Training data selection for cross-project defection prediction: which approach is better?

Yi Bin*, Kai Zhou†, Hongmin Lu*, Yuming Zhou*‡, and Baowen Xu*‡
*State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing, China
†School of Software, Southeast University, Nanjing, China
‡Corresponding author: {zhouyuming, bwxu}@nju.edu.cn

***Abstract*—Background: Many relevancy filters have been proposed to select training data for building cross-project defect prediction (CPDP) models. However, up to now, there is no consensus about which relevancy filter is better for CPDP. Goal: In this paper, we conduct a thorough experiment to compare nine relevancy filters proposed in the recent literature. Method: Based on 33 publicly available data sets, we compare not only the retaining ratio of the original training data and the overlapping degree among the retained data but also the prediction performance of the resulting CPDP models under the ranking and classification scenarios. Results: In terms of retaining ratio and overlapping degree, there are important differences among these filters. According to the defect prediction performance, global filter always stays in the first level. Conclusions: For practitioners, it appears that there is no need to filter source project data, as this may lead to better defect prediction results.**

Keywords- Defect prediction; cross-project; filter; model

I. INTRODUCTION

Recent years have seen an increasing interest of building supervised models to predict defects in software projects [1, 3, 4, 6, 26]. For a given software project, this process usually requires the historical data to train the model. However, sufficient historical data may be unavailable. This is especially true for a new software startup company or a new type of software systems. One way to deal with the training data shortage is to use the historical data from other projects (i.e. source projects) to build the prediction model and apply it to the current project (i.e. the target project). Currently, there are many publicly available data repositories such as PROMISE [9], providing an opportunity to develop cross-project defect prediction (CPDP) models.

Given a large amount of source project data available, for practitioners, one important challenge is how to find the right training data to build accurate CPDP models for the target project. In [15], Turhan et al. reported that the CPDP model would have a low prediction performance if it were built using all the available source project data. However, after applying nearest neighbor (NN) filtering to the source project data, they found that the prediction performance of the resulting CPDP model had been improved considerably. This indicates that using source project data relevant to the target project help achieve accurate CPDP models. Inspired by Turhan et al. study, in the last decade, many new relevance filters were proposed to find the right training data for building CPDP models [10, 12, 13, 14, 15, 16, 32].

These relevancy filters provide various approaches to selecting training data to help build quality CPDP models. However, for practitioners, when facing with these training data selection approaches, the following question naturally arises: which approach should be used in practice? To answer this question, several studies have been conducted to compare their effectiveness in CPDP models [14, 16, 31, 32]. However, until now, there is no consistent conclusion on how these filters affect the effectiveness of the resulting CPDP models in defect prediction.

To attack the above problem, in this study, we conduct a thorough experiment to compare nine main training data filters proposed in the literature. These filters includes global filter (GF) [12], local filter (LF) [12], Neighbor filter (NF) [10], Burak filter (BF) [15], Peters filter (PF) [14], Kawata filter (KF) [16], He filter (HF) [32], HeBurak filter (HBF) [32], and HePeters filter (HPF) [32]. These filters use different rules to select the training data to build the CPDP models. Of these filters, GF using all available external project data, LF uses local clusters to guide the selection, NF uses neighbor clusters to guide the selection, BF uses testing instances to guide the selection, PF uses training instances to guide the selection, and KF uses an advanced clustering algorithm called DBSCAN to conduct the selection. In addition, HF uses the data distribution to guide the selection, HBF uses HF and BF to guide the selection, and HPF uses HF and PF to guide the selection. These nine filters are the representative filters proposed in the last decade for selecting training data in the context of CPDP.

We make the following contributions in this paper. First, we investigate the differences between nine relevancy filters in the context of CPDP. In contrast, most previous studies only compare two or three relevancy filters. Second, we compare these filters from the following three dimensions: the retaining ratio of the original training data, the overlapping degree among the retained data, and the prediction performance of the resulting CPDP models. This enables practitioners to have a comprehensive understanding on the advantages and disadvantages of different relevancy filters. Third, we evaluate the prediction performance of the resulting CPDP models by relevancy filters under both the classification scenario and the ranking scenario. In particular, under each scenario, the employed evaluation indicators include the commonly used non-effort-aware indicators as well as the newly introduced effort-aware indicators. In summary, our study provides an in-depth understanding the differences between representative relevancy filters as well

978-1-5090-4039-1/17 $31.00 © 2017 IEEE
DOI 10.1109/ESEM.2017.49

as practical guidelines for choosing appropriate filters to build effective CPDP models in practice.

The rest of this paper is organized as follows. Section II introduces the research background. Section III describes the investigated relevancy filters for CPDP. Section IV describes the research methodology. Section V describes the experimental setup. Section VI reports the experimental results in detail. Section VII compares our work to related work. Section VIII analyzes the threats to the validity. Section IX concludes the paper and outlines directions for future research.

II. BACKGROUND

In this section, we describe the background on cross-project defect prediction and relevancy filters for training data selection.

A. Cross-project defect prediction

For a given target project, cross-project defect prediction (CPDP) denotes using historical data from other projects (i.e. source projects) as the training data to build models to predict defects in the target project. To the best of our knowledge, Briand et al.'s work is the first CPDP study, which explores the applicability of CPDP on object-oriented software projects [33]. Their result showed that the CPDP model produced a poor classification performance but had an acceptable ranking performance on the target project. This indicates that, if used appropriately, cross-project defect prediction would be helpful for practitioners.

In recent years, CPDP has attracted a rapid increasing research interest. Existing CPDP studies cover a variety of wide-ranging topics, including validating CPDP on different development environments, different programming languages, and different module granularity [5, 7, 8, 11, 14, 15, 17]. Up to now, the overall results show that CPDP has a promising prediction performance, comparable to or even better than WPDP (within-project defect prediction). These studies significantly advance our knowledge about the practical value of cross-project defect prediction. Currently, it appears that CPDP has become an active research field in defect prediction community.

B. Training set data selection

In CPDP, one main challenge is that source and target project data usually exhibit significantly different distributions [15]. This violates the similar distribution assumption of the training and test data required by most modeling techniques. Consequently, if we use all the source project data as the training data, it is very likely that the resulting CPDP model will have a poor defect prediction performance on the target project [15].

To mitigate this problem, it has been argued that source project data irrelevant to the target project should be filtered out before building the CPDP model. In other words, only the source project data relevant to the target project should be retained for model building. Based on this intuition, in [15], Turhan et al. proposed the first relevancy filter (called "Burak filter" in [14]) and demonstrated its effectiveness. After Turhan et al.'s pioneer work, a number of relevancy filters were proposed for selecting the right training data for a target project [10, 12, 14, 15, 16, 32]. These relevancy filters differ from each other, either in different granularity (project level or instance level) or in different rationale (training data driven or testing data driven).

III. TRAINING DATA SELECTION APPROACHES

In this section, we describe nine main filters investigated in our study. Let $\{s_1, s_2, \ldots, s_n\}$ be the raw external training data set (i.e. the source project data) and $\{t_1, t_2, \ldots, t_m\}$ be the test data set (i.e. the target project data). Here, s_i is the ith instance in the training data ($1 \le i \le n$) and t_j is the jth instance in the test data ($1 \le j \le m$). For the simplicity of presentation, in the following, s_i will be called a training instance and t_j will be called a testing instance. The purpose of a filter is to filter out irrelevant training data from the external training data $\{s_1, s_2, \ldots, s_n\}$ for the target test data $\{t_1, t_2, \ldots, t_m\}$. Consequently, a part of data from $\{s_1, s_2, \ldots, s_n\}$ will be retained for building a CPDP model.

A. Global filter (GF): Using all available external data

This filter is described by Menzies et al. in [10]. GF uses all the external project data as the training data to build the CPDP model. In other words, GF indeed does not filter out any source project data. During this process, only one CPDP model will be built.

B. Local filter (LF): Local cluster guided selection

This filter is described by Bettenburg et al. in [12]. LF applies cluster labels to filter out irrelevant training data. First, combine all source project metric data (i.e. independent variables) in the training data set and the metric data in testing data set to obtain a combined data set. Then, use a clustering algorithm called MCLUST [25] to partition the combined data set to different clusters: C1, C2, …, Ck. After that, for each cluster, use the training instances in this cluster to build a defect prediction model and apply the model to the testing instances in the same cluster. During this process, k CPDP models will be built in total.

C. Neighbor filter (NF): Neighbor cluster guided selection

This filter is described by Menzies et al. in [10]. In nature, NF is similar to LF. First, combine all source project metric data in the training data set and the metric data in the testing data set to obtain a combined data set. Then, use the clustering algorithm called MCLUST to get the optimal number k of clusters. After that, use k-means to partition the combined data set to k different clusters: $C_1, C_2, \ldots, C_k$. For each cluster, use the training instances in this cluster to build a defect prediction model. Consequently, obtain k defect prediction models. Finally, for the testing instances in each cluster, apply the model from the nearest cluster to conduct the test. Therefore, k CPDP models will be built in total.

D. Burak filter (BF): Testing instance guided selection

This filter is described by Turhan et al. in [15]. BF applies testing instances to guide the filtering of the training data. First, for each testing instance in the testing data, use the Euclidean distance to find 10 neighbors in the source

project training data set. Then, combine these neighbors (without duplication) to form a new training data set. After that, use the new training data to build a defect prediction model and apply the model to the testing data set. During this process, only one CPDP model will be built.

E. Peters filter (PF): Training instance guided selection

This filter is described by Peters et al. in [14]. PF applies training instances to guide the filtering of the training data. First, combine the training data set and the testing data set to obtain a combined data set. Then, use the clustering algorithm k-means to the combined data set to obtain different clusters: C_1, C_2, …, C_k. After that, retain clusters that contain at least one testing instance. Next, for each training instance in each of retained clusters, find the nearest testing instance in the same cluster. After obtaining these testing instances, for each testing instance, use the Euclidean distance find the nearest neighbors from the source project train data in the same cluster. These neighbors (without duplication) will be used to form a new training set. Note that, class labels are not considered in the above process. Finally, use the new training data to build a defect prediction model and apply the model to the testing data set. During this process, only one CPDP model will be built.

F. Kawata filter (KF): Density-based spatial clustering guided selection

This filter is described by Kawata et al. in [16]. KF applies density-based spatial clustering to guide the filtering of the training data. KF is the filter based on DBSCAN. DBSCAN (Density-Based Spatial Clustering) is one of the advanced clustering algorithms. It can remove noisy instance based on the density of instances. Therefore, it is fit for being a filter. First, combine the training data set and the testing data set to a combined data set. Then, find sub-clusters by using DBSCAN. After that, select sub-clusters that have at least one instance of the target project data. Next, collect the instances of cross-project training data in the selected sub-clusters into a new cross-project training data. Finally, use the resulting new training data to build a defect prediction model and apply the model to the test data set. During this process, only one CPDP model will be built.

G. He filter (HF): Distribution characteristic guided selection

This filter is described by He et al. in [32]. HF applies distribution characters of every source project data to guide the filtering of the training data. Compared with previous filters at instance level, it is a filter at file level. First, for each source project, construct $TDS = \{F_1, F_2, \ldots F_n\}$. Here, F_i represent the ith feature vector which consist of all instances' values of feature i. More specifically, for each feature vector F_i, the distribution characteristic vector $V = \{Min(F_i), Max(F_i), Median(F_i), Mean(F_i), St.\ D(F_i)\}$. Therefore, for each source project data set, there is a corresponding distribution characteristic vector $TV = \{Min(F_1), Max(F_1), Median(F_1), Mean(F_1), STD(F_1), \ldots Min(F_n), Max(F_n), Median(F_n), Mean(F_n), STD(F_n)\}$. Then, for the testing data set T, apply KNN to obtain K nearest $TDSs$ neighbors. After that, combine these $TDSs$ to obtain the retained training data. Finally, use the resulting training data to build a defect prediction model and apply the model to the test data. During this process, only one CPDP model will be built.

TABLE I. THE INDEPENDENT VARIABLES

Metric	Description
LOC	Lines of code
WMC	Weighted methods complexity
DIT	Depth of Inheritance tree
NOC	Number of Children
CBO	Coupling between objects
RFC	Response for a class
DAM	Data Access Metric
LCOM, LCOM3	Lack of cohesion in methods
Ca	Afferent Coupling
Ce	Efferent Coupling
NPM	Number of Public Methods
MOA	Measure of Aggregation
MFA	Measure of Functional Abstraction
CAM	Cohesion among methods of class
IC	Information Coupling
CBM	Coupling between methods
AMC	Average method complexity
AVG_CC	Average Cyclomatic complexity
Max_CC	Maximum Cyclomatic complexity

H. HeBurak filter (HBF): Distribution characteristic and testing instance guided selection

This filter is described by He et al. in [32]. HBF applies both distribution characteristic and testing instance to guide the selection. This filter is indeed a multi-granularity filter. First, apply He filter to obtain the retained train data TR. Then, apply Burak filter to TR to obtain the further retained training data. Finally, use the resulting training data to build a defect prediction model and apply the model to the test data. During this process, only one CPDP model will be built.

I. HePeters filter (HBF): Distribution characteristic and training instance guided selection

This filter is described by He et al. in [32]. HBF applies both distribution characteristic and training instance to guide the selection. First, apply He filter to source project data to obtain the retained train data TR. Then, apply Peters filter to TR to obtain the further retained training data. Finally, use the resulting training data to build a defect prediction model and apply the model to the test data set. During this process, only one CPDP model will be built.

IV. RESEARCH METHODOLOGY

In this section, we first introduce the independent and dependent variables. Then, we describe the modeling technique used. After that, we describe the research questions under study. Next, we describe the performance indicators. Finally, we report the data analysis method.

A. Independent and dependent variables

In this study, we use 20 object-oriented metrics [18, 19-21] as the independent variables. Compared with McCabe and Halstead metrics, these metrics were designed specifically for object-oriented (OO) software. Many

previous works also use these metrics as the independent variables [14-16]. Table I shows the descriptions of these OO metrics.

The dependent variable in this study is a Boolean variable. It has one of the two possible values: 1 indicating defective and 0 indicating not defective.

B. Modeling techniques

In this study, we use random forest (RF) to build the CPDP model. In previous work, RF is reported as a stable and robust classifier to build supervised defect prediction models [2, 14, 17, 30]. Meanwhile, previous studies highlight that RF can lead to high prediction accuracy [24]. Therefore, many practitioners intend to use RF to build defect prediction models. For all involved techniques, including the clustering techniques used in the filters, we use the default parameters unless particularly specified.

C. Research questions

We aim to conduct a comprehensive comparison of the nine relevancy filters for CCDP described in Section III. First, for each filter, we want to know the corresponding training data retaining ratio, i.e. the ability to reduce source project data. The higher the training data retaining ratio is, the less the original training data will be filtered out. Second, for each pair of filters, we want to know the corresponding retained data overlap degree, i.e. to the extent that the retained data by one filter is overlapped with those by another filter. The higher the retained data overlap degree, the higher possibility a pair of filters will lead to the CPDP models with the similar defect prediction performance. Third, for these nine filters, we want to know which leads to more effective CPDP. We will take into account two typical CPDP scenarios: classification and ranking [22]. In the classification scenario, modules are first classified into two categories according to their predicted defect-proneness: defective and not defective. After that, these modules classified as defective are targeted for quality enhancement. In the ranking scenario, modules are ranked by their predicted defect-proneness. A defect-proneness ranking lists modules from the most to the least defect-prone. Software managers would find such a ranking very helpful since they can simply select from the ranking list as many potential defective modules to test as available resources will allow. We want to find out which filter will lead to better defect prediction performance in the ranking and classification scenarios. More specifically, we set out to investigate the following research questions:

- RQ1 (Retaining ratio): What is the retaining ratio of each relevancy filter in our data set?
- RQ2 (Overlapping degree): How are the retained training data by these relevancy filters overlapped?
- RQ3 (Prediction effectiveness): Which relevancy filters lead to better prediction effectiveness in CPDP?

D. Performance Evaluation

We next introduce the performance indicators used for evaluating relevancy filters, including retaining ratio, overlapping degree, and classification/ranking performance indicators.

1) Retaining ratio

As described in section III, some filters such as BF are testing instance driven. Therefore, the number of retained training data is relevant to the target project. In other words, the retaining ratio is based on the target project. For a given filter *f*, we use the following steps to compute the corresponding training data retaining ratio. First, for each testing instance t_i in the target project *k*, compute $InstRetain(t_i)$ as follows:

$$InstRetain(f, t_i) = \frac{numTrain(f, t_i)}{allTrain} \quad (1)$$

Here, $numTrain(f, t_i)$ is the number of retained training instances used for training the CPDP model that is applied to ti, while *allTrain* is the total number of original training instances.

If there are n testing instances in project *k* (assume that they are $t_1, t_2 \ldots, t_n$), then the retaining ratio of *f* for the target project *k* is defined as:

$$RetainingRatio(f, k) = \frac{\sum_{i=1}^{n} InstRetain(f, t_i)}{n} \quad (2)$$

2) Overlapping degree

For a pair of filters f_1 and f_2, we use the following steps to compute their overlapping degree between the retained training data. Similar to retaining ratio, the overlapping degree is also relevant to the target project. First, for each testing instance t_i in the target project *k*, compute $InstOverlappingDegree(f_1, f_2, t_i)$ as follows:

$$InstOverlapDegree(f_1, f_2, t_i) = \frac{OverlapNumTrain(f_1, f_2, t_i)}{NumTrain(f_1)} \quad (3)$$

Here, $OverlapNumTrain(f_1, f_2, t_i)$ is the number of overlapped instances between f_1 retained training data and f_2 retained training data, while $NumberTrain(f_1)$ is the number of f_1 retained training data.

If there are *n* testing instances in project *k* (assume that they are $t_1, t_2 \ldots, t_n$), then the overlap degree between f_1 and f_2 for the target project *k* is defined as:

$$\text{OverlappingDegree}(f_1, f_2, k) = \frac{\sum_{i=1}^{n} InstOverlapDegree(f_1, f_2, t_i)}{n} \quad (4)$$

As can be seen, *OverlappingDegree* is non-symmetrical. Indeed, it reveals the percentage of f_1 retained training data that is overlapped with f_2 retained training data.

3) Classification performance indicators

a) Non-effort-aware classification performance

We use AUC (Area under ROC) to evaluate the non-effort-aware classification effectiveness of a defect prediction model [23]. Compared with traditional effectiveness measures, AUC exhibits a number of desirable properties: (1) it does not depend on the choice of the decision threshold; (2) it is invariant to the prior probabilities of positive and negative modules; and (3) it can be interpreted as the probability that a randomly chosen defective module is (correctly) rated or ranked with greater suspicion than a randomly chosen not defective module.

b) Effort-aware classification performance

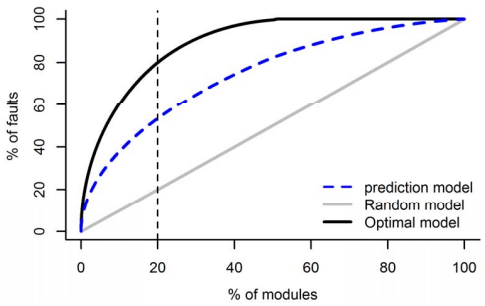


Figure 1. Alberg graph

We use ER-AVG (proposed in [2]) to evaluate the effort-aware classification effectiveness of a defect prediction model. For a given model *model*, the ER (effort reduction, called "LIR" in [27]) is defined as:

$$ER(model) = \frac{Effort(random) - Effort(model)}{Effort(random)} \quad (5)$$

Here, *Effor(model)* is the ratio of the total SLOC (source lines of code) in these predicted high-risk modules to the total SLOC in the target project and *Effor(random)* is the proportion of SLOC to test or inspect to the total SLOC in the target project that a random selection model needs to achieve the same recall of defects as the prediction model m. ER-AVG is the average effort reduction of the model over all possible thresholds. Therefore, similar to AUC, ER-AVG is independent from specific thresholds.

4) Ranking performance indicators

a) Non-effort-aware ranking performance

We use FPA (fault-percentile-average, proposed in [23]) to evaluate non-effort-aware ranking effectiveness of a defect prediction model. Assume that a project with K modules m_1, m_2..., m_K, listed it in no-decreasing order of predicted numbers of defects. Therefore, m_K is the module predicted to have the most defects. Let n_k be the actual number of defects in module k, and let $N = n_1 + ... + n_k$ be the total number of actual defects in the target project. In our study, N is indeed the total number of defective modules. For any m in 1, ..., K , the proportion of actual defects in the top m predicted modules is calculated as:

$$P_m = \frac{1}{N} \sum_{k=K-m+1}^{K} n_k \quad (6)$$

FPA is defined as the average of the P_m:

$$FPA = \frac{1}{K} \sum_{m=1}^{K} \frac{1}{N} \sum_{k=K-m+1}^{K} n_k \quad (7)$$

b) Effort-aware ranking performance

We use CE (cost-effectiveness, proposed in [26]) to the effort-aware ranking effectiveness of a defect prediction model. More specifically, CE is defined as:

$$CE(model) = \frac{Area(model) - Area(random)}{Area(optimal) - Area(random)} \quad (8)$$

Here, *Area(model)* is the area under the curve of model *model* in the Alberg diagram (shown in Fig. 1). In this diagram, the x-axis represents the cumulative percentage of SLOC of the modules selected from the module ranking and the y-axis is the cumulative percentage of defects found in the selected modules. *Area(random)* is the area under the curve of the random model (modules are randomly selected to inspect/test), while *Area(optimal)* is the area under the curve of the optimal model (modules are sorted in decreasing order according to their actual defect densities).

E. Data analysis method

We use Cliff's δ to examine whether the magnitude of the difference between retaining ratio/prediction performance of two models are practically important [28]. By convention, the magnitude of the difference is considered either trivial ($|\delta| < 0.147$), small (0.147-0.33), moderate (0.33-0.474), or large (> 0.474) [28]. In addition, we use the Scott-Knott test [29] to group the prediction models to examine whether there exist some models outperform others. In particular, we performed a double Scott-Knott test [33] that ensures that our analysis recognizes techniques that perform well across projects. At the first run, the Scott-Knott test is run over each individual target project. For each target project, we generate 60 bootstrapping samples as the test data [34]. The nine relevancy filters are respectively applied to the training data (all the other project data except the target project data) to build the CPDP models to predict defects in these test data. The Scott-Knott test is used to find the statistical distinct ranks of these filtering techniques within each project. At the second run, for each filtering technique, we have 33 different Scott-Knott ranks (i.e., one from each project), which we input into another Scott-Knott test to find the final statistically distinct ranks of techniques. The Scott-Knott test uses a hierarchical cluster analysis method to divide the prediction models into two groups according to the mean performance. If the difference between the divided groups is statistically significant, the Scott-Knott test will recursively divide each group into two different groups. The test will terminate when groups can no longer divided into statistically distinct groups.

V. Experimental Setup

In this section, we describe the data sets and the subject projects used in this study.

A. Data sets

We use 33 data sets downloaded from the PROMISE repository (http://openscience.us/repo) to conduct the experiment. Table II provides a description for these data sets, including the project name, the release no, the number of modules, the total lines of code, and the percentage of defective modules. These data sets have been extensively used in previous defect prediction studies. More importantly, these data sets are publicly available. This will facilitate other researchers to replicate our results. In our study, we choose each data set as the testing data and the other data sets are combined as the raw training data. After that, we will apply each filter to the raw training data to obtain the retained training data and then use them to build the corresponding cross-project defect prediction model.

TABLE II. THE DATA SETS USED

Project name	Release	#modules	Total SLOC	%defective modules
Ant	1.7	745	208653	22.416%
Camel	1.6	965	113055	19.585%
Ivy	2.0	352	87769	11.364%
Lucene	2.4	340	102859	59.706%
Poi	3.0	442	129327	63.575%
Synapse	1.2	256	53500	33.594%
Velocity	1.6	229	57012	34.061%
Xalan	2.6	885	411737	46.441%
Xerces	1.4	588	141180	74.320%
ArcPlatform	1.0	235	31342	11.634%
Berek	1.0	43	32320	37.209%
Ckjm	1.8	10	1469	50.000%
E-learning	1.0	64	3639	7.813%
Forrest	0.8	32	6540	6.250%
Intercafe	1.0	27	11322	14.815%
JEdit	4.3	492	202363	2.236%
KalkulatorDiety	1.0	27	4022	22.222%
log4j	1.2	205	38191	92.195%
nieruchomosci	1.0	27	4754	37.037%
Pbeans	2.0	51	15125	19.608%
PdfTranslator	1.0	33	6318	45.455%
Prop	6.0	660	97570	10.152%
Redaktor	1.0	176	59280	15.341%
Serapion	1.0	45	10505	20.000%
Skarbonka	1.0	45	15029	20.000%
SklepAGD	1.0	20	9602	60.000%
SystemData	1.0	65	15441	13.846%
SzybkaFucha	1.0	25	1910	56.000%
TermoProject	1.0	42	8239	30.952%
Tomcat	6.0	858	300674	8.974%
WorkFlow	1.0	39	4125	51.282%
WspomaganiePI	1.0	18	5685	66.667%
Zuzel	1.0	29	14421	44.828%

B. *Subject projects*

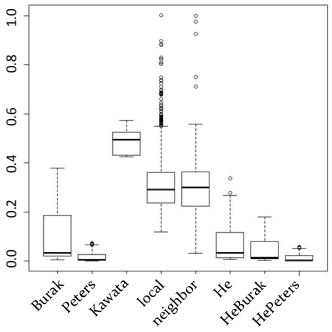
These projects cover both open-source and closed-source projects. All of them were developed using Java. The functions implemented in these projects are wide-ranging, including build management system, text editor, and text processing library. In most projects, the defect data show a highly imbalanced distribution. Fig. 2 presents the distribution of defects with regard to module size under all projects. Note that, all modules in each project have been sorted by SLOC in ascending order, the x-axis is the project name and the y-axis lists the ratio of cumulative modules to total modules. Each value in the cell represents the total number of defects on the corresponding modules.

VI. EXPERIMENTAL RESULTS

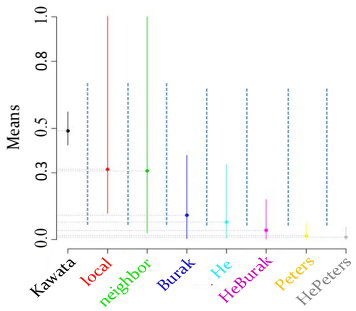
In this section, we report the results of our experiment and answer RQ1, RQ2 and RQ3 by our analysis.

A. *RQ1: Retaining ratio*

Fig. 2 presents the result of retaining ratio between these filters. From Fig. 2(a), on the one hand, we can see that Kawata filter, local filter, and neighbor filter have the highest retaining ratio. Indeed, all the three filters are cluster-based selection approaches. For these filters, the number of retained training instances is related to the distribution



(a) The distribution of retaining ratio



(b) The double Scott-Knott test

Figure 2. The results of retaining ratio

characteristic and spatial characteristic of data set. On the other hand, we can see that Peters filter and HePeters filter have the lowest retaining ratio. According to the algorithms of Peters filter, Burak filter, HePeters filter and HeBurak filter, the number of retained training instances should be similar to the number of testing instances. Burak filter and HeBurak filter are testing data guided selection approaches. For each testing instance, at most one training instance will be retained. Peters filter and HePeters filter are training data guided selection approaches. First, the testing data are filtered by the training data. Then, for each retained testing instance, at most one training instance will be retained. Therefore, if the number of training instances is much larger than the number of testing instances, Peters filter, HePeters filter, Burak filter, and HeBurak filter should have a low retained ratio.

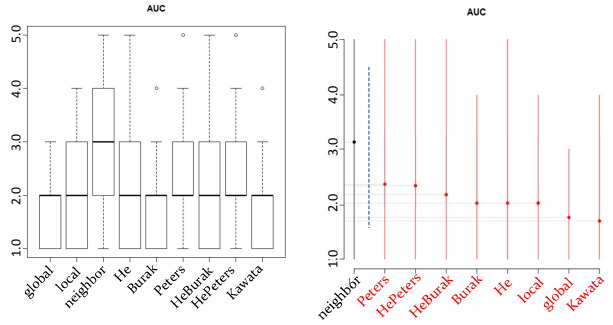
As shown in Fig. 2(b), the double Scott-Knott test shows that all the filters are significantly different with each other. This result is similar with the result of the values of Cliff's δ. According to Cliff's δ, except for the difference in retained ratio between local filter and neighbor filter is trivial (0.057). In the other cases, the Cliff's δ is moderate or large. Overall, these results indicate that most filters differ significantly in terms of the ability to filter out source project data.

B. *RQ2: Overlapping degree*

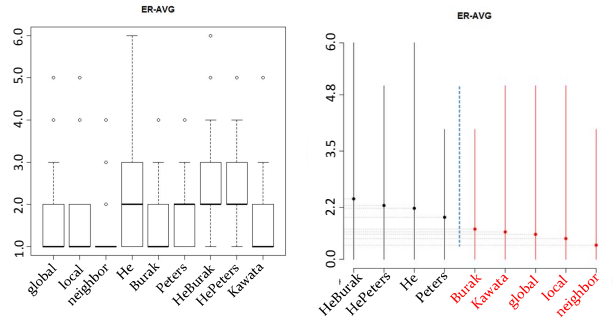
Table III reports the mean and standard deviation of the overlapping degree among the retained training data produced by different filters. When looking the columns in Table III, we can observe that: (1) most filters have a high

TABLE III. OVERLAAPING DEGREE

	Peters	Burak	Kawata	global	Local	neighbor	He	HeBurak	HePeters
Peters		0.295±0.142	0.539±0.128	1±0	0.337±0.115	0.330±0.153	0.088±0.096	0.067±0.072	0.077±0.081
Burak	0.046±0.029		0.604±0.097	1±0	0.337±0.113	0.329±0.150	0.087±0.091	0.085±0.090	0.023±0.022
Kawata	0.021±0.024	0.136±0.135		1±0	0.343±0.110	0.334±0.150	0.075±0.076	0.049±0.056	0.013±0.015
global	0.029±0.022	0.114±0.120	0.469±0.041		0.336±0.110	0.328±0.150	0.078±0.089	0.045±0.055	0.012±0.016
local	0.019±0.022	0.114±0.119	0.481±0.044	1±0		0.281±0.152	0.078±0.087	0.044±0.053	0.013±0.016
neighbor	0.019±0.023	0.114±0.119	0.480±0.044	1±0	0.298±0.130		0.077±0.087	0.043±0.053	0.012±0.016
He	0.022±0.025	0.129±0.126	0.495±0.063	1±0	0.337±0.116	0.327±0.153		0.618±0.231	0.202±0.149
HeBurak	0.027±0.029	0.196±0.170	0.566±0.087	1±0	0.338±0.117	0.327±0.154	1±0		0.233±0.144
HePeters	0.100±0.094	0.201±0.162	0.507±0.123	1±0	0.336±0.122	0.326±0.157	1±0	0.744±0.182	



(a) Non-effort-aware performance (AUC)



(b) Effort-aware performance (ER-AVG)

Figure 3. The results in classification performance

overlapping degree with Kawata filter, local filter, and neighbor filter; and (2) most filters have a low overlapping degree with Peters filter, He filter, HeBurak filter, and HePeters filters. These results are consistent with the results observed in retaining ratio. Combining the results from retained ratio and overlapping degree for different filters, we conclude that these investigated filters differ with each other significantly. Therefore, different filters may lead to considerably different CPDP models.

C. *RQ3: Prediction effectiveness*

a) Non-effort-aware classification performance

Fig. 3(a) and Table IV present the results of AUC in the classification scenario. Fig. 3(a) reports the result of AUC for nine filters: the left sub graph uses box-plot to describe for each filter the distribution of the ranks on individual projects (i.e. the ranks produced by the first level Scott-Knott test); the right sub graph reports the results of the second level Scott-Knott test.

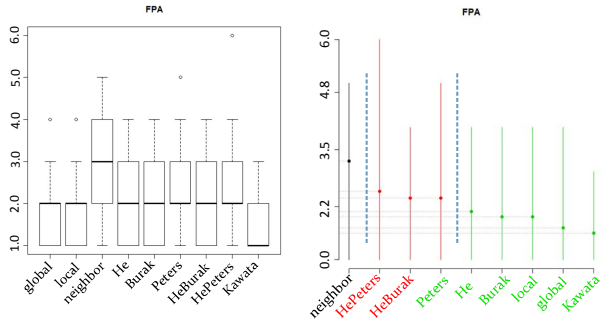
As can be seen, the double Scott-Knott test shows that all the filters are separated to two groups. Kawata filter has a higher mean rank of AUC than the other filters and only neighbor filter have practical important difference between these filters (its mean of rank is 3.154). The result of Cliff's δ is similar (Note that, in Table IV, the gray background indicates that the corresponding δ is significantly different from zero). The magnitude of the difference between neighbor filter and other filters are almost moderate. However, the magnitudes of the difference between other filters are almost small and trivial. The above results reveal that, in terms of AUC, except for neighbor filter, we can see that the filters that have a high retaining ratio tend to have a better rank.

b) Effort-aware classification performance

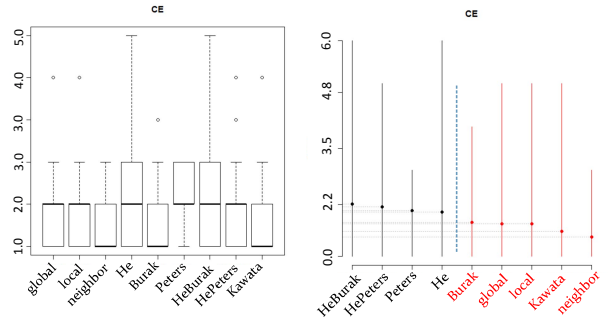
Fig. 3(b) and Table IV present the results of ER-AVG in the classification scenario. The Scott-Knott test separates all the filters to two groups. He filter, HeBurak filter, and HePeters filter have the worse rank. Global filter, Kawata filter, local filer, Burak filter, and neighbor filter have a better rank. Table V confirms that He filter, HeBurak filter, and HePeters filter perform worse. In particular, the magnitude of the difference between HeBurak filter, HePeters filter, and neighbor filter is small. In contrast, the magnitudes of the difference between other filters are all trivial. The above results reveal that, in terms of ER-AVG, the filters that have a high retaining ratio perform better.

c) Non-effort-aware ranking performance

Fig. 4(a) and Table V show the results of FPA in the ranking scenario. As can be seen, in Fig. 4(a), there are three groups separated by the double Scott-Knott test. Kawata filter have the best rank value (mean of rank is 1.333) and neighbor filter is in the worst rank group (mean of rank is 3.245). The similar grouping results can be seen in Table V. Except for neighbor, we can observe that the magnitude of the difference between high retaining ratio filters (i.e. global filter, Kawat filter and local filter) is trivial. In contrast, the results of low retaining ratio filters (i.e. He filter, HeBurak filter, HePeters filter, Burak filter, and Peters filter) are similar. In particular, the magnitudes of the difference between neighbor filter and the other high retaining ratio



(a) Non-effort-aware performance (FPA)



(b) Effort-aware performance (CE)

Figure 4. The results in ranking performance

filters are moderate ($0.333 \leq |\delta| \leq 0.421$). However, the magnitudes of the difference between neighbor filter and low retaining ratio filters like HePeters filter are small ($0.185 \leq |\delta| \leq 0.264$). Based on the above results, we conclude that, in terms of FPA, global filter stay in the first level of rank group and high retaining ratio filters except for neighbor filter perform better than low retaining ratio filters.

d) Effort-aware ranking performance

Fig. 4(b) and Table V report the results of CE in the ranking scenario. Table V show that the magnitudes of the difference between these nine filters are all trivial. From Fig. 4(b), we can observe that nine filters are separated to two groups by the double Scott-Knott test. Higher retaining ratio filters, including global filter, local filter, neighbor filter, and Kawata filter outperform the other filters. HeBurak filter (mean of rank is 2.206), which have the lowest retaining ratio, have the worst performance. Therefore, we conclude that, in terms of CE, high retaining ratio filters perform better than low retaining ratio filters.

In both classification and ranking performance, we can see that global filter always stay in the first level of rank group. Except for neighbor filter in AUC and FPA, the high retaining ratio filters including global filter, local filter, and Kawata filter perform better than the low retaining ratio filters such as Peters filter. We know that the higher retaining ratio is, the more useful information will be retained from the original training data. Therefore, we can conclude that, in CPDP, using all the available source project data as the training data will lead to better prediction results.

VII. Comparison With Related Works

Table VI compares our study with previous studies that investigate the effectiveness of relevancy filters in CPDP. The second column reports the clustering techniques used in when filtering the training data. The third column reports the modeling techniques used to build the CPDP models. The fourth column reports the investigated filters. The fifth and sixth columns summary the result obtained in each study, including the performance indicators used and the winner. In particular, ↑ means that, for the corresponding performance

TABLE IV. Cliff's δ for AUC and ER-AVG

Cliff's δ		ER-AVG								
		Burak	**Peters**	**Kawata**	**Global**	**Local**	**Neighbor**	**He**	**HeBurak**	**HePeters**
AUC	**Burak**		0.047	–0.042	–0.036	-0.016	–0.061	0.085	0.130	0.107
	Peters	0.041		–0.086	–0.080	–0.060	–0.104	0.035	0.081	0.057
	Kawata	–0.131	–0.177		0.007	0.027	–0.023	0.116	0.158	0.140
	Global	–0.099	–0.144	0.027		0.020	–0.031	0.112	0.153	0.134
	Local	–0.042	–0.084	0.086	0.057		–0.049	0.094	0.136	0.116
	Neighbor	0.285	0.249	0.408	0.371	0.320		0.129	0.170	0.152
	He	-0.004	-0.044	0.131	0.096	0.039	-0.288		0.049	0.021
	HeBurak	0.032	-0.006	0.171	0.134	0.077	-0.258	0.037		-0.028
	HePeters	0.067	0.030	0.196	0.163	0.109	-0.207	0.069	0.034	

TABLE V. Cliff's δ for FPA and CE

Cliff's δ		**CE**								
		Burak	**Peters**	**Kawata**	**Global**	**Local**	**Neighbor**	**He**	**HeBurak**	**HePeters**
FPA	**Burak**		0.066	-0.008	0.013	0.020	-0.008	0.070	0.121	0.083
	Peters	0.065		-0.074	-0.052	-0.046	-0.057	0.004	0.058	0.018
	Kawata	-0.108	-0.167		0.023	0.030	0.017	0.079	0.128	0.089
	Global	-0.083	-0.142	0.022		0.007	-0.006	0.059	0.110	0.071
	Local	-0.030	-0.092	0.073	0.051		–0.012	0.053	0.105	0.065
	Neighbor	0.333	0.264	0.421	0.394	0.350		0.062	0.114	0.074
	He	0.052	-0.010	0.149	0.129	0.081	-0.259		0.057	0.018
	HeBurak	0.117	0.054	0.213	0.191	0.145	-0.193	0.062		-0.039
	HePeters	0.114	0.053	0.205	0.185	0.140	-0.185	0.058	-0.001	

TABLE VI. RELATED WORD ABOUT FILTERS OF DEFECT PREDICTION

Study	Cluster technique	Modeling technique	Filter	Result		Statistical test	Effect size
				Indicator	Winner		
Turhan et al.[15]	KNN	NB	Burak	Proposed Burak filter; do not compare with others			
Peters et al.[14]	K-means, KNN	RF, LR, NB, KNN	Burak, Peters	g-measure ↑	Peters	No	No
He et al. [32]	KNN	J48,LR,NB, RF,SVM	HE, Burak, HeBurak, HePeters,	precision ↑	HeBurak, HePeters	Yes	No
				f-measure ↑	LR,NB,RF: HePeters SVM: He J48: HeBurak		
				g-measure ↑	LR,NB:HePeters J48,SVM,RF: Burak		
Kawata et al.[16]	DBSCAN, KNN	LR, RF, NB, KNN	Peters, Burak, Kawata, Global	AUC ↑	Kawata	No	No
				g-measure ↑	Kawata		
				f-measure ↑	LR,KNN: Kawata NB,RF: Peters		
Herbold et al.[31]	EM, WHERE, KNN	SVM	Global, Local, Burak	AUC ↑	Local	Yes	Yes
				recall ↑ , error ↓	inconclusive		
				Precision, f-measure ↑	similar		
Our study	MCLUST K-means DBSCAN	RF	All the 9 filters	AUC ↑ , FPA ↑	High retaining ratio filter	Yes	Yes
				CE ↑ , ER-AVG ↑			

indicator, the larger the value is, the better the prediction performance is. The last two columns reports whether statistical tests are applied and whether the effect sizes are examined. As can be seen, compared with previous work, our study investigates the effectiveness of nine relevancy filters in CPDP under both the classification and ranking scenarios. In particular, under each scenario, our study employs not only non-effort-aware indicators but also effort-aware indicators to evaluate the prediction performance of the CPDP models.

VIII. THREATS TO THE VALIDITY

A. Threats to internal validity

The first threat is the unknown effect of the techniques used to build the CPDP models. To mitigate this threat, we used Naïve Bayes and logistic regression, another two popular machine learning techniques, to build the CPDP models. After that, we rerun the analysis. We found that our finding remained no change, regardless of which modeling technique was considered.

The second threat is from the indicators used to evaluate the performance in defect prediction for the CPDP models. Our study used FPA and CE to evaluate the overall ranking performance. In practice, however, practitioners may be more interested in the performance of the top-ranked modules (for example, the top 20% modules). In order to determine the influence of cut-off values on our conclusion, we computed FPA and CE only for the top 10%, top 20%, and top 30% modules. We found that, our finding on the nine filters largely remained the same, regardless of which cut-off value was taken into account.

The third threat is from the clustering methods used in the investigated filters. As can be seen, different filters use different clustering methods. Consequently, it is hard to know whether the difference in the results is due to the different filtering mechanisms or the different clustering methods. This problem needs to be investigated in the future.

B. Threats to external validity

The most important threat is that our finding may not be generalized to other projects. Our study uses more than 30 projects, which are across a wide range of project types and scales, as the target projects. The experimental results drawn from these target projects are quite consistent. Furthermore, the data sets collected from these projects are large enough to draw statistically meaningful conclusions. Nonetheless, we do not claim that our finding can be generalized to all projects, as the subject target projects under study might not be representative of projects in general. To mitigate this threat, there is a need to replicate our study across a wider variety of projects in the future.

IX. CONCLUSION AND FUTURE WORK

The motivation of our work is to investigate which training data selection approaches for CPDP is better in practice. To this end, we use 9 main filters proposed in recent literature to select training data and build CPDP models. In particular, we compare not only their training data retaining ratio and the overlapping degree of the retained training data but also the defect prediction performance of these CPDP models under the classification and ranking scenarios. In order to obtain a comprehensive evaluation, under each scenario, we use both non-effort-aware and effort-aware performance indicators. Based on 33 publicly available data sets, the experimental results show that, in terms of retaining ratio and overlapping degree, there are important differences among these filters. In particular, we find that, according to the defect prediction performance, global filter always stays in the first level. Considering the time cost of applying the filter algorithms, we recommend for practitioners there is no need to filter source project data, as this may lead to better defect prediction results.

In the future, we will use more projects to compare the usefulness of these filters. The purpose is to examine if our finding can be generalized to the other projects.

ACKNOWLEDGMENT

This work is partially supported by the National Key Basic Research and Development Program of China (2014CB340702), the National Natural Science Foundation of China (61772259, 61432001, 61472178), and the National Natural Science Foundation of Jiangsu Province (BK20130014).

REFERENCES

[1] N. Nagappan. Mining metrics to predict component failures. ICSE 2006: 452-461.

[2] Y. Zhou, B. Xu, H. Leung, L. Chen. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM Transactions on Software Engineering & Methodology, 23(1), 2014: 1-51.

[3] C. Catal, B. Diri. A systematic review of software fault prediction studies. Expert Systems with Applications, 36(4), 2009: 7346-7354.

[4] Z. He, F. Peters, T. Menzies, Y. Yang. Learning from open-source projects: An empirical study on defect prediction. ESEM 2013:45-54.

[5] Z. He, F. Shu, Y. Yang, M. Li, Q. Wang. An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering ,19(2), 2012: 167-199.

[6] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell. A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 38(6), 2012: 1276-1304.

[7] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy. Cross-project defect prediction:a large scale experiment on data vs. domain vs. process. FSE 2009:91-100.

[8] Y. Ma, G. Luo, X. Zeng, A. Chen. Transfer learning for cross-company software defect prediction. Information & Software Technology, 54(3), 2012: 248–256.

[9] T. Menzies, M. Rees-Jones, R. Krishna, C. Pape. The Promise Repository of Empirical Software Engineering Data; http://openscience.us/repo. North Carolina State University, Department of Computer Science , 2015.

[10] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan. Local versus global lessons for defect prediction and effort estimation. IEEE Transactions on Software Engineering, 39(6), 2013: 822-834.

[11] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, D. Cok. Local vs. global models for effort estimation and defect prediction. ASE 2011:343-351.

[12] N. Bettenburg, M. Nagappan, A.E. Hassan. Think locally, act globally:improving defect and effort prediction models. MSR 2012: 60-69.

[13] N. Bettenburg, M. Nagappan, A.E. Hassan. Towards improving statistical modeling of software engineering data: think locally, act globally!. Empirical Software Engineering, 20(2), 2015: 294-335.

[14] F. Peters, T. Menzies, A. Marcus. Better cross company defect prediction. MSR 2013: 409-418.

[15] B. Turhan, T. Menzies, A.B. Bener, J.D. Stefano. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 14(5) , 2009: 540-578.

[16] K. Kawata, S. Amasaki, T. Yokogawa. Improving relevancy filter methods for cross-project defect prediction. ACIT 2015: 2-7.

[17] A. Kaur, K. Kaur. Value and applicability of academic projects defect datasets in cross-project software defect prediction. CIN 2016: 154-159.

[18] V.R. Basili, L.C. Briand, W.L. Melo. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering, 22(10), 1996: 751-761.

[19] S.R. Chidamber, C.F. Kemerer. A metrics suite for object oriented design. OOPLSA 1991: 197-211.

[20] F. Fioravanti, P. Nesi. A study on fault-proneness detection of object-oriented systems. CSMR 2001: 121-130.

[21] Y. Zhou, B. Xu, H. Leung. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. Journal of Systems & Software, 83(4), 2010: 660-674.

[22] Y. Yang, Y. Zhou, H. Lu, L. Chen, Z. Chen, B. Xu, H. Leung, Z. Zhang. 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.

[23] E.J. Weyuker, T.J. Ostrand, R.M. Bell. Comparing the effectiveness of several modeling methods for fault prediction. Empirical Software Engineering, 15(3), 2010: 277-295.

[24] T.K. Ho. Random decision forests. ICDAR 1995: 278.

[25] C. Fraley, A.E. Rafter. MCLUST Version 3 for R: Normal mixture modeling and model-based clustering. Department of Statistics University of Washington, 2006.

[26] E. Arisholm, L.C. Briand, E.B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems & Software, 83(1), 2010: 2-17.

[27] Y. Shin, A. Meneely, L. Williams, J.A. Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Transactions on Software Engineering, 37(6), 2011: 772-787.

[28] J. Romano, J.D. Kromrey. Appropriate statistics for ordinal level data : Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys? FAIR 2006.

[29] N. Mittas, L. Angelis. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. IEEE Transactions on Software Engineering, 39(4), 2013: 537-551.

[30] A. Kaur, K. Kaur. An empirical study of robustness and stability of machine learning classifiers in software defect prediction. Advances in Intelligent Informatics 2015: 383-397.

[31] S. Herbold, A. Trautsch, J. Grabowski. Global vs. local models for cross-project defect prediction. Empirical Software Engineering, 2017, to appear.

[32] P. He, B. Li, D. Zhang, Y. Ma. Simplification of training data for cross-project defect prediction. arXiv:1405.0773, 2014.

[33] L.C. Briand, W.L. Melo, J. Wüst. Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Transactoins on Software Engineering, 28(7), 2002: 706-720.

[34] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, K. Matsumoto. An empirical comparison of model validation techniques for defect prediction models. IEEE Transactions on Software Engineering, 43(1), 2017: 1-18.