TPTL, we implement the baselines according to the pseudo-code in each paper, it might not be same as the original technique. Also, this study applies grid search for parameter optimization, the best classifier of eCPDP may change according to parameter gird settings.

7.2 Construct Validity

This study applies FIR and CIR as effort and cost aware performance metrics, and a single metric for evaluating effort and cost can be a threat. However, to the best of our knowledge, FIR and CIR are only applicable metrics due to applicable time differences between eCPDP and existing techniques.

7.3 External Validity

This study uses 24 projects in 3 datasets that are publicly available and have been used in many other studies. In addition, they have diverse in size, feature, and percentage of defective instance, the diversity help draw generalization of out finding.

8 Conclusion

Existing TL-based CPDP techniques are not applicable at the unit testing phase since they require the entire historical target project data. As a result, they lose a chance to increase the product's reliability in the earlier phase. To address the issue, this study proposes eCPDP (i.e., a novel TL-based CPDP technique applicable at the unit testing phase) utilizing SVD that enables TL only using source project data.

This study evaluates the performance of eCPDP with 5 state-of-the-art TL-based CPDP techniques over 24 projects on 9 different performance metrics. The result shows that eCPDP has similar or better performance than the baseline on every performance metric. In conclusion, eCPDP is applicable at the unit testing phase and helps practitioners find and fix defects earlier than other TL-based CPDP techniques.

For future work, we will apply other types of matrix factorization techniques instead of SVD. Since this study confirms that SVD is an applicable method in CPDP literature, it is worth finding a more appropriate matrix factorization technique. In addition, it is also worth studying to find the best cut-off values. Since eCPDP can confirm the correctness of the prediction result at the unit testing phase, practitioners can adaptively change cut-off values toimprove prediction performance

Acknowledgements

This research was supported by the National Research Foundation of Korea (NRF-2020R1F1A1071888), the Ministry of Science and ICT (MSIT), Korea, under the Information Technology Research Center (ITRC) support program supervised by the Institute of Information & Communications Technology Planning & Evaluation (IITP-2021-2020-0-01795).

Declarations

Conflict of interest: The authors declare no competing interests

References

Arcuri A, Briand L (2011) A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: 2011 33rd International Conference on SoftwareEngineering (ICSE), IEEE, pp 1–10

Ba Q, Li X, Bai Z (2013) Clustering collaborative filtering recommendation system basedon svd algorithm. In: 2013 IEEE 4th International Conference on Software Engineering and Service Science, IEEE, pp 963–967

Bennin KE, Toda K, Kamei Y, Keung J, Monden A, Ubayashi N (2016) Empirical evalua- tion of cross-release effort-aware defect prediction models. In: 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, pp 214–221

Brunton SL, Kutz JN (2019) Data-driven science and engineering: Machine learning, dy- namical systems, and control. Cambridge University Press

Chen L, Fang B, Shang Z, Tang Y (2015) Negative samples reduction in cross-company software defects prediction. Information and Software Technology 62:67–77

Cohen J (1988) Statistical power analysis for the behavioral sciences–second edition. 12 lawrence erlbaum associates inc. Hillsdale, New Jersey 13

D'Ambros M, Lanza M, Robbes R (2010) An extensive comparison of bug prediction approaches. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), IEEE, pp 31–41

Gong L, Jiang S, Bo L, Jiang L, Qian J (2020) A novel class-imbalance learning approach for both within-project and cross-project defect prediction. IEEE Transactions on Reli- ability 69(1):40–54

Gretton A, Borgwardt KM, Rasch MJ, Schölkopf B, Smola A (2012) A kernel two-sample test. The Journal of Machine Learning Research 13(1):723–773

Herbold S (2013) Training data selection for cross-project defect prediction. Proceedings of the 9th international conference on predictive models in software engineering pp 1–10

Hosseini S, Turhan B, Mäntylä M (2018) A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. Information and Software Technology 95:296–312

Jureczko M, Madeyski L (2010) Towards identifying software project clusters with regardto defect prediction. In: Proceedings of the 6th international conference on predictive models in software engineering, pp 1–10

Kang J, Kwon S, Ryu D, Baik J (2021) Haspo: Harmony search-based parameter optimiza- tion for just-in-time software defect prediction in maritime software. Applied Sciences 11(5):2002

Kawata K, Amasaki S, Yokogawa T (2015) Improving relevancy filter methods for cross- project defect prediction. In: 2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence, IEEE, pp 2–7

Kwon S, Ryu D, Baik J (2021) ecpdp : Early cross-project defect prediction. In: 2021 21th IEEE international Conference on Software Quality, Reliability, and Security (QRS), p in press

Li K, Xiang Z, Chen T, Tan KC (2020a) Bilo-cpdp: bi-level programming for automated model discovery in cross-project defect prediction. In: 2020 35th IEEE/ACM Interna- tional Conference on Automated Software Engineering (ASE), IEEE, pp 573–584

Li K, Xiang Z, Chen T, Wang S, Tan KC (2020b) Understanding the automated param-eter optimization on transfer learning for cross-project defect prediction: an empirical study. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp 566–577

Li Z, Jing XY, Zhu X, Zhang H (2017) Heterogeneous defect prediction through multi-ple kernel learning and ensemble learning. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp 91–102

Limsettho N, Bennin KE, Keung JW, Hata H, Matsumoto K (2018) Cross project defect pre-diction using class distribution estimation and oversampling. Information and Software Technology 100:87–102
Liu C, Yang D, Xia X, Yan M, Zhang X (2019) A two-phase transfer learning model for cross-project defect prediction. Information and Software Technology 107:125–136
Mende T, Koschke R (2010) Effort-aware defect prediction models. In: 2010 14th EuropeanConference on Software Maintenance and Reengineering, IEEE, pp 107–116
Misra S, Adewumi A, Maskeliū nas R, Damaševičius R, Cafer F (2017) Unit testing in global software development environment. In: International Conference on Recent De- velopments in Science, Engineering and Technology, Springer, pp 309–317
Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In: 2013 35th international conferenceon software engineering (ICSE), IEEE, pp 382–391
Ni C, Liu W, Gu Q, Chen X, Chen D (2017) Fesch: a feature selection method using clustersof hybrid-data for cross-project defect prediction. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), IEEE, vol 1, pp 51–56
Panichella A, Alexandru CV, Panichella S, Bacchelli A, Gall HC (2016) A search-based training algorithm for cost-aware defect prediction. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp 1077–1084
Pascarella L, Palomba F, Bacchelli A (2019) Fine-grained just-in-time defect prediction. Journal of Systems and Software 150:22–36
Planning S (2002) The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology
Reddy MS, Adilakshmi T (2014) Music recommendation system based on matrix factor-ization technique-svd. In: 2014 International Conference on Computer Communication and Informatics, IEEE, pp 1–6
Shin Y, Meneely A, Williams L, Osborne JA (2010) Evaluating complexity, code churn, anddeveloper activity metrics as indicators of software vulnerabilities. IEEE transactionson software engineering 37(6):772–787
Sun Z, Li J, Sun H, He L (2021) Cfps: Collaborative filtering based source projects selectionfor cross-project defect prediction. Applied Soft Computing 99:106940
Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2018) The impact of au-tomated parameter optimization on defect prediction models. IEEE Transactions on Software Engineering 45(7):683–711
Turhan B, Menzies T, Bener AB, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineer- ing 14(5):540–578
Wilcoxon F (1946) Individual comparisons of grouped data by ranking methods. Journal of economic entomology 39(2):269–270
Xia X, Lo D, Pan SJ, Nagappan N, Wang X (2016) Hydra: Massively compositional model for cross-project defect prediction. IEEE Transactions on software Engineering 42(10):977–998
Xu Z, Pang S, Zhang T, Luo XP, Liu J, Tang YT, Yu X, Xue L (2019) Cross project defect prediction via balanced distribution adaptation based transfer learning. Journalof Computer Science and Technology 34(5):1039–1062
Yatish S, Jiarpakdee J, Thongtanunam P, Tantithamthavorn C (2019) Mining software defects: should we consider affected releases? In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 654–665
Yuan X, Han L, Qian S, Xu G, Yan H (2019) Singular value decomposition based recom-mendation using imputed data. Knowledge-Based Systems 163:485–494
Zhang F, Keivanloo I, Zou Y (2017) Data transformation in cross-project defect prediction. Empirical Software Engineering 22(6):3186–3218
Zhang H, Cheung SC (2013) A cost-effectiveness criterion for applying software defect pre-diction models. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp 643–646
Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) 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, pp 91–100