Cross Version Defect Prediction with Representative Data via Sparse Subset Selection

Zhou Xu
School of Computer, Wuhan University, Wuhan, China

Shuai Li, Yutian Tang, Xiapu Luo*
Department of Computing, The Hong Kong Polytechnic University, Hong Kong

Tao Zhang
College of Computer Science and Technology, Harbin Engineering University, Harbin, China

Jin Liu†
School of Computer, Wuhan University, Wuhan, China

Jun Xu
Department of Computing, The Hong Kong Polytechnic University, Hong Kong

ABSTRACT

Software defect prediction aims at detecting the defect-prone software modules by mining historical development data from software repositories. If such modules are identified at the early stage of the development, it can save large amounts of resources. **C**ross **V**ersion **D**efect **P**rediction (**CVDP**) is a practical scenario by training the classification model on the historical data of the prior version and then predicting the defect labels of modules of the current version. However, software development is a constantly-evolving process which leads to the data distribution differences across versions within the same project. The distribution differences will degrade the performance of the classification model. In this paper, we approach this issue by leveraging a state-of-the-art **D**issimilarity-based **S**parse **S**ubset **S**election (**DS**^{3}) method. This method selects a representative module subset from the prior version based on the pairwise dissimilarities between the modules of two versions and assigns each module of the current version to one of the representative modules. These selected modules can well represent the modules of the current version, thus mitigating the distribution differences. We evaluate the effectiveness of DS^3 for CVDP performance on total 40 cross-version pairs from 56 versions of 15 projects with three traditional and two effort-aware indicators. The extensive experiments show that DS^3 outperforms three baseline methods, especially in terms of two effort-aware indicators.

CCS CONCEPTS

• **Software and its engineering → Software creation and management**; **Software verification and validation**; *Software defect analysis*;

KEYWORDS

Cross version defect prediction, representative data, sparse subset selection, pairwise dissimilarities

*The corresponding author email: csxluo@comp.polyu.edu.hk
†The corresponding author email: jinliu@whu.edu.cn

ICPC '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5714-2/18/05…$15.00
https://doi.org/10.1145/3196321.3196331

ACM Reference Format:
Zhou Xu, Shuai Li, Yutian Tang, Xiapu Luo, Tao Zhang, Jin Liu, and Jun Xu. 2018. Cross Version Defect Prediction with Representative Data via Sparse Subset Selection. In *ICPC '18: ICPC '18: 26th IEEE/ACM International Confernece on Program Comprehension , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3196321.3196331

1 INTRODUCTION

Due to some unexpected mistakes in the design, development or configuration process, it is inevitable to introduce defects into the software products [14]. To fix the defects before releasing the products, effectively detecting them is essential for **S**oftware **Q**uality **A**ssurance (**SQA**). Thanks to the prevalence of the version control systems and the issue tracking systems in software development, it is easy to obtain the historical software development data from these software repositories. In the past decade, researchers have proposed various defect prediction methods for improving software quality. These methods analyze the historical data and explore the inherent relationship between the software metrics (such as code complexity metrics and process metrics) and the defect information which can be collected from the software repositories [25].

Many studies on defect prediction investigate the performance of various methods, especially the machine learning methods, for **W**ithin **P**roject **D**efect **P**rediction (**WPDP**) [15, 29, 56]. Generally, WPDP trains a classification model on labeled software modules, and then tests on the unlabeled modules within the same project. Most existing studies conduct WPDP by using cross-validation methods to partition the defect data of a specific version of a project into the training set and test set, which is also called **I**nner **V**ersion **D**efect **P**rediction (**IVDP**). For a mature project with multiple versions, it is more practical to use the historical defect data of the prior version to conduct defect prediction on the unlabeled modules (the methods, classes or packages) of the upcoming version (a.k.a. current version) [35], i.e., **C**ross **V**ersion **D**efect **P**rediction (**CVDP**). However, there are only a few studies on CVDP.

1.1 Motivation

Compared with IVDP, CVDP has some unique characteristics. More concretely, for IVDP, the training set and test set are usually derived from the same defect data of a specific project version, thus, the data distributions of the two sets are identical. By contrast, in CVDP scenario, as the software functions are increasingly complicated, the modules undergo frequent changes during the version update. For example, the current version of the project inherits, refactories

and deletes some existing modules from the prior version, or adds some new modules [6, 35]. These operations can cause a certain degree of data distribution differences across versions. Note that such data distribution differences may result in the majority of existing methods for CVDP failing to achieve satisfactory CVDP performance, because these methods use all the labeled data of the prior version to build classification models and then conduct defect prediction on the unlabeled modules of the current version.

Two recent studies considered the issue of data distribution differences for CVDP. Lu et al. [35] selected some candidate unlabeled modules from the current version with an active learning method, and then labeled these modules by querying the software experts before adding them into the prior version to form a mixed training set. They expected that these modules can alleviate the distribution differences by supplementing some distribution information of the current version into the prior version. However, there are two major limitations in this method. First, the candidate modules selected by the active learning method are only informative while not representative for the current version [20, 31]. Second, this method needs to query the labels of the candidate modules by the domain experts, thus, it involves extra labor cost. Another related study is conducted by Amasaki [1]. He selected a subset of the prior version with a nearest neighbor filter method [49] which is originally designed for **C**ross **P**roject **D**efect **P**rediction (**CPDP**). They found that the method is not applicable for improving CVDP performance. However, they do not explore the reasons behind.

In this paper, we utilize a new method to mitigate the effect of the data distribution differences, which screens representative modules from the prior version instead of picking up and labeling modules from the current version. In particular, we leverage a novel **D**issimilarity-based **S**pare **S**ubset **S**election (**DS^3**) method [11] to achieve this purpose. DS^3 selects a representation module subset from the prior version based on the pairwise dissimilarities between the modules of the two versions. The selected subset can effectively represent each module of the current version, which promotes to narrow the gap of the data distribution differences across versions. Since we preserve the modules of the prior version that can well represent the modules of the current version and eliminate the irrelevant ones, the model built on this selected subset tends to obtain higher performance compared with the model built on all the modules of the prior version. In addition, we replicate the work in [1] and perform a further analysis for the reasons of inapplicability.

To evaluate the effectiveness of the DS^3 method, we choose 56 versions of 15 projects as our benchmark dataset and conduct extensive experiments on total 40 cross-version pairs with five evaluation indicators, including three traditional and two effort-aware indicators. The three traditional indicators, i.e., F-measure, g-mean and Balance, do not consider the cost of quality assurance effort required to review the modules [7, 49, 63]. As inspecting all the modules is not always feasible due to the constraint of the test resources, we further employ two effort-aware indicators, i.e., **E**ffort-**A**ware **Recall** (**EARecall**) and **E**ffort-**A**ware **F-measure** (**EAF-measure**). We calculate the two indicators by improving a new module ranking method recently proposed in [19].

By using the 40 cross-version pairs, DS^3 achieves average F-measure, g-mean, Balance, EARecall and EAF-measure of 0.347, 0.451, 0.476, 0.345 and 0.313, respectively. The experimental results show that DS^3 performs better than three baseline methods (i.e., ALL method, Turhan Filter, and Peter Filter), especially in terms of the two effort-aware indicators. More specifically, the average EARecall with DS^3 improves by 15.0%, 16.3%, and 22.5%, while the average EAF-measure with DS^3 improves by 13.2%, 15.5%, and 18.3% compared with the three baseline methods, respectively.

1.2 Contributions

In this paper, we highlight the following main contributions:

(1) Since the data distribution differences between two versions have adverse impacts on the CVDP performance, in this paper, we address this issue by leveraging a state-of-the-art DS^3 method to select a representative module subset from the prior version as the new training set. The refined module subset can well represent the modules of the current version.
(2) We perform extensive experiments on total 40 cross-version pairs, derived from 56 versions of 15 projects. To the best of our knowledge, this is the first work to conduct such a large-scale empirical study for CVDP.
(3) We employ five indicators, including three typical and two effort-aware ones, as our performance measurement. This enables us to perform a comprehensive evaluation on the effectiveness of the selected module subset by DS^3 for CVDP. In addition, we improve a novel module ranking method to calculate the effort-aware indicators.
(4) Compared with three baseline methods, the experiments manifest that DS^3 achieves encouraging results with fewer modules (no more than 50% of the original set) in most cases, especially in terms of the two effort-aware indicators. In addition, a further analysis shows that DS^3 can alleviate the class imbalance issue to some extent by increasing the proportion of defective modules in the training set on most cross-version pairs.

2 RELATED WORK

2.1 Cross Version Defect Prediction

Bennin et al. [4] evaluated the defect prediction performance of 11 basic classification models in IVDP and CVDP scenarios with an effort-aware indicator. They conducted experiments on 25 projects (each one has two versions with process metrics) and found that the optimal models for the two defect prediction scenarios are not identical due to different data as the training set. However, the performance differences of the 11 models are not significant in both scenarios. Premraj et al. [46] investigated the impacts of code and network metrics on the defect prediction performance of six classification models. They considered three scenarios, including IVDP, CVDP and CPDP. CPDP uses the defect data of another project as the training set. Experiments on three projects (each with two versions) suggested that the network metrics are better than the code metrics in most cases. Holschuh et al. [18] explored the performance of CVDP on a large software system by collecting four types of metrics. The experiments on six projects (each with three versions) showed that the overall performance is unsatisfactory. Monden et al. [40] evaluated the cost effectiveness of defect prediction on three classification models by comparing seven test effort allocation strategies. The results on one project with five versions revealed that the reduction of test effort relied on the appropriate test strategy. Khoshgoftaar et al. [28] studied the performance of

six classification models on one project with four versions and found that CART model with lease absolute deviation performed the best. Zhao et al. [62] investigated the relationship between the context-based cohesion metrics and the defect-proneness in IVDP and CVDP scenarios. They conducted CVDP study on four projects with total 19 versions and found that context-based cohesion metrics had negative impacts on defect prediction performance but can be complementary to non-context-based metrics. Yang et al [61] surveyed the impacts of code, process and slice-based cohesion metrics on defect prediction performance in IVDP, CVDP and CPDP scenarios. They conducted CVDP study on one project with seven versions and found that slice-based cohesion metrics had adverse impacts on defect prediction performance but can be complementary to the commonly used metrics. Wang et al. [50] explored the performance of their proposed semantic metrics on defect prediction in CVDP and CPDP scenarios. The experiments on ten projects with 26 versions showed the superiority of the semantic metrics compared with traditional CK metrics and AST metrics.

These studies fed all modules of the prior version into the classification model without taking the data distribution differences into account and conducted experiments on small datasets. Although two recent studies considered the distribution differences issue, they exist some limitations as presented in Section 1.1. Different from the above studies, in this paper, we introduce a novel subset selection method that can be well suitable for CVDP and use a larger benchmark dataset with total 56 versions of 15 projects.

2.2 Training subset selection

Although no studies have proposed a customized training subset selection for CVDP, there are some related studies for CPDP which focus on selecting a module subset from a project (a.k.a. source project) to build the classification model for the modules of another project (a.k.a. target project). Turhan et al. [49] proposed a nearest neighbor filter method, called **T**urhan **F**ilter (**TF**), to select a module subset of the source project. More specifically, for each module in the target project, TF first selects its top-k' nearest modules, then these candidate modules without duplication constitute the training set. They found that CPDP with TF achieved similar classification performance compared with IVDP on some cross-project pairs. Peter et al. [45] proposed a module selection strategy, called **P**eter **F**ilter (**PF**), with a clustering algorithm. More concretely, PF first combined the data of the two projects, then used k-means clustering algorithm to cluster the mixed data and discarded the clusters that did not contain any module of the target project. For each module of the target project in the remaining clusters, the method selected its nearest neighbor module in the source project. These selected modules were fed into the classification model. The results showed that PF led to the performance improvement compared with the TF method and IVDP performance. Kawata et al. [27] proposed a relevancy filter method, called **K**awata **F**ilter (**KF**), for source project data simplification. KF first used the DBSCAN algorithm to cluster the mixed project data. For the clusters that contained at least one module of the target project, the modules of the source project in such clusters formed the final training data.

All these methods were designed for CPDP and took Euclidean distance or data density as filter criterion to select the modules from the source project as the training set. However, the distributions between diverse projects have significant difference [22]. In CVDP scenario, since the current version usually carries plenty of information from the prior version [9], the distribution differences across versions usually are smaller than that across projects. This is an essential distinction between CVDP and CPDP. In addition, although Amasaki [1] has manifested that TF is not applicable for CVDP, the applicability of other methods for CVDP is not clear yet.

In this work, we introduce an advanced subset selection method, called DS^3, for CVDP. Different from the above methods, DS^3 utilizes the pairwise dissimilarities (not limit to the Euclidean distance) between the modules of two versions to minimize the representing cost towards the modules of the current version by the modules of the prior version. In addition, the above subset selection methods for CPDP could not determine the number of selected modules, while DS^3 can achieve it by tuning a control parameter.

3 THE DS^3 METHOD

We denote the modules of the prior version as $\mathbf{S} = [s_1, s_2, ..., s_m]$, where $s_i = [s_{i1}, s_{i2}, ..., s_{ir}]^{\mathrm{T}} \in \mathbb{R}^r$, m and r indicate the number of modules and metrics of the prior version, respectively. Similarly, we denote the modules of the current version as $\mathbf{T} = [t_1, t_2, ..., t_n]$, where $t_j = \left[t_{j1}, t_{j2}, ..., t_{jq}\right]^{\mathrm{T}} \in \mathbb{R}^q$, n and q indicate the number of modules and metrics of the current version, respectively. Note that r is equal to q in our CVDP scenario. In addition, we define a dissimilarity matrix $\mathbf{D} = \left[d_{ij}\right]_{i=1,...,m}^{j=1,...,n}$ between the modules of the two versions, where d_{ij} indicates the dissimilarity between the module s_i and t_j which indicates how well s_i represents t_j. Here, the dissimilarity can be defined as the Euclidean, Hamming, Manhattan or Chi-square distance between the pairwise modules across the two versions. More specifically, smaller d_{ij} means that s_i represents t_j better. The purpose of DS^3 is to find a representative module subset of **S** that can effectively represent each module of **T**.

Dissimilarity matrix **D** is formulated as

$$\mathbf{D} = \begin{bmatrix} \mathbf{d_1^{\mathrm{T}}} \\ \vdots \\ \mathbf{d_m^{\mathrm{T}}} \end{bmatrix} = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{1}$$

where $\mathbf{d}_i \in \mathbb{R}^n$ is the ith row of **D**, i.e., the dissimilarities between the ith module of **S** and each module of **T**.

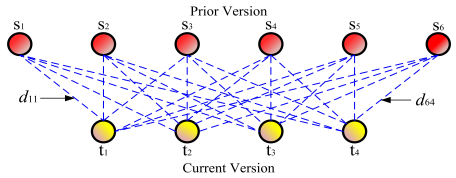
To indicate whether a module in **S** is a representative of the module in **T**, we define a 0-1 indicator matrix **P** as

$$\mathbf{P} = \begin{bmatrix} \mathbf{p_1^{\mathrm{T}}} \\ \vdots \\ \mathbf{p_m^{\mathrm{T}}} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{2}$$

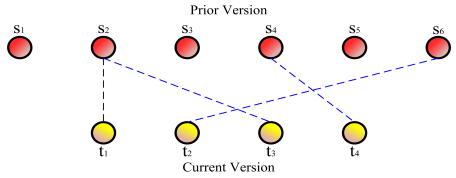
where $p_{ij} \in \{0, 1\}$ is the indicator of s_i representing t_j. $p_{ij} = 1$ indicates that s_i is a representative of t_j while $p_{ij} = 0$ indicates that s_i is not a representative of t_j. In order to guarantee that each t_j is represented by one s_i, we have the constrain as $\sum_{i=1}^{m} p_{ij} = 1$.

To have a better understanding of the above description, we provide a graphic depiction in Figure 1. Each red circle denotes a module of the prior version while each yellow circle represents a module of the current version, and the dotted line signifies the correlation (i.e., dissimilarity in DS^3) between two modules. Note

that if there exists a dotted line between a red circle and a blue circle after conducting DS^3 method, it means that the red circle can well represent the yellow circle. Subfigure 1(a) shows the inputs of DS^3 method, i.e., the pairwise dissimilarities between the modules of the two versions. Subfigure 1(b) depicts the results after running DS^3 method. It shows that DS^3 method selects three modules (i.e., s_2, s_4 and s_6) from the prior version as the representatives to represent each module of the current version. Note that there is only one dotted line connecting with each yellow circle since we assign each module of the current version with only one module of the prior version. However, there can have multiple dotted lines connecting with each red circle since each module of the prior version can represent multiple modules of the current version, for example, s_2 can represent t_1 and t_3.



(a) Pairwise dissimilarities between two versions



(b) Selected modules that represent the current version

Figure 1: Illustration of the function of DS^3.

To select a representative subset of **S** based on the dissimilarity matrix **D**, DS^3 simultaneously optimizes the following two terms: minimize the representing cost towards **T** by **S** and the number of selected modules from **S** . In terms of the first term, if s_i is selected as a representative of t_j, then the representing cost towards t_j by s_i is calculated as $d_{ij}p_{ij} \in \{0, d_{ij}\}$ (since $p_{ij} \in \{0, 1\}$). Further, the representing cost towards y_i by **S** is calculated as $\sum_{i=1}^{m} d_{ij}p_{ij}$, and the representing cost towards **T** by **S** is calculated as $\sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij}$. In terms of the second term, if s_i is a representative towards some modules of **T**, then not all elements in the ith row of **P** are zeros, i.e., $\sum_{j=1}^{n} p_{ij} \neq 0$. Thus, restricting the number of the selected representative modules means to control the number of nonzero rows in the indicator matrix **P**. From the above description, we can clearly see that the optimization problem turns to solve the element p_{ij} in the indicator matrix **P**.

The optimization problem is formulated as the following row-sparsity regularized trace minimization problem

$$\begin{aligned} \min_{\{p_{ij}\}} \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij} + \lambda \sum_{i=1}^{m} \mathbb{I}(\|p_i\|_2) \\ \text{s.t.} \quad & \sum_{i=1}^{m} p_{ij} = 1, \forall j; \quad p_{ij} \in \{0, 1\}, \forall i, j, \end{aligned} \tag{3}$$

where $\|\cdot\|_2$ denotes the l_2 norm, $\mathbb{I}(\cdot)$ denotes the indicator function that is equal to 0 if all the elements of p_i ($i \in [1, 2, ..., m]$) are zeros and is equal to 1 otherwise. The first term $\sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij}$ refers to the representing cost towards **T** by **S**, while the second term denotes the number of selected representatives from **S** which corresponds to the number of nonzero rows in matrix **P**. The parameter λ is used to make a trade-off between the two terms. Small λ value indicates that more emphases are put on lower representing cost towards **T** by **S**, thus more representative modules will be selected.

However, Eq.(3) is a non-convex optimization problem since it involves to count the sum of a set of indicator function values (i.e., the term $\sum_{i=1}^{m} \mathbb{I}(\|p_i\|_2)$ constrained by $p_{ij} \in \{0, 1\}$) of **P**. Elhamifar et al. [11] suggested to solve the above optimization problem as the following convex relaxation

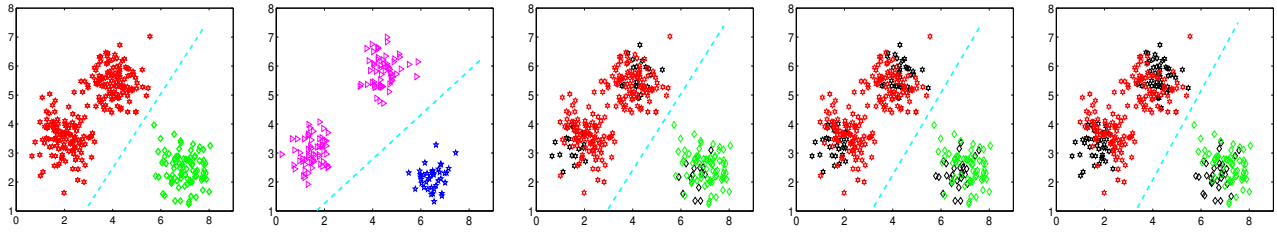
$$\begin{aligned} \min_{\{p_{ij}\}} \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}p_{ij} + \lambda \sum_{i=1}^{m} \|p_i\|_2 \\ \text{s.t.} \quad & \sum_{i=1}^{m} p_{ij} = 1, \forall j; \quad p_{ij} \in [0, 1], \forall i, j. \end{aligned} \tag{4}$$

Compared with Eq.(3), Eq.(4) calculates the sum of the l_2 norm of all rows in matrix **P** instead of the sum of the indicator function values and uses the non-negative $p_{ij} \in [0, 1]$ to relax the term $p_{ij} \in \{0, 1\}$. From this perspective, p_{ij} can be treated as the probability of t_j represented by s_i. Eq.(4) can be rewritten as the matrix form

$$\begin{aligned} \min_{\mathbf{P}} \quad & \text{tr}(\mathbf{D}^{\mathrm{T}}\mathbf{P}) + \lambda\|\mathbf{P}\|_2 \\ \text{s.t.} \quad & \mathbf{1}^{\mathrm{T}}\mathbf{P} = \mathbf{1}^{\mathrm{T}}, \mathbf{P} \in [0, 1], \end{aligned} \tag{5}$$

where $\|\mathbf{P}\|_2 = \sum_{i=1}^{m} \|p_i\|_2$, $\mathbf{1}$ is a vector whose elements are all 1, and $\text{tr}(\cdot)$ denotes the trace function used to calculate the inner product of two matrices. Eq.(5) can be solve by the **A**lternating **D**irection **M**ethod of **M**ultipliers (**ADMM**) framework [5, 13]. After obtaining the optimal solution $\widehat{\mathbf{P}}$, the line numbers of the nonzero rows in matrix $\widehat{\mathbf{P}}$ correspond to the indexes of the modules in **S** that are selected to represent the modules of **T**. For example, for Subfigure 1(b) that selected three representative modules of the prior version (i.e., **S**) to represent the four modules of the current version (i.e., **T**), the optimal solution $\widehat{\mathbf{P}}$ is like the following formula $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$, where the values in the positions with 1 are nonzero. The line numbers of the nonzero rows in matrix $\widehat{\mathbf{P}}$ is 2, 4, 6, which indicates that the second (i.e., s_2), the fourth (i.e., s_4) and the sixth (i.e., s_6) module of the prior version (i.e., **S**) are selected as the representatives.

To have an intuitive feeling about the effect of DS^3 for selecting the representative modules, we conduct a case study on two synthetic datasets to simulate CVDP scenario. We generate the non-defective modules of the prior version by drawing data points (red hexagrams) from a mixture of Gaussians with means (2,3.5) and (4,5.5) with 120 points in each set, and the defective modules by drawing 90 data points (green rhombuses) from a mixture of Gaussians with means (6.5,2.5), as showed in Subfigure 2(a). To reflect the distribution differences between two versions, we generate the non-defective modules of the current version by drawing data points (purple triangles) from a mixture of Gaussians with means



(a) Prior Version (b) Current Version (c) $\lambda = 0.005$ (d) $\lambda = 0.001$ (e) $\lambda = 0.0001$

Figure 2: An illustrative example for Representative modules selected by DS^3 with different λ on synthetic data.

(1.5,3) and (4.5,6) with 60 points in each set, and the defective modules by drawing 40 data points (blue pentagrams) from a mixture of Gaussians with means (6.5,2.5), as showed in Subfigure 2(b). Subfigures 2(c), 2(d), and 2(e) depict the selected non-defective modules (black hexagrams) and defective modules (black rhombuses) from the prior version by DS^3 with three different λ values. From the last three subfigures, we can see that the selected modules are close to the positions of the modules in 2(b). In addition, we observe that as the λ decreases, the number of selected modules by DS^3 increases.

4 EXPERIMENTAL SETUP

4.1 Benchmark Dataset

In this work, we use 56 versions of 15 software projects provided by Madeyski and Jureczko [37] as our benchmark dataset. Each module denotes a class file of the Java project and is characterized by 20 software metrics and a binary label for the defect proneness (i.e., a defective module is labeled as 1, otherwise as 0). The detailed descriptions of these metrics are available in [24].

Detailed statistic description of each version for all 15 projects is reported in Table 1, where # M, # DM, % DM denote the number of modules, the number of defective modules and the percentage of defective modules, respectively.

4.2 Evaluation Indicators

In this section, we describe the five evaluation indicators, including three traditional indicators and two effort-aware indicators, used to measure the performance of CVDP. In terms of the traditional evaluation indicators, we choose F-measure, g-mean and Balance, which are widely used in previous defect prediction studies [17, 22, 23, 30, 39, 43, 47, 48, 51, 52, 55, 57]. The three indicators are derived from the basic indicators listed in Table 2 and defined as

$$\text{F-measure} = \frac{2 * \text{recall} * \text{precision}}{\text{recall} + \text{precision}} \tag{6}$$

$$\text{g-mean} = \sqrt{(\frac{\text{TN}}{\text{TN} + \text{FP}}) * (\frac{\text{TP}}{\text{TP} + \text{FN}})} \tag{7}$$

$$\text{Balance} = 1 - \sqrt{\frac{(0 - \text{FPR})^2 + (1 - \text{TPR})^2}{2}} \tag{8}$$

The three indicators do not consider the quality assurance effort required to review the modules by assuming that there are enough resources to inspect all the modules [7, 49, 63]. However, inspecting all the modules is not always practical due to the limited test resources. As suggested by Mende et al. [38], it is more realistic to use indicators that consider the inspection effort (also known as effort-aware indicators) in defect prediction.

Effort-aware indicators evaluate the defect prediction performance within a limited effort required to review the predicted defective modules [3, 26, 60]. In reality, we always want to maximize the benefit of any effort for quality assurance. Generally, the effort denotes the number of **L**ines **O**f **C**ode (**LOC**) that need to be inspected, and the benefit is the number or percentage of defective modules discovered. In this work, we set the number of LOC reviewed as 20% of total LOC following previous studies [21, 53, 58].

To calculate the effort-aware indicators, the common method is to rank the modules according to a rule first, and then simulate an

Table 1: Statistic of the Benchmark Dataset

Project	# M	# DM	% DM	Project	# M	# DM	% DM
ant-1.3	126	20	15.9%	xalan-2.4	723	110	15.2%
ant-1.4	178	40	22.5%	xalan-2.5	803	387	48.2%
ant-1.5	293	32	10.9%	xalan-2.6	885	411	46.4%
ant-1.6	352	92	26.1%	xalan-2.7	909	898	98.8%
ant-1.7	745	166	22.3%	xerces-init	162	77	47.5%
camel-1.2	608	216	35.5%	xerces-1.2	440	71	16.1%
camel-1.4	872	145	16.6%	xerces-1.3	453	69	15.2%
camel-1.6	965	188	19.5%	xerces-1.4.4	588	437	74.3%
ivy-1.1	111	63	56.8%	prop2-ver225	1864	147	7.9%
ivy-1.4	241	16	6.6%	prop2-ver236	2403	76	3.2%
jEdit-3.2.1	272	90	33.1%	prop2-ver245	2023	103	5.1%
jEdit-4.0	306	75	24.5%	prop2-ver256	2025	625	30.9%
jEdit-4.1	312	79	25.3%	prop2-ver-265	2372	229	9.7%
jEdit-4.2	367	48	13.1%	prop3-ver285	1709	177	10.4%
jEdit-4.3	492	11	2.2%	prop3-ver292	2330	209	9.0%
log4j-1.0	135	34	25.2%	prop3-ver305	2388	89	3.7%
log4j-1.1	109	37	33.9%	prop3-ver318	2440	365	15.0%
log4j-1.2	205	189	92.2%	prop4-ver347	2906	162	5.6%
lucene-2.0	195	91	46.7%	prop4-ver355	2802	924	33.0%
lucene-2.2	247	144	58.3%	prop4-ver362	2865	213	7.4%
lucene-2.4	340	203	59.7%	prop5-ver4	3514	264	7.5%
poi-1.5	237	141	59.5%	prop5-ver40	3815	466	12.2%
poi-2.5.1	385	248	64.4%	prop5-ver85	3509	930	26.5%
poi-3.0	442	281	63.6%	prop5-ver121	3445	425	12.3%
synapse-1.0	157	16	10.2%	prop5-ver157	2863	367	12.8%
synapse-1.1	222	60	27.0%	prop5-ver185	3260	268	8.2%
velocity-1.4	196	147	75.0%	prop42-ver452	317	33	10.4%
velocity-1.5	214	142	66.4%	prop42-ver453	259	20	7.7%

Table 2: Basic Indicators for Defect Prediction

	# Predicted defective	# Predicted non-defective
# Actual defective	**T**rue **P**ositive (**TP**)	**F**alse **N**egative (**FN**)
# Actual non-defective	**F**alse **P**ositive (**FP**)	**T**rue **N**egative (**TN**)
True **P**ositive **R**ate (**TPR**) or recall	$\frac{\text{TP}}{\text{TP+FN}}$	
False **P**ositive **R**ate (**FPR**)	$\frac{\text{FP}}{\text{FP+TN}}$	
precision	$\frac{\text{TP}}{\text{TP+FP}}$	

expert to inspect these modules one at a time in order. In the meanwhile, the percentage of LOC reviewed and the number of detected defective modules are counted. The process is terminated when 20% of total LOC have been inspected. The proportion of inspected defective modules among all the actual defective modules is treated as an effort-aware indicator, which is called **PofB**20 (**P**ercentage **of B**ugs) or **CE**20 (**C**ost **E**ffectiveness) in [21, 53, 59, 61].

In previous studies, researchers ranked the modules in a descending order based on the degree of their predicted risk values. The risk values are defined as the probability outputs of a classification model for the modules [21, 38, 53] or the ratios of the probability outputs to LOC of the corresponding modules [58, 59, 61]. Recently, Huang et al. [19] proposed a novel ranking method to calculate the effort-aware indicators for defective change prediction, called **C**lassifier **B**efore **S**orting (**CBS**). Their experimental results have shown that the derived indicator values significantly improved based on this ranking method. The basic idea of CBS is that among the changes that are predicted to be potentially defective by a classifier, small changes that are measured by the modified LOC should be inspected first, since they give the best bang for the buck [19]. However, this ranking method can not obtain the indicator values in some cases. More specifically, CBS only ranks the predicted defective changes without considering the other changes. Therefore, if the sum of the modified LOC of these ranked changes does not reach 20% of total modified LOC, the effort-aware indicators could not be calculated. To remedy this limitation, in this work, we propose an improved ranking method based on CBS to rank the modules in our CVDP scenario. More specially, we divide the modules into two parts (i.e., the predicted defective and non-defective modules) according to their predicted labels by the logistic regression classifier. First, we rank the predicted defective modules in a ascending order based on their LOC values, this process is identical to CBS. Then we rank the predicted non-defective module with the same process. Finally, we concatenate the latter ranking results behind the former ranking results. This ensures to always calculate the two effort-aware indicator values.

We concisely describe how to calculate the two effort-aware indicators. Let the current version of a given project have n modules and n_1 defective modules. After inspecting 20% of total LOC based on our improved ranking method, n' modules and $n_1{}'$ defective modules have been reviewed.

The first effort-aware indicator is defined as the proportion of the inspected defective modules among all actual defective modules, which is called Recall by Huang et al. [19]. To distinguish it from the traditional Recall indicator, we name it **E**ffort-**A**ware **Recall** (**EARecall**). EARecall is defined as

$$\text{EARecall} = \frac{n_1{}'}{n_1} \tag{9}$$

Higher EARecall value indicates that more defective modules can be detected when 20% of total LOC are inspected.

In addition, we define **E**ffore-**A**ware **Precision** (**EAPrecision**) which is equal to the proportion of the inspected defective modules among all inspected modules, i.e., EAPrecision = $\frac{n_1{}'}{n'}$. Higher EAPrecision signifies lower false alarm which will enhance the developers' confidence on the CVDP performance when inspecting 20% of total LOC [19].

Given the definition of EARecall and EAPrecision, another effort-aware indicator, **E**ffort-**A**ware **F-measure** (**EAF-measure**) is defined as

$$\text{EAF-measure} = \frac{2 * \text{EARecall} * \text{EAPrecision}}{\text{EARecall} + \text{EAPrecision}} \tag{10}$$

The worst case is that the inspected modules are all non-defective modules, then the values of EARecall and EAPrecision are all equal to 0. Thus, the value of EAF-measure makes no sense since the denominator in Eq.(10) is 0. In this case, we set the value of EAF-measure as 0 which indicates the worst CVDP performance.

4.3 Classification Model

In this work, we select the logistic regression classifier as our basic learner. This classifier is extensively used in previous defect prediction studies [29, 32–34, 41–44, 53, 58, 59, 61, 63]. We implement this classifier with the LIBLINEAR package [12], a well-known package for solving large-scale problems with the coordinate descent algorithm. We run the package with the options "-S 0" (which denotes the logistic regression) and "-B 1" (which means no bias term added) following the previous work [32–34, 43].

4.4 DS^3 configuration

Regarding the implementation of DS^3, we measure the dissimilarity with Chi-square distance following the original work [11]. The Chi-square distance of module s_i and t_j is defined as $\sum_{o=1}^{r} \frac{(s_{io}-t_{jo})^2}{2*(s_{io}+t_{jo})}$, where r denotes the feature dimension. In terms of the λ which is used to control the number of selected modules, lower λ value means more representative modules will be chosen. In this work, we determine the λ value based on a threshold which denotes the desired proportion of the selected modules. More specifically, we set the initial value of λ as 0.05 and gradually reduce it until the proportion of the selected modules is larger than the threshold for the first time. We empirically set six thresholds, i.e., 20%, 30%,..., 70%, to determine the values of λ. According to the above description, the final proportion of the selected modules may be a little higher than the corresponding threshold.

4.5 Cross Version Scenario Design

In this work, we conduct the CVDP experiment between two nearest versions. More specifically, for the ant project in benchmark dataset, the selected modules by DS^3 from version 1.3 are used to train the logistic regression classifier. Then this model is tested on the modules in version 1.4.

5 EXPERIMENTAL RESULTS

RQ1: Could the modules selected by DS^3 achieve better CVDP performance than the whole modules of the prior version?
Method: As the aim of this work is to pick up a representative module subset of the prior version that well describes the modules of the current version, this question investigates whether the elaborately selected modules by DS^3 can achieve better or comparative CVDP performance than the whole modules of the prior version. Therefore, we choose the method that uses all data of the prior version to conduct CVDP as our baseline method, which is called **ALL**. For the results analysis, Wilcoxon signed-rank test [16] is used to examine whether the differences between DS^3 and ALL are statistically significant at a confidence level of 95% across all cross-version pairs. This test is applicable to check two variables whose

Table 3: Detailed Results of for the Five Indicators for DS^3 and ALL Method on Each Cross-Version Pair

Cross-Version Pair			F-measure		g-mean		Balance		EARecall		EAF-measure		L	T
			All	DS3	All	DS3	All	DS3	All	DS3	All	DS3		
ant	1.3	1.4	0.143	**0.230**	0.302	**0.397**	0.361	**0.412**	0.100	**0.175**	0.151	**0.246**	1.8E-03	70%
	1.4	1.5	0.308	**0.324**	0.534	**0.576**	0.510	**0.551**	0.344	**0.375**	0.163	**0.324**	2.7E-03	60%
	1.5	1.6	0.350	**0.419**	0.471	**0.525**	0.454	**0.492**	0.196	**0.250**	0.308	**0.380**	1.1E-03	70%
	1.6	1.7	**0.525**	0.511	**0.643**	0.625	**0.605**	0.585	0.247	**0.253**	0.340	**0.357**	1.4E-03	70%
camel	1.2	1.4	0.314	**0.386**	0.504	**0.598**	0.488	**0.577**	0.241	**0.331**	0.283	**0.336**	1.6E-03	**50%**
	1.4	1.6	**0.214**	0.172	**0.360**	0.315	**0.387**	0.364	0.255	**0.436**	0.200	**0.242**	2.8E-03	**30%**
ivy	1.1	1.4	0.160	**0.189**	0.592	**0.635**	0.591	**0.635**	0.250	0.250	0.085	**0.104**	1.8E-03	**40%**
jEdit	3.2.1	4.0	0.509	**0.510**	**0.668**	0.657	**0.660**	0.640	0.373	**0.413**	0.373	**0.434**	1.9E-03	**40%**
	4.0	4.1	0.603	**0.615**	0.680	**0.695**	0.632	**0.649**	0.380	**0.405**	0.513	**0.529**	1.4E-03	**50%**
	4.1	4.2	0.533	**0.542**	0.764	**0.766**	0.748	**0.750**	0.417	**0.438**	0.396	**0.420**	2.0E-04	70%
	4.2	4.3	0.195	**0.250**	0.586	**0.657**	0.548	**0.613**	0.364	**0.455**	0.200	**0.250**	5.0E-04	60%
log4j	1.0	1.1	**0.656**	0.646	**0.721**	0.716	**0.689**	0.687	0.351	**0.378**	0.473	**0.491**	6.0E-03	**40%**
	1.1	1.2	0.441	0.441	**0.500**	0.360	**0.487**	0.363	0.164	**0.185**	0.279	**0.302**	2.4E-02	**20%**
lucene	2.0	2.2	0.598	**0.678**	**0.612**	0.422	**0.605**	0.433	0.306	**0.556**	0.429	**0.544**	2.6E-02	**20%**
	2.2	2.4	**0.730**	0.682	0.318	**0.492**	0.368	**0.489**	**0.557**	0.507	**0.542**	0.536	4.0E-03	**30%**
poi	1.5	2.5.1	**0.844**	0.838	0.727	**0.742**	0.702	**0.725**	0.544	**0.548**	0.638	**0.651**	1.1E-03	60%
	2.5.1	3.0	**0.789**	0.787	0.639	**0.756**	0.619	**0.755**	**0.505**	0.459	0.577	**0.592**	2.6E-03	**40%**
synapse	1.0	1.1	0.313	**0.420**	0.451	**0.556**	0.444	**0.533**	0.167	**0.250**	0.253	**0.330**	2.6E-02	**20%**
velocity	1.4	1.5	**0.777**	0.766	0.198	**0.275**	0.321	**0.349**	**0.697**	0.690	0.656	0.656	1.0E-02	**20%**
xalan	2.4	2.5	0.162	**0.187**	0.297	**0.321**	0.358	**0.369**	0.080	**0.093**	0.145	**0.166**	1.2E-02	**20%**
	2.5	2.6	0.558	**0.572**	**0.603**	0.490	**0.599**	0.489	0.294	**0.423**	0.360	**0.391**	6.0E-03	**30%**
	2.6	2.7	0.576	**0.656**	**0.636**	0.597	0.579	**0.591**	0.238	**0.292**	0.385	**0.451**	8.0E-03	**20%**
xerces	init	1.2	**0.286**	0.281	**0.489**	0.465	**0.486**	0.466	0.676	**0.690**	**0.265**	0.260	1.1E-03	70%
	1.2	1.3	**0.152**	0.082	**0.293**	0.208	**0.354**	0.324	0.072	**0.551**	0.128	**0.180**	8.0E-03	**20%**
	1.3	1.4.4	0.209	**0.212**	0.342	**0.344**	0.375	**0.377**	0.101	**0.105**	0.183	**0.190**	2.5E-03	**40%**
prop2	225	236	0.038	**0.054**	0.195	**0.226**	0.320	**0.330**	0.013	**0.026**	0.013	**0.029**	4.0E-03	**20%**
	236	245	0.036	**0.070**	0.139	**0.197**	0.307	**0.320**	0.379	**0.398**	0.061	**0.066**	6.0E-04	**40%**
	245	256	**0.019**	0.009	**0.098**	0.069	**0.300**	0.296	0.365	**0.413**	0.277	**0.299**	1.1E-03	**50%**
	256	265	0.208	**0.222**	**0.518**	0.508	**0.509**	0.497	**0.262**	0.258	0.171	**0.193**	1.4E-03	**40%**
prop3	285	292	0.182	**0.209**	0.324	**0.352**	0.367	**0.381**	0.201	**0.211**	0.077	**0.086**	1.0E-04	70%
	292	305	0.161	**0.197**	0.333	**0.380**	0.372	**0.396**	**0.180**	0.146	0.054	**0.197**	1.0E-04	70%
	305	318	0.047	**0.057**	0.157	**0.173**	0.310	**0.314**	**0.436**	0.430	0.191	**0.201**	3.0E-04	60%
prop4	347	355	**0.024**	0.021	**0.109**	0.104	0.301	0.301	0.326	**0.342**	0.265	**0.271**	1.1E-03	**20%**
	355	362	0.152	**0.220**	**0.491**	0.467	**0.487**	0.456	0.155	**0.183**	0.086	**0.185**	8.0E-04	**30%**
prop5	4	40	0.083	0.083	0.212	0.212	0.325	0.325	0.511	**0.534**	0.170	0.170	3.9E-03	**20%**
	40	85	**0.121**	0.096	**0.254**	0.225	**0.339**	0.329	0.427	**0.438**	**0.305**	0.302	1.9E-03	**30%**
	85	121	**0.338**	0.329	**0.560**	0.526	**0.534**	0.502	**0.268**	0.242	0.275	**0.280**	1.8E-03	**40%**
	121	157	**0.220**	0.203	**0.369**	0.348	**0.391**	0.380	**0.134**	0.123	**0.213**	0.203	1.8E-03	**30%**
	157	185	0.358	**0.385**	0.522	**0.543**	0.490	**0.506**	0.243	**0.265**	0.323	**0.351**	4.0E-03	**20%**
prop42	452	453	0.235	**0.316**	0.438	**0.534**	0.434	**0.504**	0.200	**0.300**	0.242	**0.316**	2.0E-02	**20%**
Average			0.329	**0.347**	0.441	**0.451**	0.469	**0.476**	0.300	**0.345**	0.276	**0.313**		
W/D/L			24/2/14		22/1/17		23/2/15		31/1/8		34/2/4			
p			0.012		0.255		0.202		1.55E-04		3.26E-07			
d			0.067		0.038		0.04		0.193		0.149			

distributions are unclear [2, 10]. The difference is significant if the p value is lower than 0.05. Further, effect size, Cliff's Delta [36], is applied to qualify the amount of the difference. The difference is substantial if the d value is greater than or equal to 0.146 [8, 53, 58].

Results: Since we set six thresholds to control the proportion of selected modules, we obtain six sets of results for each indicator on each cross-version pair. In this work, as we emphasize the importance and practicability of the effort-aware indicators on CVDP performance especially for the EAF-measure, we only report the results correspond to the best EAF-measure value among the six sets of results for each cross-version pair. The detailed results under each threshold can be found in our online supplemental materials [54]. Table 3 reports the detailed results of the five evaluation indicators for DS^3 and ALL method on each cross-version pair. For the ease of replicating our experimental results, we also report the threshold and λ value corresponding to the best EAF-measure. The source codes and benchmark dataset of this work are available in our online materials. In Table 3, the best indicator values among DS^3 and ALL are in bold while the same values are shaded in gray. The three numbers in **W**in/**D**raw/**L**oss (**W/D/L**) denote the counts of indicator values by DS^3 larger than, equal to and lower than that by ALL, respectively. From Table 3, we have the following observations:

First, in terms of the three traditional indicators, DS^3 achieves the better average values on the three indicators across the 40 cross-version pairs compared with ALL. More specifically, the W/D/L values show that DS^3 is better than ALL method on 24, 22, and 23 cross-version pairs respectively, but the average values of the

three indicators by DS^3 only gain small improvements against ALL. In addition, the p values signify that the difference between DS^3 and ALL is statistically significant for F-measure while not significant for g-mean and Balance, and the d values indicate that the differences on the three indicators are not substantial.

Second, in terms of the two effort-aware indicators, DS^3 also achieves the highest average values against ALL. More concretely, in terms of EARecall, the W/D/L value shows that DS^3 is superior to ALL on 31 cross-version pairs and the average EARecall by DS^3 gains the improvement by 15.0% against ALL. In terms of EAF-measure, the W/D/L value shows that DS^3 outperforms ALL on 34 cross-version pairs and the average EAF-measure by DS^3 makes the improvement by 13.2% against ALL. In addition, the p values and d values indicate that the differences between DS^3 and ALL for the two indicators are statistically significant and substantial.

Third, in terms of the threshold which indicates the approximate percentage of the selected modules, we observe that DS^3 obtains the best EAF-measure values with thresholds no less than 50% (as in bold) on 29 out of 40 cross-version pairs. It indicates that DS^3 can select only a small proportion of modules of the prior version to achieve encouraging CVDP performance on effort-aware indicators and comparative CVDP performance on traditional indicators in most cases.

Discussion: The takeaway lesson of the third finding is that when conducting CVDP, it is not necessary to use all the modules of the prior version to train the defect prediction model for the current version, since some modules may be the noisy modules. These useless modules may have no discriminant ability or even negative impacts towards the classification task. A representative module subset of the prior version can replace the whole original data to construct an effective prediction model in most cases. The reason is that each module of the current version is assigned to a representative module of the module set selected by DS^3, the modules of the prior version that may not effectively represent the modules of the current version are eliminated. Therefore, the model built on the module subset has the potential to improve the CVDP performance compared with the method using all modules of the prior version.

However, the CVDP performance of the model built on the module subset is not superior to that built on all modules in some cases, such as the cross-version pairs on prop5 project. The potential reason is that the module subset sometimes may cause information loss by removing some informative modules, which will decrease the CVDP performance of the model. For example, considering the cross-version pairs on prop5 project, the last column shows that the proportion of the selected modules are between 20% and 40%. Compared with the all modules, the subset may contain less information. However, from the indicator values, we can observe that only in a few cases, the differences between DS^3 and ALL are relatively large. While in other cases, the indicator values of the two methods are similar. Since using a module subset to conduct CVDP can save a large amount of memory and computing overheads, the advantage of DS^3 will become more apparent when the number of modules of the prior version increases.

Answer: To summary, DS^3 is effective in selecting a representative subset for encouraging and comparable CVDP performance against the whole modules of the prior version in most cross-version pairs, especially in terms of the two effort-aware indicators.

RQ2: Are the representative modules selected by DS^3 more effective than that by other subset selection methods?

Method: Since various subset selection methods will select different modules as the representatives, this question explores how effective the selected modules by DS^3 are compared with that by other subset selection methods. A sample method for selecting the module subset is random sampling. The main drawback of random sampling is that it may select a subset that only consists of non-defective modules. This is due to the class imbalance issue in most defect data, i.e., the non-defective modules comprise the majority. Therefore, when the module set is highly imbalanced, it is a high probability that random sampling fails to select any defective module. In that case, we could not build a feasible classification model. In addition, since the modules are selected without any criterion, the performance of the resultant model will be unstable. Hence, we do not choose random sampling as our baseline method. However, to the best of our knowledge, there have no existing subset selection methods that are tailored for CVDP. In this work, we select some typical subset selection methods that are designed for CPDP task (as mentioned in Section 2.2) as our baseline methods. Thus, this question can also investigate whether these methods are appropriate for CVDP. In CVDP scenario, the modules of the prior (current) version are treated as the ones of the source (target) project in CPDP scenario. The first baseline method is the TF method [49]. We follow the original work to set k' as 10. Although Amasaki [1] has verified that this method was not helpful for improving CVDP performance, but they did not analyze the reasons. We replicate this method for CVDP on our benchmark dataset and explore the reasons in **RQ3**. The second baseline method is the PF method [45]. We follow the original study to set the parameter k of k-means algorithm as $\frac{m+n}{10}$ (where m and n denote the number of modules in the prior and current version respectively), expecting that each cluster tends to have 10 modules in average. Note that the k-means algorithm needs to randomly initialize the central points, we run PF 30 times and record the average indicator values. In addition, we also implement the KF method [27]. This method involves two

Table 4: Average Indicator Values and Statistic Values of The Five Indicators for DS^3 and The Two Baseline Methods

Indicators	Methods	Average	W/D/L	p	d
F-measure	DS3	**0.347**			
	TF	0.326	24/4/12	0.004	0.076
	PF	0.310	35/1/4	4.00E-06	0.114
g-mean	DS3	**0.451**			
	TF	0.436	21/2/17	0.218	0.051
	PF	0.436	27/1/12	0.024	0.056
Balance	DS3	**0.476**			
	TF	0.464	22/2/16	0.207	0.046
	PF	0.465	27/1/12	0.043	0.043
EARecall	DS3	**0.345**			
	TF	0.297	31/2/7	6.90E-05	0.206
	PF	0.282	34/0/6	9.00E-06	0.247
EAF-measure	DS3	**0.313**			
	TF	0.271	36/0/4	1.52E-07	0.175
	PF	0.264	40/0/0	4.29E-08	0.204

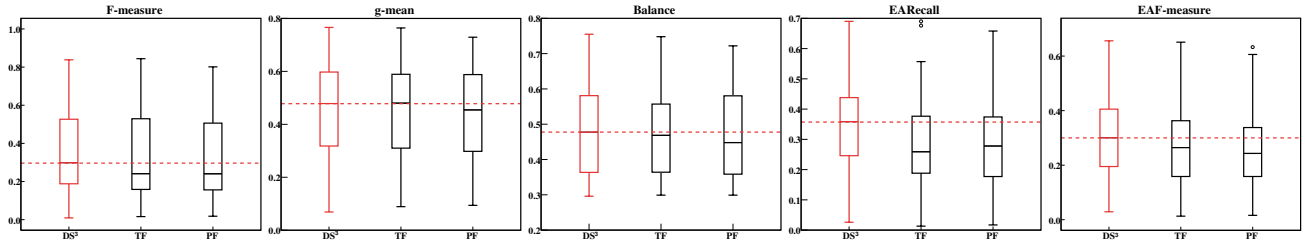


Figure 3: Box-plots of five indicators for DS^3 and two subset selection baseline methods.

parameters, the number of minimum records and the distance. We implement KF following the parameter setting in the original study and the default setting in Weka tool. The experiment show that, in some cross-version pairs, KF under both settings only selects very few modules as the training set since they treat many modules as noise and discard them. However, insufficient training set may lead the model overfitting. In addition, KF under the two settings only selects the non-defective modules as the training set in some pairs. Further, we conduct more experiments with different parameter settings and find that the results of KF are very sensitive to the settings. Thus, we do not choose KF as our baseline method.

Results: Due to the space limit, we only present the average indicator values and the statistic values in Table 4, and the box-plots of the five indicators for DS^3 and the two baseline methods across all pairs in Figure 3. Detailed results are available in our online materials. From the table and figure, we have the following findings:

First, in terms of the three traditional indicators, Table 4 shows that DS^3 achieves the best average values compared with the two baseline methods. More specifically, the W/D/L values show that DS^3 is better than the two methods on more than half of cross-version pairs for all three indicators; compared with the two baseline methods, the average F-measure by DS^3 gains the improvements by 6.3% and 11.9% respectively, the average g-mean by DS^3 makes the improvements by 3.6% and 3.5% respectively,while the average Balance by DS^3 achieves the improvements by 2.6% and 2.5% respectively. The p values signify that the differences on F-measure between DS^3 and two baseline methods, and the differences on g-mean and Balance between DS^3 and PF are significant. However, the d values signify that the differences are not substantial.

Second, in terms of EARecall, Table 4 shows that DS^3 obtains better performance on 31 and 34 out of 40 cross-version pairs than the two baseline methods, respectively; the average EARecall by DS^3 gains the improvements by 16.3% and 22.5% against TF and PF, respectively. In terms of EAF-measure, DS^3 obtains better performance on 36 and all cross-version pairs than TF and PF, respectively; the average EAF-measure by DS^3 makes the improvements by 15.5% and 18.3% against TF and PF, respectively. In addition, the p values and d values indicate that the differences on the two indicators between DS^3 and the two baseline methods are statistically significant and substantial.

Third, Figure 3 shows that, in terms of the three typical indicators, the median values of the three indicators by DS^3 are higher than that by PF, the median values of the g-mean and Balance by DS^3 are nearly the same as that by TF; in terms of the two effort-aware indicators, the median values of the EARecall and EAF-measure by DS^3 are much higher than that by the two baseline methods.

Discussion: The principles of DS^3 and the two baseline methods are different. DS^3 selects the subset by solving an optimization problem involving two terms. The first term utilizes the pairwise dissimilarities to minimize the representing cost, instead of simply selecting the neighbors of the modules of the current version as the two baseline methods do. DS^3 assigns each module of the current version to a representative module of the prior version, therefore, the selected modules by DS^3 can better characterize the modules of the current version. The second term is used to constrain the number of the selected modules, while the two baseline methods can not determine how many modules are selected. This means that DS^3 can control the proportion of the training set according to the practice conditions, such as the constraints of the memory and computing power, thus DS^3 is more flexible to conduct subset selection for CVDP.

Answer: To sum up, the module set selected by DS^3 is more effective to improve CVDP performance than the modules selected by the two baseline methods, especially in terms of the two effort-aware indicators.

RQ3: What are the differences between the modules selected by DS^3 and that by the two baseline methods?

Method: The subset selection methods differ in the modules selected. In this question, we further explore the differences among the module subsets of the prior version selected by the three methods in **RQ2**. Here, we mainly focus on the number of the selected modules by these methods. In general, the method that selects fewer modules as training set to achieve satisfactory CVDP performance is more preferred.

Results: To answer this question, we record the number of modules selected by the three methods. In addition, as most defect data are imbalanced, this may lead a method to select a subset that only contains defective modules. In this case, this method will loss its purpose to select an effective training set from the prior version since only one type of modules can not construct a discriminable model. Therefore, we also record the number of selected defective modules. Table 5 reports the detailed results, where % D denotes the percentage of the defective modules in original data, # SM, # SD, % SD denote the number of selected modules, the number of selected defective modules and the percentage of selected defective modules, respectively. From Table 5, we have the following observations:

First, in terms of TF, from Table 1 and Table 5, we find that TF almost selects all the modules of the prior version as the training

Table 5: Detailed Statistic of Selected Modules by DS^3 and Two Subset Selection Baseline Methods

Cross-Version Pair			ALL	DS3			TF			PF		
			% D	# SM	# SD	% SD	# SM	# SD	% SD	# SM	# SD	% SD
ant	1.3	1.4	15.9%	90	14	15.6%	126	20	15.9%	112	18	16.1%
	1.4	1.5	22.5%	107	24	22.4%	178	40	22.5%	137	31	22.6%
	1.5	1.6	10.9%	206	19	9.2%	293	32	10.9%	257	24	9.3%
	1.6	1.7	26.1%	250	60	24.0%	352	92	26.1%	298	75	25.2%
camel	1.2	1.4	35.5%	305	119	**39.0%**	599	214	35.7%	442	161	36.4%
	1.4	1.6	16.6%	265	57	**21.5%**	863	144	16.7%	655	107	16.3%
ivy	1.1	1.4	56.8%	45	23	51.1%	111	63	56.8%	83	49	59.0%
jEdit	3.2.1	4.0	33.1%	112	35	31.3%	272	90	33.1%	239	77	32.2%
	4.0	4.1	24.5%	155	38	24.5%	305	75	24.6%	254	57	22.4%
	4.1	4.2	25.3%	237	58	24.5%	311	79	25.4%	254	63	24.8%
	4.2	4.3	13.1%	231	32	**13.9%**	367	48	13.1%	280	37	13.1%
log4j	1.0	1.1	25.2%	55	16	**29.1%**	133	34	25.6%	93	23	24.7%
	1.1	1.2	33.9%	22	8	**36.4%**	109	37	33.9%	96	31	32.3%
lucene	2.0	2.2	46.7%	40	27	**67.5%**	194	91	46.9%	169	74	43.8%
	2.2	2.4	58.3%	88	59	**67.0%**	247	144	58.3%	202	112	55.4%
poi	1.5	2.5.1	59.5%	144	84	58.3%	237	141	59.5%	165	95	57.6%
	2.5.1	3.0	64.4%	158	80	50.6%	369	232	62.9%	265	145	54.7%
synapse	1.0	1.1	10.2%	35	7	**20.0%**	155	16	10.3%	104	13	12.5%
velocity	1.4	1.5	75.0%	40	36	**90.0%**	195	147	75.4%	113	91	80.5%
xalan	2.4	2.5	15.2%	151	29	**19.2%**	719	109	15.2%	545	78	14.3%
	2.5	2.6	48.2%	263	146	**55.5%**	801	387	48.3%	636	313	49.2%
	2.6	2.7	46.4%	199	111	**55.8%**	811	348	42.9%	636	263	41.4%
xerces	init	1.2	47.5%	118	62	**52.5%**	162	77	47.5%	97	44	45.4%
	1.2	1.3	16.1%	98	17	**17.3%**	434	71	16.4%	326	46	14.1%
	1.3	1.4.4	15.2%	188	37	**19.7%**	447	69	15.4%	266	45	16.9%
prop2	225	236	7.9%	376	45	**12.0%**	1778	144	8.1%	960	101	10.5%
	236	245	3.2%	1005	51	**5.1%**	2169	76	3.5%	954	45	4.7%
	245	256	5.1%	1019	76	**7.5%**	1998	103	5.2%	1517	79	5.2%
	256	265	30.9%	821	243	29.6%	1914	618	32.3%	906	301	33.2%
prop3	285	292	10.4%	1305	145	**11.1%**	1555	172	11.1%	761	97	12.7%
	292	305	9.0%	1646	186	**11.3%**	2146	208	9.7%	1308	147	11.2%
	305	318	3.7%	1476	70	**4.7%**	2247	88	3.9%	1700	66	3.9%
prop4	347	355	5.6%	622	53	**8.5%**	2708	153	5.6%	1243	60	4.8%
	355	362	33.0%	868	260	30.0%	2699	906	33.6%	1812	591	32.6%
prop5	4	40	7.5%	704	74	**10.5%**	3302	256	7.8%	1688	151	8.9%
	40	85	12.2%	1147	157	**13.7%**	3653	408	11.2%	2297	223	9.7%
	85	121	26.5%	1436	426	**29.7%**	3429	892	26.0%	2464	666	27.0%
	121	157	12.3%	1060	211	**19.9%**	3181	419	13.2%	1494	261	17.4%
	157	185	12.8%	597	133	**22.3%**	2752	366	13.3%	1498	233	15.6%
prop42	452	453	10.4%	73	13	**17.8%**	292	30	10.3%	177	20	11.3%

set in most cases, which will lead TF to achieve the similar performance as the ALL method. Thus TF could not improve the CVDP performance which is consistent with the conclusion in [1].

Second, in terms of the number of selected modules, we find that DS^3 selects the fewest modules on 36 out of 40 cross-version pairs compared with the two baseline methods, which signifies that DS^3 is more effective for data reduction while still maintaining the great performance as mentioned in **RQ2**.

Third, in terms of the percentage of selected defective modules, the percentage values for DS^3 that are larger than the original percentages (i.e., reported in % D) are in bold. We find that DS^3 improves the percentage of defective modules in the new training set on 28 out of 40 cross-version pairs. In addition, PF improves the percentage of defective modules on 20 pairs while the percentage values of the modules selected by TF are quite similar to that of the original data on almost all pairs.

Discussion: Since the defective modules will cause the software failure or incorrect outcomes [14], in general, SQA team expects to detect more defective modules at the early stage of the software development, although it may increase the probability of false alarm (i.e., a non-defective module is predicted as defective one). Increasing the percentage of the defective modules in the training set is conducive to the trained model bias to the defective modules, especially for the imbalanced defect data. It means that, to a certain extent, DS^3 and PF can promote to relieve the class imbalance of the training set for better CVDP performance.

Answer: Overall, DS^3 can choose fewer modules from the prior version as the representative ones compared with the two baseline methods on most cross-version pairs. In addition, DS^3 can alleviate the class imbalance issue by improving the percentage of the defective modules in the selected module subset on most cross-version pairs.

6 THREATS TO VALIDITY

Threats to External Validity: External Validity focuses on the generalization of the experimental conclusions to other datasets. To minimize the threats, we choose 56 versions of 15 projects to conduct the CVDP experiments. However, the metrics of the benchmark dataset are at the code level, thus, whether our findings are identical to other datasets with process metrics, network metrics and text metrics is unclear. In addition, the 15 projects are developed with Java language, experiments on other projects that developed with C, C++, or Python will be performed in future.

Threats to Internal Validity: Internal Validity pays attention to the impacts of classification models, parameter settings on the conclusions. Here, we only use the logistic regression classifier, the effectiveness of the selected module subset on other classifiers need to be further explored since the performance of various classifiers has significant differences [15]. Regarding DS^3 method, there is only one parameter, i.e., λ, needs to be tuned. This parameter is related to the number of the selected modules. We empirical set six thresholds to determine this parameter. A fine-grained threshold setting will be considered in our future work.

Threats to Construct Validity: Construct Validity relates to the suitability of the evaluation indicators. Here, we choose three typical and two effort-aware indicators as our measurement criteria, which enables us to have a comprehensive evaluation of the CVDP performance with different module subsets as the training data.

7 CONCLUSION

In this work, we introduce a new subset selection method DS^3 to relieve the distribution differences between cross-version data for CVDP. Given the pairwise dissimilarities between the modules across versions, DS^3 finds a module subset of the prior version to well represent the modules of the current version by solving a row-sparsity regularized trace minimization problem. As illustrated in Section 5.1 and 5.2, the experiments on total 40 cross-version pairs show that the module subset selected by DS^3 achieves better CVDP performance compared with all data and the subsets selected by two baseline methods, especially in terms of effort-aware indicators.

Our future work involves employing more classifiers and defect datasets to enrich our experiments. In addition, we also plan to investigate the applicability of DS^3 for CPDP scenario and other software engineering problems, such as test case selection.

ACKNOWLEDGMENT

The authors acknowledge the support provided by the grants of the National Natural Science Foundation of China (61572374, 61472423, U1636220, 61602258), Academic Team Building Plan for Young Scholars from Wuhan University (WHU2016012), China Postdoctoral Science Foundation (No.2017M621247), Hong Kong GRF (152279/16E, 152223/17E), Hong Kong RGC Project (CityU C1008-16G), and Hong Kong ITF (ITS/197/17FP).

REFERENCES

[1] Sousuke Amasaki. 2017. On Applicability of Cross-project Defect Prediction Method for Multi-Versions Projects. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 93–96.

[2] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*. IEEE, 1–10.

[3] Erik Arisholm, Lionel C Briand, and Eivind B Johannessen. 2010. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software* 83, 1 (2010), 2–17.

[4] Kwabena Ebo Bennin, Koji Toda, Yasutaka Kamei, Jacky Keung, Akito Monden, and Naoyasu Ubayashi. 2016. Empirical Evaluation of Cross-Release Effort-Aware Defect Prediction Models. In *Proceedings of the International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 214–221.

[5] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning* 3, 1 (2011), 1–122.

[6] Gemma Catolino, Fabio Palomba, Andrea De Lucia, Filomena Ferrucci, and Andy Zaidman. 2017. Developer-related factors in change prediction: an empirical assessment. In *Proceedings of the 25th International Conference on Program Comprehension*. IEEE Press, 186–195.

[7] Lin Chen, Bin Fang, Zhaowei Shang, and Yuanyan Tang. 2015. Negative samples reduction in cross-company software defects prediction. *Information and Software Technology* 62 (2015), 67–77.

[8] Norman Cliff. 2014. *Ordinal methods for behavioral data analysis*. Psychology Press.

[9] Hoa Khanh Dam, Truyen Tran, Trang Pham, Shien Wee Ng, John Grundy, and Aditya Ghose. 2017. Automatic feature learning for vulnerability prediction. *arXiv preprint arXiv:1708.02368* (2017).

[10] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* 7, Jan (2006), 1–30.

[11] Ehsan Elhamifar, Guillermo Sapiro, and S Shankar Sastry. 2016. Dissimilarity-based sparse subset selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38, 11 (2016), 2182–2197.

[12] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research* 9, Aug (2008), 1871–1874.

[13] Daniel Gabay and Bertrand Mercier. 1976. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications* 2, 1 (1976), 17–40.

[14] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Computing Surveys (CSUR)* 50, 4 (2017), 1–36.

[15] Baljinder Ghotra, Shane McIntosh, and Ahmed E Hassan. 2015. Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1 (ICSE)*. IEEE Press, 789–800.

[16] Jean Dickinson Gibbons and Subhabrata Chakraborti. 2011. Nonparametric statistical inference. In *International Encyclopedia of Statistical Science*. Springer, 977–979.

[17] Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. 2012. An investigation on the feasibility of cross-project defect prediction. *Automated Software Engineering* 19, 2 (2012), 167–199.

[18] Tilman Holschuh, Markus Pauser, Kim Herzig, Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. 2009. Predicting defects in SAP Java code: An experience report. In *Proceedings of the 31st International Conference on Software Engineering-Companion Volume*. IEEE, 172–181.

[19] Qiao Huang, Xin Xia, and David Lo. 2017. Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 159–170.

[20] Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. 2014. Active Learning by Querying Informative and Representative Examples. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10, 36 (2014), 1936–1949.

[21] Tian Jiang, Lin Tan, and Sunghun Kim. 2013. Personalized defect prediction. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 279–289.

[22] Xiaoyuan Jing, Fei Wu, Xiwei Dong, Fumin Qi, and Baowen Xu. 2015. Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 496–507.

[23] Xiao-Yuan Jing, Shi Ying, Zhi-Wu Zhang, Shan-Shan Wu, and Jin Liu. 2014. Dictionary learning based software defect prediction. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. ACM, 414–423.

[24] Marian Jureczko and Lech Madeyski. 2010. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. ACM, 1–10.

[25] Yasutaka Kamei and Emad Shihab. 2016. Defect prediction: Accomplishments and future challenges. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 5. IEEE, 33–45.

[26] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering* 39, 6 (2013), 757–773.

[27] Kazuya Kawata, Sousuke Amasaki, and Tomoyuki Yokogawa. 2016. Improving relevancy filter methods for cross-project defect prediction. In *Applied Computing & Information Technology*. Springer, 1–12.

[28] Taghi M Khoshgoftaar and Naeem Seliya. 2003. Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering* 8, 3 (2003), 255–283.

[29] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering* 34, 4 (2008), 485–496.

[30] Ming Li, Hongyu Zhang, Rongxin Wu, and Zhi-Hua Zhou. 2012. Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering* 19, 2 (2012), 201–230.

[31] Xin Li and Yuhong Guo. 2013. Adaptive active learning for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 859–866.

[32] Zhiqiang Li, Xiao-Yuan Jing, Fei Wu, Xiaoke Zhu, Baowen Xu, and Shi Ying. 2017. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. *Automated Software Engineering* (2017), 1–45.

[33] Zhiqiang Li, Xiao-Yuan Jing, Xiaoke Zhu, and Hongyu Zhang. 2017. Heterogeneous Defect Prediction Through Multiple Kernel Learning and Ensemble Learning. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 91–102.

[34] Zhiqiang Li, Xiao-Yuan Jing, Xiaoke Zhu, Hongyu Zhang, Baowen Xu, and Shi Ying. 2017. On the Multiple Sources and Privacy Preservation Issues for Heterogeneous Defect Prediction. *IEEE Transactions on Software Engineering* (2017).

[35] Huihua Lu, Ekrem Kocaguneli, and Bojan Cukic. 2014. Defect prediction between software versions with active learning and dimensionality reduction. In *Proceedings of the 25th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 312–322.

[36] Guillermo Macbeth, Eugenia Razumiejczyk, and Rubén Daniel Ledesma. 2011. Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica* 10, 2 (2011), 545–555.

[37] Lech Madeyski and Marian Jureczko. 2015. Which process metrics can significantly improve defect prediction models? An empirical study. *Software Quality Journal* 23, 3 (2015), 393–422.

[38] Thilo Mende and Rainer Koschke. 2010. Effort-aware defect prediction models. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR)*. IEEE, 107–116.

[39] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering* 33, 1 (2007), 2–13.

[40] Akito Monden, Takuma Hayashi, Shoji Shinoda, Kumiko Shirai, Junichi Yoshida, Mike Barker, and Kenichi Matsumoto. 2013. Assessing the cost effectiveness of fault prediction in acceptance testing. *IEEE Transactions on Software Engineering* 39, 10 (2013), 1345–1357.

[41] Jaechang Nam, Wei Fu, Sunghun Kim, Tim Menzies, and Lin Tan. 2017. Heterogeneous defect prediction. *IEEE Transactions on Software Engineering* (2017).

[42] Jaechang Nam and Sunghun Kim. 2015. Clami: Defect prediction on unlabeled datasets. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 452–463.

[43] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. 2013. Transfer defect learning. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. IEEE Press, 382–391.

[44] Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2014. Cross-project defect prediction models: L'union fait la force. In *Proceedings of the Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*. IEEE, 164–173.

[45] Fayola Peters, Tim Menzies, and Andrian Marcus. 2013. Better cross company defect prediction. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 409–418.

[46] Rahul Premraj and Kim Herzig. 2011. Network versus code metrics to predict defects: A replication study. In *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 215–224.

[47] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. 2012. Recalling the imprecision of cross-project defect prediction. In *Proceedings of the 20th International Symposium on the Foundations of Software Engineering (FSE)*. ACM, 1–11.

[48] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. 2011. A general software defect-proneness prediction framework. *IEEE Transactions on Software Engineering* 37, 3 (2011), 356–370.

[49] Burak Turhan, Tim Menzies, Ayşe B Bener, and Justin Di Stefano. 2009. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering* 14, 5 (2009), 540–578.

[50] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. ACM, 297–308.

[51] Shuo Wang and Xin Yao. 2013. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability* 62, 2 (2013), 434–443.

[52] Tiejian Wang, Zhiwu Zhang, Xiaoyuan Jing, and Liqiang Zhang. 2016. Multiple kernel ensemble learning for software defect prediction. *Automated Software Engineering* 23, 4 (2016), 569–590.

[53] Xin Xia, David Lo, Sinno Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. 2016. Hydra: Massively compositional model for cross-project defect prediction. *IEEE Transactions on software Engineering* 42, 10 (2016), 977–998.

[54] Zhou Xu, Shuai Li, Yutian Tang, Xiapu Luo, Tao Zhang, Jin Liu, and Jun Xu. 2018. supplemental materials. (2018). https://icpc-ds3.github.io/icpc2018/

[55] Zhou Xu, Jin Liu, Xiapu Luo, and Tao Zhang. 2018. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In *Proceedings of the 25th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, to appear.

[56] Zhou Xu, Jin Liu, Zijiang Yang, Gege An, and Xiangyang Jia. 2016. The impact of feature selection on defect prediction performance: An empirical comparison. In *27th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 309–320.

[57] Zhou Xu, Jifeng Xuan, Jin Liu, and Xiaohui Cui. 2016. MICHAC: Defect Prediction via Feature Selection based on Maximal Information Coefficient with Hierarchical Agglomerative Clustering. In *Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 370–381.

[58] Xinli Yang, David Lo, Xin Xia, Yun Zhang, and Jianling Sun. 2015. Deep learning for just-in-time defect prediction. In *Proceedings of the International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 17–26.

[59] Yibiao Yang, Mark Harman, Jens Krinke, Syed Islam, David Binkley, Yuming Zhou, and Baowen Xu. 2016. An empirical study on dependence clusters for effort-aware fault-proneness prediction. In *Proceedings of the 31st International Conference on Automated Software Engineering (ASE)*. IEEE, 296–307.

[60] Yibiao Yang, Yuming Zhou, Jinping Liu, Yangyang Zhao, Hongmin Lu, Lei Xu, Baowen Xu, and Hareton Leung. 2016. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. ACM, 157–168.

[61] Yibiao Yang, Yuming Zhou, Hongmin Lu, Lin Chen, Zhenyu Chen, Baowen Xu, Hareton Leung, and Zhenyu Zhang. 2015. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study. *IEEE Transactions on Software Engineering* 41, 4 (2015), 331–357.

[62] Yangyang Zhao, Yibiao Yang, Hongmin Lu, Jinping Liu, Hareton Leung, Yansong Wu, Yuming Zhou, and Baowen Xu. 2017. Understanding the value of considering client usage context in package cohesion for fault-proneness prediction. *Automated Software Engineering* 24, 2 (2017), 393–453.

[63] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. 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 (FSE)*. ACM, 91–100.