Information and Software Technology 55 (2013) 1101–1118

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

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

Empirical evaluation of the effects of mixed project data on learning defect predictors

Burak Turhan [a,*], Ayşe Tosun Mısırlı [a], Ayşe Bener [b]

[a] Dept. of Information Processing Science, University of Oulu, 90014 Oulu, Finland
[b] Ted Rogers School of ITM, Ryerson University, Toronto, ON, Canada M5B 2K3

ARTICLE INFO

Article history:
Available online 29 October 2012

Keywords:
Cross project
Within project
Mixed project
Defect prediction
Fault prediction
Product metrics

ABSTRACT

Context: Defect prediction research mostly focus on optimizing the performance of models that are constructed for isolated projects (i.e. within project (WP)) through retrospective analyses. On the other hand, recent studies try to utilize data across projects (i.e. cross project (CP)) for building defect prediction models for new projects. There are no cases where the combination of within and cross (i.e. mixed) project data are used together.

Objective: Our goal is to investigate the merits of using mixed project data for binary defect prediction. Specifically, we want to check whether it is feasible, in terms of defect detection performance, to use data from other projects for the cases (i) when there is an existing within project history and (ii) when there are limited within project data.

Method: We use data from 73 versions of 41 projects that are publicly available. We simulate the two above-mentioned cases, and compare the performances of naive Bayes classifiers by using within project data vs. mixed project data.

Results: For the first case, we find that the performance of mixed project predictors significantly improves over full within project predictors (p-value < 0.001), however the effect size is small (*Hedges'* $g = 0.25$). For the second case, we found that mixed project predictors are comparable to full within project predictors, using only 10% of available within project data (p-value = 0.002, $g = 0.17$).

Conclusion: We conclude that the extra effort associated with collecting data from other projects is not feasible in terms of practical performance improvement when there is already an established within project defect predictor using full project history. However, when there is limited project history, e.g. early phases of development, mixed project predictions are justifiable as they perform as good as full within project models.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

Defect predictors are decision support systems for prioritizing the list of software modules to be tested, in order to allocate limited testing resources effectively, and to detect as many defects as possible with minimum effort. Other than guiding quality assurance (QA) activities by pointing out the primary locations to test, defect predictors can also be used for different purposes, such as planning and management of quality strategies, and monitoring the effectiveness of the planned QA activities [1,2]. In this paper, we set the scope of defect prediction to the case where models are constructed with the goal of identifying the most defect-prone parts of the code through binary classification.

Defect prediction studies usually formulate the problem as a supervised learning problem, where the outcome of a defect predictor model depends on historical data. Expected use of such models in practice is to train and calibrate them with past project data and then to apply them to new projects. Though there are many publications on the problem, almost all ignore the practical aspect that the purpose of a defect predictor is to identify the defects of *new projects*, which are different than those used in model construction, and majority of publications focus on the algorithmic aspects and report simulation results of defect predictors that are trained on a specific project and tested on the reserved portion of the same project (e.g. refer to the list in the systematic review by Hall et al. [3]). While this approach aims at validating the effectiveness of these models, it does not address the practical purposes. Though there are studies that apply defect predictors to the consecutive versions of the same project, they are longitudinal case studies and do not address predictions across different projects, e.g. [4,5].

* Corresponding author. Tel.: +358 401676636.
E-mail addresses: burak.turhan@oulu.fi (B. Turhan), ayse.tosunmisirli@oulu.fi (A. Tosun Mısırlı), ayse.bener@ryerson.ca (A. Bener).

0950-5849/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2012.10.003

We are curios about why defect prediction research fail to utilize data across projects. Is it because such an approach is useless in defect prediction context? We are optimistic about the answer. Just consider the problem of cost estimation, which is technically similar to defect prediction, i.e. a supervised learning problem utilizing past data.[1] Though the effectiveness of resulting models may vary, cost estimation research have made use of cross project data for a long time. A systematic review comparing within company vs. cross company cost estimation models concluded that some companies may benefit from cross company cost estimations, while others may not [6]. Data gathered from different projects are extensively used in cost estimation studies, i.e. COCOMO models and ISBSG dataset [7,8]. However, we should note that the abstraction levels for cost estimation and defect prediction are different, i.e. project level and module (e.g. method, class, file) level. While a project corresponds to a single data point for cost estimation, the same level of abstraction provides many data points for defect prediction. Nevertheless, it is the idea of using data from other projects, which turned out to be applicable at least to some extent in cost estimation, that is of interest for this paper.

Our optimism not only relies on the analogy with cost estimation, but also on the recent research results in cross-project defect prediction studies (please see Section 2). Another motivation for pursuing the research on cross project data for defect prediction is that successful applications will have significant implications in practice. Companies will be able to employ defect prediction techniques in their projects, even if they have no or limited historical local data to build models with. Another scenario is that companies may already have their defect prediction models in place and making use of external data may improve the performance of models learned from local project data. However, there are no studies addressing the latter case, i.e. the effects of incorporating cross project data in existing within project defect predictors, which we address in this paper.

Our contribution is to investigate the merits of mixed-project predictions and exploring if (and when) they may be effective, and we have identified the following research questions for this purpose:

1. *RQ1: How effective is using data from other projects in order to improve the performance of existing within project defect predictors?*
2. *RQ2: How much within project data should be enriched with data from other projects to achieve comparable performance with full within project data predictions?*

Detailed descriptions and rationale of our research questions are provided in Section 3. This paper is an extension of our previous work published elsewhere [9]. Specifically, the results presented in Section 4 have partially appeared in [9] for 10 datasets. In this paper, we increase the number of datasets and provide additional research questions, experiments and associated results. This paper is organized as follows: in the next section we provide a review of existing literature on cross-project defect prediction. Section 3 describes the methodology including research questions, hypotheses, data, methods, the setup we used in our experiments, and the threats to validity. We present our results in Section 4 before we conclude our work in Section 5.

2. Related work

In this section, we provide a discussion of cross-project prediction studies, which is an emerging area with very limited number of published studies, based on selected studies representing major research effort on the topic. We have identified nine empirical studies [9–17]. We focus our discussions on these studies rather than providing a general review of defect prediction literature, which is out of the scope of this paper (we refer the reader to [3] for a systematic review of defect prediction studies in general). Detailed descriptions and discussions on these studies are given below.

To the best of our knowledge, the earliest work on cross-project prediction is by Briand et al. [10]. They use logistic regression and MARS models to learn a defect predictor from an open-source project (i.e. Xpose), and apply the same model to another open-source project (Jwriter), which is developed by an identical team with different design strategies and coding standards. They observe that cross-project prediction is indeed better than a random and a simple, class-size based model. Yet, cross-project performance of the model was lower compared to its performance on the training project. They argue that cross-project predictions can be more effective in more homogeneous settings, adding that such an environment may not exist in real life. They identify the challenge as of high practical interest, and not straightforward to solve.

Turhan et al. make a thorough analysis of cross project prediction using 10 projects collected from two different data sources (i.e. a subset of the projects analyzed in this paper) [11]. They identify clear patterns that cross project predictions dramatically increase the probability of detecting defective modules (from median value of 75% to 97%), but the false alarm rates as well (from median value of 29% to 64%). They claim that improvements in detection rates are due to extra information captured from cross project data and the increased false alarms can be explained by the irrelevancies in the data. They propose a nearest-neighbor based data selection technique to filter the irrelevancies in cross project data and achieve performances that are close to, but still worse than within project predictions. They conclude that within company predictions is the best path to follow and cross project predictions with data filtering can be used as a stop-gap technique before a local repository is constructed. Turhan et al.'s findings are confirmed in a replication study by Nelson et al. [15].

Zimmermann et al. consider different factors that may affect the results of cross project predictions [12]. They categorize projects according to their domain, process characteristics and code measures. In their initial case study they run experiments to predict defects in Internet Explorer (IE) using models trained with Mozilla Firefox and vice versa. These products are in the same domain and have similar features, but development teams employ different processes. Their results show that Firefox can predict the defects in IE successfully (i.e. 76.47% precision and 81.25% recall), however the opposite direction does not seem to provide useful outcomes (i.e. 4.12% recall). Zimmermann et al. then collect data from 10 additional projects and perform 622 pairwise predictions across project components. This is a slightly different approach than Turhan et al.'s, who constructed predictors from a common pool of cross project data with data filtering in order to satisfy Briand et al.'s homogeneity argument. Zimmermann et al. classify a prediction as successful if precision, recall and accuracy values are all above 75%, which results in only 21 successful predictions corresponding to 3.4% success rate. They do not mention the performance of predictions that are below the threshold. They derive a decision tree using these prediction results to estimate expected performance from a cross project predictor in order to guide practitioners. An interesting pattern in their predictions is that open-source projects are good predictors of close-source projects, however open-source projects cannot be predicted by any other projects.

In a following study, Turhan et al. investigate whether the patterns in their previous work [11] are also observed in open-source

[1] We should note that there are also studies using *unsupervised techniques* both for cost estimation and defect prediction, however this is not the point behind the motivation explained here.

software and analyzed three additional projects [14]. Similar to Zimmermann et al. they find that the patterns they observed earlier are not easily detectable in predicting open-source software defects using proprietary cross project data.

Cruz and Ochimizu train a defect prediction model with an open-source project (Mylyn) and test the performance of the same model on six other projects [13]. Before training and testing the model, they try to obtain similar distributions in training and test samples through data transformations (i.e. power transformation). They also remove outliers in data by trimming the tails of distributions. They observe that using transformed training and test data yields better cross-project prediction performances [13].

Jureczko and Madeyski look for clusters of similar projects in a pool of 92 versions from 38 proprietary, open-source and academic projects [16]. Their idea is to reuse same defect predictor model among the projects that fall in the same cluster. They use a comprehensive set of code metrics to represent the projects and compare the performances of prediction models that are trained on the same project vs. other projects in the same cluster. They identify three statistically significant clusters (out of 10), where cross project predictions are better than within project predictions in terms of the number of classes that must be visited to detect 80% of the defects.

Liu et al. conduct similar experiments to [11], differing in the data selection approach [17]. They employ a search-based strategy, using genetic algorithms, to select data points from seven NASA MDP projects in order to build cross-project defect prediction models. They use 17 different machine learning methods and majority voting to build defect predictors. They consistently observe lower misclassification errors than trivial cross-project models (i.e. using all available cross-project data without data selection). They argue that single project data may fail to represent the overall quality trends, and recommend development organizations to combine multiple project repositories using their approach for exploiting the capabilities of their defect prediction models [17]. However, they do not provide a comparison with baseline within project defect predictors.

3. Methodology

In this section, we describe: the rationale to our research questions, the data used in the experiments, the analyses and methods along with the experimental design.

3.1. Research questions and hypotheses

1. *RQ1: How effective is using data from other projects in order to improve the performance of existing within project defect predictors?*
   We want to check whether using additional data from other projects improve the performance of an existing, project specific defect prediction model in terms of defect detection accuracy. In order to check the performance of mixed project approach, we simulate the cases where the majority of WP data is available (i.e. 90% in our simulations) for single version projects and also use multi-version software for predictions across releases, by incrementally adding CP data. We should note that gathering CP data brings an extra burden to data collection activities, therefore mixed-project approach would be feasible in practice only if there is significant (both statistical and practical) performance increase. The hypothesis under evaluation for this research question is: *Mixed project (WP + CP) defect prediction improves the performance* (*measured* by balance value[2]) *of within (WP) project defect prediction*. Formally, we define and evaluate $H1$ as:

$$H1_0 : median(bal(WP + CP)) \leqslant median(bal(WP)) \vee g \leqslant 0.50$$
$$H1_A : median(bal(WP + CP)) > median(bal(WP)) \wedge g \geqslant 0.50$$

   where g is the corrected Hedges' g to evaluate the effect size [18]. Please note that the first term in the quantified hypothesis corresponds to the evaluation of improvement in balance in terms of statistical significance, whereas the second term corresponds to the evaluation of practical significance in terms of effect size measured by Hedges' g, i.e. at least a medium effect size.
2. *RQ2: How much within project data are needed to be enriched with data from other projects to achieve comparable performance with full within project data predictions?*
   While RQ1 deals with the cases where there are already a significant amount of WP data, RQ2 investigates whether mixed project predictions can achieve comparable performance with full WP models when there is only limited amount of WP data, i.e. early phases in development. A comparable prediction performance in early phases of development against retrospective analysis would justify the extra effort associated with gathering CP data. In order to simulate such cases, we use only $(10\%, 20\%, \ldots, 80\%)$ of the available WP data together with CP data in order to check the corresponding hypothesis:
   Mixed project defect prediction using only a small portion of WP data (WP(k%) + CP) performs (measured by balance value) better (or same) than full within (WP) project prediction. Formally, we define and evaluate H2 as:

$$H2_0 : \forall k \in [10 : 10 : 80] : median(bal(WP(k\%) + CP)) < median(bal(WP))$$
$$H2_A : \exists k \in [10 : 10 : 80] : median(bal(WP(k\%) + CP)) \geqslant median(bal(WP))$$

Please note that we are not including effect size as a part of $H2$, since we are looking for (at least) a similar performance between the two approaches.

3.2. Data to build and validate the models

We use data collected by three different research groups, which are publicly available in PROMISE repository [19]. Data from systems that include the label "NASA" in their names come from NASA aerospace projects and these were collected as part of the metric data programme (MDP). On the other hand, systems with the label "SOFTLAB" are from a Turkish software company developing embedded controllers for home appliances and the related data were collected by the authors for an earlier work [14]. The projects in these two groups are all single version and the metrics along with the defect information are available at the functional method level. The details of data collection for NASA projects is available at the MDP website,[3] and data for SOFTLAB projects were collected following the same metric definitions. Table 1 provides summary information for these projects. One caveat of cross project data utilization is that all projects need to have the same set of metrics in order to be able to pool data from different projects. Therefore, though the projects in Table 1 have more available metrics, we are limited to use only those that are common in all analyzed projects. The final set of 16 metrics include the following:

- complexity and control flow [20]
  - cyclomatic complexity

[2] Defined in Section 3.5.

[3] http://mdp.ivv.nasa.gov.

Table 1
Projects with functional method level metrics, sorted by "# methods".

System	Type	Language	# Versions	# Methods	% Defect
NASA-pc1	Proprietary	C++	1	1109	6.94
NASA-kc1	Proprietary	C++	1	845	15.45
NASA-kc2	Proprietary	C++	1	522	20.49
NASA-cm1	Proprietary	C++	1	498	9.83
NASA-kc3	Proprietary	Java	1	458	9.38
NASA-mw1	Proprietary	C++	1	403	7.69
SOFTLAB-ar4	Proprietary	C	1	107	18.69
SOFTLAB-ar3	Proprietary	C	1	63	12.70
NASA-mc2	Proprietary	C++	1	61	32.29
SOFTLAB-ar5	Proprietary	C	1	36	22.22

  - design complexity
  - branch count
- size
  - LOC total
  - LOC code and comment
  - LOC comments
  - LOC executable
- Halstead metrics [21]
  - total/unique operators
  - total/unique operands
  - volume
  - difficulty
  - effort
  - error
  - programming time

The third group of data used in this paper contains measurements at the class level for either single or multiple version projects, and these were collected by Jureczko and Spinellis [22]. There are 63 samples from 31 projects (six proprietary projects developed by the same company, 46 releases of 14 open-source projects, and 11 student projects) in this group. The details of the project data are available in [22], nevertheless we provide a summary here.

Jureczko and Spinellis report that the student projects were developed by senior undergraduates of computer science, who worked in groups of 3–6 for a duration of 1 year, using a strictly iterative development process with high level of test coverage. Since their size is small, in our experiments, we have also used a merged version of these student projects named as *student*. Further, the proprietary projects are custom built solutions for customers in insurance domain, all developed by the same company. Finally, the multi-version project data were collected from open-source (Apache) projects [22].

The set of available metrics for all projects are as follows:

- weighted methods per class (WMC)
- depth of inheritance tree (DIT)
- number of children (NOC)
- coupling between objects (CBO)
- response for a class (RFC)
- lack of cohesion in methods (LCOM) [23]; LCOM3 [24]
- number of public methods (NPM)
- data access metric (DAC)
- measure of aggregation (MOA)
- measure of functional abstraction (MFA)
- cohesion among methods of class (CAM) [25]
- inheritance coupling (IC)
- coupling between methods (CBM)
- average method complexity (AMC) [26]
- afferent couplings (Ca)
- efferent couplings (Ce) [27]
- maximum and average McCabe's cyclomatic complexity (Max(CC) and Avg(CC)) [20]
- lines of (binary) code (LOC).

These class level code metrics were collected by Jureczko and Spinellis using the open source ckjm tool.[4] Then they used another tool named BugInfo[5] to mine the versioning repository logs, matching each commit message against information from bug tracking systems based on the regular expression representations of guidelines for each project, in order to classify each commit as a bug fix (or not) [22]. In this study, we use the bug count information as a binary variable (i.e. a class is defective or defect-free) in our classification framework. Table 2 provides a list of all projects with context information.

Please note that there are many possible groupings of the available data in different abstraction levels. Possible grouping variables are the (i) type of available metrics (functional method level, class level), (ii) type of projects (proprietary, open-source, student), and (iii) version information (multi-version, single-version). We chose to use the first grouping while presenting the data descriptions in this section. Also, please note that none of the functional method level projects are multi-version, and all multi-version projects are open-source systems. Since this paper is an extension of our previous work, where we used only functional method level data, we use only the class level data for extending our previous analyses and investigating the new research questions.

3.3. Methods to build the models

Before using the data, we applied log-transformation (i.e. replaced all numeric values with their logarithms) as recommended by previous studies [28,11] to satisfy the normality assumption of Naive Bayes classifier that we used as our underlying defect prediction model. There are many justifications for this choice. For instance, Menzies et al. demonstrated the effectiveness of this technique in a series of data mining experiments on NASA datasets [28,29]. Further, Lessmann et al. compared commonly used classification techniques on the same datasets and found no significant difference between the performances of top 15 classifiers, including Naive Bayes. They concluded that the choice of the classifier is not that important for building defect predictors [30]. Last but not the least, Naive Bayes has been listed among the top 10 algorithms in data mining by the corresponding community [31]. Given the evidence from other modules and the prior probability of the *defective* class, a software module is classified as *defective*, if its posterior probability of being defective is the higher than the alternative, i.e. defect free.

In our experiments, we use cross project data after applying the filtering method proposed in [11], i.e. nearest-neighbor (NN)-filtering, due to its simplicity and documented effectiveness. With this filtering, it is expected to obtain a subset of available cross project data that shows similar characteristics to that of test project's data.[6] Note that the granularity of data is at the class level, hence NN-filtering identifies the classes with similar characteristics based on their Euclidean distances in the available metric space[7] (i.e. does not make project-level comparisons for similarity). In order to

[4] http://www.spinellis.gr/sw/ckjm.
[5] http://kenai.com/projects/buginfo.
[6] During the selection of additional samples from cross project data with NN-filtering, only the portion that is reserved for testing (from within/local project) is used to calculate the similarities, not the whole available data (from within/local project). Otherwise, the simulation would not reflect a real setting for practical application, nor it would allow a fair comparison with within project prediction.
[7] Excluding the target variable of defect information.

Table 2
Projects with class level metrics, used in this study. Size and defect rate columns show the average size and defect rates across releases for multi-version software. A project is multi-version if the value of # versions column is greater than one. (Grouped by project type and then sorted by "system" name.)

System	Type	Language	# Versions	(Avg) # classes	(Avg) % defect
arc	Student	Java	1	234	11.54
berek	Student	Java	1	43	37.21
nieruchomosci	Student	Java	1	27	37.04
pdftranslator	Student	Java	1	33	45.45
redaktor	Student	Java	1	176	15.34
serapion	Student	Java	1	45	20.00
skarbonka	Student	Java	1	45	20.00
sklebagd	Student	Java	1	20	60.00
termoproject	Student	Java	1	42	30.95
workflow	Student	Java	1	39	51.28
wspomaganiepi	Student	Java	1	18	66.67
prop-1	Proprietary	Java	1	18,471	14.82
prop-2	Proprietary	Java	1	23,014	10.56
prop-3	Proprietary	Java	1	10,274	11.49
prop-4	Proprietary	Java	1	8718	9.64
prop-5	Proprietary	Java	1	8516	15.25
prop-6	Proprietary	Java	1	660	10.00
tomcat	Open-source	Java	1	858	8.97
ant	Open-source	Java	5	338	19.58
camel	Open-source	Java	4	696	18.87
forrest	Open-source	Java	3	22	13.39
ivy	Open-source	Java	3	235	24.92
jedit	Open-source	Java	5	350	19.65
log4j	Open-source	Java	3	150	50.44
lucene	Open-source	Java	3	261	54.89
pbeans	Open-source	Java	2	39	48.27
poi	Open-source	Java	4	345	49.82
synapse	Open-source	Java	3	212	23.60
velocity	Open-source	Java	3	213	58.47
xalan	Open-source	Java	4	830	52.16
xerces	Open-source	Java	3	494	35.23

implement the NN-filter, we first calculate the pairwise distances between the reserved test set and the *candidate* training set samples (i.e. all cross project data). Let N be the test set size. For each test instance, we pick its $k = 10$ nearest neighbors from the candidate training set. Then we come up with a total of $10 \times N$ similar instances. Note that these $10 \times N$ instances may not be unique (i.e. a single data point can be the nearest neighbor for many data points in the test set). Using only unique ones, we form the final cross project training set and use it in our experiments. We should note that this is a special case of analogy based learning and no class information is used in the process; hence there is no violation of experiment design principles.

3.4. Model validation procedure

We provide the pseudocode for the simulation experiments in: (a) Fig. 1 for comparing full within project (WP) and mixed project (WP + CP) defect predictors in order to answer *RQ1* and (b) Fig. 2 for investigating how much WP data is enough to form mixed project (WP(k%) + CP) defect predictors comparable to full within project (WP) predictors, in order to answer *RQ2*.

We conduct our experiments for RQ1 on two groups of datasets (DATA1 in line 1, and DATA2 in line 2 of Fig. 1) individually since each set shares different set of metrics. Specifically, DATA1 corresponds to the projects with functional method level metrics, whereas DATA2 corresponds to the projects with class level metrics. Fig. 1 reads as follows:

- We identify common class level code metrics among DATA2 projects; a pre-processing that is required for using data from other projects.
- Our independent variable is the type of the training sets used for learning defect predictors, i.e. *WP_MODEL* vs. (*WP + CP*)_*MODEL* in the figure, and our dependent variable is the performance (pd, pf, bal) of the resulting defect predictors.
- Between lines 9–12, we prepare the training and the test sets for the multi-version projects.
  - We reserve a single version of each project as the test set (*TEST* at line 11).
  - We use the remaining versions of the project as the (initial) training set (WP + CP) for WP model (*WPTrain* at line 10).
- If the project has a single version, we apply 90–10% division rule on *data* to form the training (*WPTrain*) and the test (*TEST*) sets (lines 15–19).
- We use all other projects with all available versions to form the CP training set (*CPTrain* at line 7).
- We form the training set of the mixed model, *WP + CP*, in three steps (lines 26–36).
  - *Step 1:* Between lines 26–29, *CPTrain* is filtered by selecting top 10 nearest neighbors for each test instance in *TEST*.
  - *Step 2:* Line 32 performs a cleaning for duplicate instances selected from *CPTrain*.
  - *Step 3:* Line 35 iterates on the dataset *NNCPTrain* in order to find the minimum amount of CP data necessary to build the mixed model ((*WP + CP*)_*MODEL*) with maximum performance. This iteration is done to select only a subset of other projects' data to avoid the dominance of CP in the mixed model.
    - WPTrain is included into the training set of the mixed model, *WP + CP*, in line 36.
    - Line 37 trains naive Bayes learner with *WP + CP* data and forms the mixed model ((*WP + CP*)_*MODEL*).
    - To find the best subset of CP data in the mixed model, we record the performance for each j (Line 39).
    - We select the best performing mixed model among different sizes (j) of CP data (Line 43).
- Lines 45–46 record the performance of WP model on the same test set.

```
1: DATA1 = {pc1,kc1,kc2,cm1,kc3,mw1,mc2,ar3,ar4,ar5}
2: DATA2 = {arc, berek,nieruchomosci,pdftranslator, redaktor, serapion, skarbonka, sklebagd, systemdata, szybkafucha, termoproject, workflow, wspomaganiepi, zuzel, tomcat, student, prop 1, prop 2, prop 3, prop 4, prop 5, prop 6, ant,camel,ivy, jedit, log4j, lucene, pbeans, poi, synapse, velocity, xalan, xerces}
3: LEARNER = {Naive Bayes}
4: {DATA is DATA1 or DATA2}
5: for data ∈ DATA do
6:     {Use all versions of other datasets as CP training set}
7:     CPTrain = DATA - data
8:
9:     if data has multiple versions then
10:        WPTrain = Add other versions of data
11:        TEST = data
12:    end if
13:
14:    for i = 1 → 30 do
15:        if data is a single version project then
16:            {Shuffle data in each iteration}
17:            WPTrain = Select 90% of data
18:            TEST = data - WPTrain
19:        else
20:            {Use previously formed WPTrain and TEST}
21:        end if
22:        {Apply log filtering on WPTrain, CPTrain, TEST}
23:
24:        {Form the training set (WP+CP) of the mixed model}
25:        {Step 1:NN filtering: Select 10 nearest neighbours in CPTrain for each test instance}
26:        for test ∈ TEST do
27:            dist = NNAlgorithm(test,CPTrain)
28:            NNCP ← Select 10 instances in CPTrain with min(dist)
29:        end for
30:
31:        {Step 2:Remove duplicate CP instances}
32:        NNCPTrain = SelectUnique(NNCP)
33:
34:        {Step 3:Incrementally add CP data from NNCPTrain into WP+CP}
35:        for j = 10 : size(NNCPTrain) do
36:            WP+CP = WPTrain + Select random j instances in NNCPTrain
37:            (WP + CP)_MODEL = Train LEARNER with (WP+CP)
38:            {Report the performance for each j}
39:            [pd(j), pf(j), bal(j)] = Apply (WP + CP)_MODEL on TEST
40:        end for
41:
42:        {Select the best performance among j (different sizes) on NNCPTrain}
43:        [wpcp_pd(i), wpcp_pf(i), wpcp_bal(i)] ← Select max(bal) on TEST
44:
45:        WP_MODEL = Train LEARNER with WPTrain
46:        [wp_pd(i), wp_pf(i), wp_bal(i)] = Apply WP_MODEL on TEST
47:    end for
48: end for
```

Fig. 1. Pseudo-code for the experimental setup of RQ1.

- We randomized the overall process 30 times ($i = 30$) to collect the performance statistics on: the probability of detection, the probability of false alarm, and balance, and to apply significance tests between the within project (WP) and the mixed (WP + CP) models. We also shuffled the data (as indicated in line 16) in each iteration, prior to forming training and test sets for single-version projects, in order to avoid sampling bias.

In order to address RQ2, we use a smaller ratio (from 10% to 80%) of within project data against which we use 90% of within project data, to observe whether mixed project predictions are practically useful in case of limited within project data. However, in this experiment setup, we did not use one version of project X as test and other versions as training data while conducting experiments on multi-version projects.[8] This experimental setup is presented in Fig. 2:

- All projects' first versions are divided into training and test sets using 90–10% division rule (Lines 9–10).
- In addition to Steps 1–3 in Fig. 1, we add an intermediate step (Line 23 as Step 3) before sampling from CP data (Line 27 as Step 4). In Step 3, a loop iteratively selects *k*% of within project data from WPTrain (lines 24–25).[9]
- Line 36 stores the performance of the mixed model ((*WP*(*k*%) + *CP*)*_MODEL*) for each *k* and *i*, i.e., when *k*% of WP data is added to training set (WPSample + CP) at iteration *i*. In total, there are 30 performance measures stored for each *k* in the mixed model in order to conduct significance tests for RQ2.

3.5. Evaluation

We use three performance measures to assess the classification (i.e. defective and defect-free) performance of the defect

[8] The reason for this design choice is that our goal is to investigate the phenomenon under the conditions where only limited within project data are available. In multi-version projects, all versions after the first release has the opportunity to use the previous releases' data. Hence a simulation using all versions is of no practical interest, and we limit our simulation with the first available version only.

[9] For simplicity in the presentation of the setup, *k* is listed to vary from 10% to 80%. Indeed, since the simulated full WP model uses 90% of the available data, 10% corresponds to 10% × 90% = 9% of all available data; 20% corresponds to 18% and so on.

```
DATA = {arc, berek,nieruchomosci,pdftranslator, redaktor, serapion, skarbonka, sklebagd, systemdata, szybka-
fucha, termoproject, workflow, wspomaganiepi, zuzel, tomcat, student, prop 1, prop 2, prop 3, prop 4, prop 5,
prop 6, ant,camel,ivy, jedit, log4j, lucene, pbeans, poi, synapse, velocity, xalan, xerces}
LEARNER = {Naive Bayes}
for data ∈ DATA do
    {Use all versions of other datasets as CP training set}
    CPTrain = DATA - data

    for i = 1 → 30 do
        {Shuffle data in each iteration}
        WPTrain = Select 90% of data
        TEST = data - WPTrain
        {Apply log filtering on WPTrain, CPTrain, TEST}

        {Form the training set (WPSample+CP) of the mixed model}
        {Step 1:NN filtering: Select 10 nearest neighbours in CPTrain for each test instance}
        for test ∈ TEST do
            dist = NNAlgorithm(test,CPTrain)
            NNCP ← Select 10 instances in CPTrain with min(dist)
        end for

        {Step 2:Remove duplicate CP instances}
        NNCPTrain = SelectUnique(NNCP)

        {Step 3:Incrementally add WP data from WPTrain into WPSample+CP}
        for k = 10 → 80 do
            WPSample = Select random k% from WPTrain

            {Step 4:Incrementally add CP data from NNCPTrain into WPSample+CP}
            for j = 10 : size(NNCPTrain) do
                WPSample+CP = WPSample + Select random j instances in NNCPTrain
                (WP(k%) + CP)_MODEL = Train LEARNER with (WPSample+CP)
                {Report the performance for each j}
                [pd(j), pf(j), bal(j)] = Apply (WP(k%) + CP)_MODEL on TEST
            end for

            {Store the best performance among j (different sizes) on NNCPTrain}
            [wpcp_pd(i,k), wpcp_pf(i,k), wpcp_bal(i,k)] ← Select max(bal) on TEST
            k ← k + 10
        end for

        WP_MODEL = Train LEARNER with WPTrain
        [wp_pd(i), wp_pf(i), wp_bal(i)] = Apply WP_MODEL on TEST
    end for
end for
```

Fig. 2. Pseudo-code for the experimental setup of RQ2.

predictors: *probability of detection, probability of false alarm, balance*. Since our datasets are unbalanced in terms of defects, following the recommendations in [32,28], we avoided the use of measures such as *accuracy* and *precision*. Using a confusion matrix, we count the number of true positives (tp), true negatives (tn), false positives (fp), false negatives (fn) and derive the performance measures described below [28].

Probability of detection (pd) is a measure of accuracy for correctly identifying defective classes. It should be as high as possible (ideal case is when pd = 1):

$$\text{pd} = \text{tp}/(\text{tp} + \text{fn}) \tag{1}$$

Probability of false alarm (pf) is a measure for false alarms and it is an error measure for incorrectly flagging the defect free classes. False alarms causes testing efforts to be spent in vain. Thus, a defect predictor should lower pf as much as possible (ideal case is when pf = 0):

$$\text{pf} = \text{fp}/(\text{fp} + \text{tn}) \tag{2}$$

Balance (bal) is a single measure to indicate the tradeoff between pd and pf rates. It is defined as the normalized Euclidean distance from the desired point (1,0) to observed (pd, pf) in a ROC curve. Larger bal rates indicate that the performance is closer to the ideal case.

$$\text{bal} = 1 - \frac{\sqrt{(1-\text{pd})^2 + \text{pf}^2}}{\sqrt{2}} \tag{3}$$

We use Mann–Whitney *U* test to check for statistical differences between the performances of different predictors *in terms of bal* when reporting for individual projects. Hence, while we report pd and pf values, the main criteria for comparing performances is the bal values. We also report the (two-tailed) *p*-values for each individual comparison. In order to evaluate our hypotheses H_1 and H_2, we use Wilcoxon signed-rank test with the median balance results of all projects as our sample for different treatments (e.g. *WP + CP* vs. *WP*). In all tests we use the significance level $\alpha = 0.05$ (two-tailed). For reporting effect size, we use corrected Hedges' *g* defined in [18]. Experiment scripts are implemented in Matlab R2007a.[10]

3.6. Threats to validity

We assess the potential threats to the validity of our study in three categories: construct, internal and external validity following the guidelines provided by Wohlin et al. [33].

[10] http://www.mathworks.com/support/sysreq/release2007a/index.html.

3.6.1. Construct validity

The size metrics were collected from the binaries rather than source files, but for the projects we use, the correlation between the source lines of code and the size of binary code is near perfect [34]. Other researchers have also discussed that binary code may be a better representation of size as not all available code is included in the released products [35]. Further, the defect matching process has limitations, since it relies on the regular expression representations of the development guidelines and there is no information as to what level these guidelines have been followed. Like other researchers who have used these datasets previously, we had to assume that the level of conformance to those guidelines was to the best extent possible. Another threat to construct validity is the *interaction of different treatments*, specifically the confounding effects of the different possible subgroups in the datasets. Finally, though there are many alternative measures to assess the classification performance of defect predictors, the ones we have used have been widely employed and accepted by previous research studies. Further, the reported performance measures can be used to calculate other performance measures using the method (and the tool) developed by Bowes et al. [36].

3.6.2. Internal validity

RQ2 requires a representation of the early stages of a project, where various properties of the project may be different as the project may not have achieved a stable setting. Our experiment design does not take any chronological information into account. This is due to the lack of such information in our datasets. It can be argued that for multi version projects chronology can be estimated at least through an ordering of versions. However, this high level estimation does not fulfill our purposes of answering RQ2, as explained earlier (i.e. if a project already has a prior version, it is possible to use that earlier version rather than a combination of data from existing version and other projects). In order to address this representation problem, we have taken two measures: (i) using only the first versions of multi version projects and (ii) repeating the random 10% sample selection strategy many times to avoid *selection bias* and to represent different initial characteristics.

3.6.3. External validity

Problems with external validity are concerned about the generalization of results. To address this issue at least to some extent, we used a wide range of projects with varying contexts, i.e. academic student projects, open-source projects and commercial projects from different data sources. Nevertheless, it is difficult to draw general conclusions from empirical studies in software engineering and our results are limited to the analyzed data and context [37]. Further, the improvements may not be practical in real world due to poor prediction results. For instance, our NASA results are inline with what is reported in [28], however our results also include lower performance measures than the average values obtained for NASA projects. These could be related to data quality issues, selection of models or various other reasons and exact performance numbers should not be used as a basis for comparison for potential prediction applications for other projects. That is why we included pd and pf measures along with the bal measure, which is used as the main comparison criteria in this work.

4. Results

4.1. RQ1: Full WP vs. WP + CP

This section presents the results of our WP vs. WP + CP (i.e. within vs. mixed project) defect prediction experiments in order to address our first research question. The results are summarized with median performance values in Tables 3–5 for functional method level data, single-version class level data and multiple-version class level data projects, respectively. In all tables, the statistically significant results are denoted in bold face and the projects are ordered by their size in ascending order. We also provide summary of Mann–Whitney *U*-test results for all projects in Appendix A.

Table 3
RQ1: Single-version functional method level results sorted by size ascending. Statistically significant results are denoted in bold face.

Project #	WP			WP + CP			*p*-Value for bal
	Pd	Pf	Bal	Pd	Pf	Bal	
pc1	63	26	68	63	26	68	0.137
kc1	80	31	74	79	28	75	0.342
kc2	77	27	77	77	26	77	0.203
cm1	80	31	70	80	29	74	0.203
kc3	75	26	74	75	22	75	0.083
mw1	67	24	71	67	24	71	0.052
ar4	45	**6**	61	**75**	24	**75**	$\ll$0.001
ar3	88	40	70	88	40	70	0.342
mc2	80	36	67	80	**27**	**71**	$\ll$0.001
ar5	88	29	78	**100**	29	**80**	$\ll$0.001

For single version functional method level predictions, we observe significant improvement in *bal* in three out of 10 cases, while in the remaining seven cases there is no difference. For single version class level predictions, we observe significant improvement in only five cases and no difference in 14 cases. Finally, for multi-version class level predictions we observe significant improvement in *bal* with mixed project predictions in 21 out of 32 cases, no difference in two cases, and nine significant results in favor of within project predictions. In total, there are significant improvements in 29 of 61 cases with mixed project predictions.

Further, *single version functional method level* predictions yield two cases where pd is significantly better for mixed project predictions, while there is no difference in the remaining eight cases. For pf, there is one statistically significant case in favor of mixed project predictions, and one case in favor of within project predictions, while there is no difference in the remaining eight cases. For *single version class level* predictions, mixed project predictions yield five cases with significantly better pd's (no difference for the rest).

Table 4
RQ1: Single-version class level results sorted by size ascending. Statistically significant results are denoted in bold face. The *student* dataset is the merged version of all student projects.

Project #	WP			WP + CP			*p*-Value for bal
	Pd	Pf	Bal	Pd	Pf	Bal	
wspomaganiepi	0.90	0.60	0.55	0.85	0.40	0.65	0.427
sklebagd	0.70	0.55	0.48	0.75	0.45	0.57	0.519
nieruchomosci	1.00	0.48	0.66	1.00	0.30	0.79	0.107
pdftranslator	0.90	0.13	0.84	0.80	0.03	0.84	0.724
workflow	0.53	0.35	0.53	0.48	0.28	0.53	0.898
termoproject	0.65	0.38	0.54	0.80	0.30	0.68	0.077
berek	0.85	0.07	0.86	0.85	0.02	0.88	0.599
serapion	0.80	0.24	0.72	0.85	0.15	0.82	0.126
skarbonka	0.95	0.29	0.77	1.00	0.31	0.78	0.966
redaktor	0.75	0.28	0.69	0.82	0.28	0.73	0.454
arc	0.53	0.29	0.59	**0.78**	0.30	**0.70**	0.024
prop6	0.52	0.26	0.60	0.64	0.26	0.68	0.074
student	0.67	0.27	0.69	0.69	0.25	0.73	0.233
tomcat	0.75	0.23	0.75	**0.81**	**0.18**	**0.81**	$\ll$0.001
prop5	0.35	0.18	0.53	0.36	0.17	0.53	0.598
prop4	0.29	**0.14**	0.49	**0.36**	0.17	**0.53**	$\ll$0.001
prop3	0.39	0.22	0.54	0.40	0.20	0.55	0.262
prop1	0.37	**0.17**	0.54	**0.44**	0.20	**0.58**	$\ll$0.001
prop2	0.21	**0.09**	0.44	**0.23**	0.11	**0.45**	0.009

Table 5
RQ1: Multi-version class level results sorted by size ascending. Statistically significant results are denoted in bold face.

Project	Version #	WP			WP + CP			*p*-Value for bal
		Pd	Pf	Bal	Pd	Pf	Bal	
forrest	2	0.00	0.46	0.22	**0.20**	0.17	**0.42**	≪0.001
forrest	3	1.00	**0.17**	**0.88**	**1.00**	0.47	0.63	≪0.001
pbeans	2	1.00	**0.61**	**0.57**	1.00	0.66	0.53	≪0.001
log4j	2	0.68	**0.18**	0.74	**0.73**	0.24	**0.75**	≪0.001
ant	2	0.38	**0.33**	0.50	**0.55**	0.45	**0.55**	≪0.001
log4j	3	0.46	**0.25**	0.58	**0.65**	0.31	**0.67**	≪0.001
velocity	2	0.69	0.63	0.51	**0.82**	**0.58**	**0.57**	≪0.001
synapse	2	0.55	**0.10**	0.67	**0.65**	0.23	**0.70**	≪0.001
velocity	3	**0.81**	0.54	0.60	0.79	**0.46**	**0.64**	≪0.001
ivy	2	0.69	**0.44**	**0.62**	**0.72**	0.54	0.58	≪0.001
lucene	2	0.61	**0.44**	0.59	**0.73**	0.49	**0.61**	≪0.001
synapse	3	0.55	**0.25**	0.63	**0.64**	0.34	**0.65**	≪0.001
ant	3	0.75	**0.26**	0.74	**0.81**	0.29	0.75	0.253
jedit	2	0.75	**0.31**	**0.72**	**0.76**	0.34	0.70	≪0.001
jedit	3	0.77	**0.31**	**0.73**	**0.78**	0.33	0.72	≪0.001
poi	2	0.54	**0.60**	0.47	**0.81**	0.65	**0.52**	≪0.001
lucene	3	0.70	**0.41**	**0.64**	**0.71**	0.45	0.63	≪0.001
ant	4	0.73	0.25	0.74	**0.77**	0.25	**0.76**	≪0.001
ivy	3	0.70	**0.29**	0.71	**0.75**	0.31	**0.71**	≪0.001
jedit	4	0.90	**0.36**	**0.73**	0.90	0.37	0.73	≪0.001
poi	3	0.67	**0.23**	0.72	**0.71**	0.25	**0.73**	≪0.001
poi	4	0.73	**0.27**	0.73	**0.78**	0.28	**0.75**	≪0.001
xerces	2	0.14	0.37	0.34	**0.22**	**0.35**	**0.39**	≪0.001
jedit	5	0.55	**0.37**	**0.58**	0.55	0.38	0.58	≪0.001
xerces	3	0.44	0.11	0.60	**0.47**	0.11	**0.62**	≪0.001
camel	2	0.28	**0.16**	0.48	0.28	0.18	0.48	0.775
ant	5	0.77	**0.29**	**0.74**	0.77	0.30	0.73	≪0.001
xalan	2	0.40	**0.22**	0.55	**0.52**	0.30	**0.60**	≪0.001
camel	3	0.65	**0.34**	0.65	**0.66**	0.35	**0.66**	0.003
xalan	3	0.59	**0.30**	0.64	**0.66**	0.34	**0.66**	≪0.001
xalan	4	0.50	0.00	0.64	**0.55**	0.00	**0.68**	≪0.001
camel	4	0.54	**0.38**	0.58	**0.57**	0.40	**0.59**	≪0.001

However, in three of these five cases pf's are significantly better in favor of within project predictions. There is only one project, tomcat, where mixed project predictions produce significantly better pf along with significant better pd. Finally, for *multi-version class level* predictions, there are 26 cases with significantly better pd's favoring mixed project predictions, only one case in favor of within project prediction, and no difference in five cases. However, there are 25 cases with significantly better pf's favoring within project predictions, three cases in favor of within project prediction, and no difference in four cases. In velocity and xerces projects, both pd and pf are significantly better with mixed project predictors. In 21 cases pd's favor mixed project prediction, while pf's favor within project prediction. This is consistent with our previous study, where we observed that data from other projects increase pd at the cost of increased pf [11]. Nevertheless, the net effect in terms of balance is in favor of mixed project predictions with 21 cases, as stated above.

For the evaluation of H_1, comparing the paired median bal performances across all projects with Wilcoxon signed-rank test yields a statistically significant improvement in favor of *WP + CP* with p-val $\ll 0.001$. However, the effect size is $g = 0.25$ that is commonly interpreted as a *small* effect, and it is less than our target value 0.50 (*medium*).[11] Therefore, we fail to reject $H1_0$.

4.2. RQ2: Partial WP vs. WP + CP

This section presents the results of WP vs. WP(k%) + CP experiment, where we set $k = 10$ for reporting. Tables 6 and 7 summarize median performance values for single-version and multi-version class level data predictions, respectively. We report the summary of Mann–Whitney U-tests along with the box plots of all performance measures for all values of k, in Appendix A.

Table 6
RQ2: Single-version class level results sorted by size ascending. Statistically significant bal values are denoted in bold face. Column WP represents the full within project predictions, whereas column WP(10%) + CP represents mixed project predictions using only 10% of within project data.

Project	WP			WP(10%) + CP			*p*-Value for bal
	Pd	Pf	Bal	Pd	Pf	Bal	
wspomaganiepi	**1.00**	1.00	0.29	1.00	**0.00**	0.29	0.630
sklebagd	1.00	1.00	0.29	1.00	0.00	0.29	0.134
nieruchomosci	**1.00**	0.50	**0.65**	0.00	**0.00**	0.29	0.010
pdftranslator	**1.00**	0.00	**1.00**	1.00	0.00	1.00	0.013
workflow	0.50	0.50	0.50	0.50	0.00	0.50	0.933
termoproject	1.00	0.33	0.53	1.00	**0.17**	0.65	0.366
berek	1.00	0.00	1.00	1.00	**0.00**	1.00	0.800
serapion	1.00	0.25	0.82	1.00	**0.00**	1.00	0.051
skarbonka	1.00	0.50	0.65	1.00	**0.25**	0.82	0.062
redaktor	0.67	0.27	0.68	0.67	0.27	0.74	0.056
arc	0.67	0.29	0.62	**0.67**	**0.24**	**0.73**	≪0.001
prop6	0.57	0.29	0.63	**0.71**	**0.25**	**0.74**	≪0.001
student	0.67	0.29	0.68	0.67	**0.22**	**0.71**	0.036
tomcat	0.75	0.24	0.74	0.75	**0.15**	**0.78**	0.011
prop5	0.36	**0.17**	0.53	**0.43**	0.21	**0.58**	0.024
prop4	0.29	**0.14**	0.49	**0.40**	0.19	**0.55**	≪0.001
prop3	0.40	0.22	0.55	0.42	0.24	0.56	0.355
prop1	0.37	**0.16**	0.54	**0.48**	0.25	**0.59**	≪0.001
prop2	0.21	**0.09**	0.44	**0.26**	0.13	**0.47**	≪0.001

Table 6 shows that using only 10% of WP data, in eight out of 19 projects, significantly better results are achieved compared to full within project predictions, in terms of bal. There are two cases where within project predictors are significantly better, and there is no difference in nine cases. Table 7 summarizes the results of

[11] Non-standardized effect size is 2% increase in median (bal) performance.

Table 7
RQ2: Multi-version class level results sorted by size ascending. Statistically significant bal values are denoted in bold face. Column WP represents the full within project predictions, whereas column WP(10%) + CP represents mixed project predictions using only 10% of within project data.

Project	WP			WP(10%) + CP			*p*-Value for bal
	Pd	Pf	Bal	Pd	Pf	Bal	
pbeans	**0.75**	0.00	0.65	0.50	0.00	0.29	0.088
ivy	0.67	0.30	0.63	0.67	**0.20**	0.69	0.447
ant	1.00	0.20	0.65	**1.00**	0.20	**0.86**	0.006
log4j	0.67	0.30	0.68	0.67	**0.10**	**0.75**	0.002
synapse	1.00	0.21	0.72	1.00	0.21	0.82	0.116
lucene	0.78	0.30	0.70	0.72	**0.20**	0.70	0.146
velocity	0.73	0.40	0.66	0.63	**0.20**	0.66	0.236
poi	0.75	0.40	0.68	0.71	**0.25**	**0.71**	0.023
jedit	0.78	0.22	0.74	**0.78**	0.19	**0.80**	0.004
camel	1.00	0.17	0.80	**1.00**	0.15	**0.89**	0.004
xerces	**0.57**	0.30	0.60	0.42	**0.18**	0.58	0.128
xalan	0.73	0.25	0.71	**0.82**	0.25	**0.77**	0.004

WP vs. WP(*k*%) + CP only on the first version of multi-version open-source projects. Half of the projects (i.e. six out of 12) indicate that using only 10% of WP data is enough to achieve significantly better bal with the mixed model compared to the full within project model, and there is no difference in the other six cases. In total, there are significant improvements or no difference in 29 of 31 cases, when only 10% of within project data are used for mixed project prediction.

For *single version class level* predictions, mixed project predictions yield six cases with significantly better pd, and 10 cases with significantly better pf. Within project predictions have significantly better pd in three cases, and significantly better pf in four cases. There is no difference for the remaining 10 cases for pd and five cases for pf. In arc and prop6 projects, both pd and pf are significantly better with mixed project predictions. For the first versions of *multi-version class level* predictions, there are four cases with significantly better pd's favoring mixed project predictions, and two cases in favor of within project prediction, and no difference in six cases. Finally, there are six cases with significantly better pf's favoring within project predictions, and no difference in the remaining six cases.

For the evaluation of H_2, comparing the paired median performances,[12] when $k = 10$, across all projects with Wilcoxon signed-rank test yields a statistically significant improvement in favor of *WP*(10%) + *CP* with *p*-val = 0.002. Therefore, we reject $H2_0$.

4.3. Answers to research questions

1. *RQ1: How effective is using data from other projects in order to improve the performance of existing within project defect predictors?*:
   We were interested in whether it is feasible to use additional data from other projects to improve the performance of an existing, project specific defect prediction model in terms of defect detection performance. Although, we observed a statistically significant improvement in favor of mixed project models, the effect size was small. Therefore, the effectiveness of the mixed project approach in this case is not justifiable considering the extra effort associated with gathering data from other projects, since there is no guaranteed *practical* benefit for performance increase.
2. *RQ2: How much within project data (in per cent) should be enriched with data from other projects to achieve comparable performance with full within project data predictions?*:
   We were interested in whether mixed project defect prediction performs comparable to full within project prediction, using only a small portion of WP data, which would justify the extra effort associated with gathering CP data. In our experiments, we have observed evidence in favor, i.e. using only 10% of available WP data, which is also confirmed by statistical significance test results. Therefore, we strongly suggest the use of mixed prediction models at the early stages of development activities.

[12] The standardized and non-standardized effect sizes are $g = 0.17$ (small) and 3% increase in median (bal) performance, respectively.

Table 8
Summary of Mann Whitney *U*-test results for NASA projects when moving from within project to mixed project predictions (from Ref. [9]).

WP → WP + CP			Data sets
Pd	*Pf*	*Bal*	
Same	Same	Same	cm1, kc1, kc2, mw1, pc1, kc3
Same	Decreased	Increased	mc2

Table 9
Summary of Mann Whitney *U*-test results for SOFTLAB projects when moving from within project to mixed project predictions (from Ref. [9]).

WP → WP + CP			Data sets
Pd	*Pf*	*Bal*	
Same	Same	Same	ar3
Increased	Increased	Increased	ar4
Increased	Same	Increased	ar5

Table 10
Summary of Mann Whitney *U*-test results for single version class level data when moving from within project to mixed project predictions.

WP → WP + CP			Data sets
Pd	*Pf*	*Bal*	
Increased	Increased	Increased	prop1, prop2, prop4
Same	Increased	Increased	tomcat
Increased	Same	Increased	arc
Same	Same	Same	berek, nieruchomosci, pdftranslator, redaktor, serapion, skarbonka, sklebagd, termoproject, tomcat, workflow, wspomaganiepi, prop3,prop5,prop6

Table 11
Summary of Mann Whitney *U*-test results for multi-version class level data when moving from within project to mixed project predictions. Version numbers of datasets are provided after the dataset name with a 'v'.

WP → WP + CP			Data sets
Pd	*Pf*	*Bal*	
Increased	Increased	decreased	forrest v3, ivy v2, jedit v2 and v3, lucene v3, ant v5
Same	Increased	Decreased	pbeans v2, jedit v4, jedit v5
Same	Increased	Same	camel v2
Increased	Increased	Same	ant v3
Increased	Increased	Increased	*Rest of datasets*
Increased	Same	Increased	ant v4, forrest v2,xalan v4, xerces v3
Increased	Decreased	Increased	velocity v2, xerces v2

5. Conclusions

Defect prediction is an important area of research with potential applications in industry for early detection of the problematic parts in software allowing significant reductions in project costs and schedules. However, most defect prediction studies focus only on optimizing the performance of the models that are constructed

Table 12
Summary of Mann Whitney *U*-test results for student projects when moving from within project to mixed project predictions with varying WP ratio. 'Inc' and 'Dec' are the short forms of 'Increased' and 'Decreased', respectively.

Pd	*Pf*	*Bal*	*WP* → *WP*(*k*%) + *CP*							
			k = 10	*k* = 20	*k* = 30	*k* = 40	*k* = 50	*k* = 60	*k* = 70	*k* = 80
Dec	Same	Dec	pdftrans-lator							
Dec	Dec	Dec	nieru-chomosci							
Inc	Same	Same							sklebagd	
Dec	Same	Same			pdftrans-lator	pdftrans-lator				nieru-chomosci
Dec	Dec	Same	wspoma-ganiepi	nieru-chomosci, wspomaganiepi	nieru-chomosci	nieru-chomosci	nieru-chomosci	nieru-chomosci		
Same	Same	Same	redaktor, sklebagd, termoproject, workflow	berek, pdftranslator, redaktor, skarbonka, termoproject, workflow	redaktor, skarbonka, termoproject, workflow	berek, redaktor, skarbonka, sklebagd, termoproject, workflow, wspomaganiepi	berek, pdftranslator, redaktor, skarbonka, termoproject, workflow	berek, pdftranslator, redaktor, serapion, skarbonka, sklebagd, termoproject, workflow, wspomaganiepi	berek, pdftranslator, redaktor, skarbonka, termoproject, workflow	berek, pdftranslator, redaktor, skarbonka, sklebagd, termoproject, workflow
Same	Inc	Same	berek		berek					
Same	Dec	Same	serapion, skarbonka	serapion, student	serapion, wspomaganiepi, student	serapion	serapion, wspomaganiepi, student		nieru-chomosci, student	serapion, wspomaganiepi, student
Inc	Same	Inc		sklebagd			arc, sklebagd	arc	arc	arc
Inc	Dec	Inc	arc		arc	arc				
Same	Same	Inc			sklebagd				serapion	
Same	Dec	Inc	student	arc		student		student	wspoma-ganiepi	

Table 13
Summary of Mann Whitney *U*-test results for open-source projects when moving from within project to mixed project predictions with varying WP ratio. 'Inc' and 'Dec' are the short forms of 'Increased' and 'Decreased', respectively.

Pd	*Pf*	*Bal*	*WP* → *WP*(*k*%) + *CP*							
			k = 10	*k* = 20	*k* = 30	*k* = 40	*k* = 50	*k* = 60	*k* = 70	*k* = 80
Dec	Same	Dec					pbeans		pbeans	
Same	Same	Dec								
Dec	Same	Same	pbeans	pbeans	pbeans	pbeans		pbeans		pbeans
Dec	Dec	Same	xerces							
Same	Same	Same	synapse	ivy, poi, synapse	ant, ivy, lucene, poi	ant, ivy, lucene, poi	ant, ivy, lucene, poi	ant, ivy, lucene, poi	ivy, lucene, poi	ivy, jedit, lucene, poi, synapse
Same	Dec	Same	ivy, lucene, velocity	lucene, xerces	xerces	xerces	velocity, xerces	velocity, xerces	log4j, velocity	log4j, velocity
Inc	Same	Inc	ant, camel, jedit, xalan	camel, jedit, xalan	camel, jedit, xalan	camel, synapse, xalan	camel, jedit, xalan		jedit, xalan	xalan
Inc	Dec	Inc								
Same	Same	Inc		ant	synapse	jedit	synapse	jedit, synapse, xalan	ant, synapse	ant, camel
Same	Dec	Inc	tomcat, log4j, poi	tomcat, log4j, velocity	tomcat, log4j, poi, velocity	tomcat, log4j, velocity	tomcat, log4j	tomcat, camel, log4j	tomcat, camel, xerces	tomcat, velocity, xerces

Table 14
Summary of Mann Whitney *U*-test results for proprietary projects when moving from within project to mixed project predictions with varying WP ratio.

Pd	*Pf*	*Bal*	*WP* → *WP*(*k*%) + *CP*							
			k = 10	*k* = 20	*k* = 30	*k* = 40	*k* = 50	*k* = 60	*k* = 70	*k* = 80
Same	Same	Same	prop3	prop3, prop5	prop3, prop5	prop3, prop5	prop3, prop5	prop3, prop5	prop3, prop5	prop3, prop5
Same	Decreased	Same								
Increased	Increased	Increased	prop1, prop2, prop4, prop5	prop1, prop2, prop4	prop1, prop2, prop4 prop6	prop1, prop2, prop4	prop1, prop2, prop4	prop1, prop2, prop4	prop1, prop2, prop4	prop1, prop2, prop4
Increased	Same	Increased								
Increased	Decreased	Increased	prop6	prop6		prop6				
Same	Same	Increased					prop6	prop6	prop6	prop6

using data from individual projects, usually in the form of retrospective analysis. On the other hand, recent studies approach the problem from a different perspective: looking for ways to utilize cross project data for building defect predictors. These studies identify cross project defect prediction as a challenge with cost effective opportunities, such as using open-source data

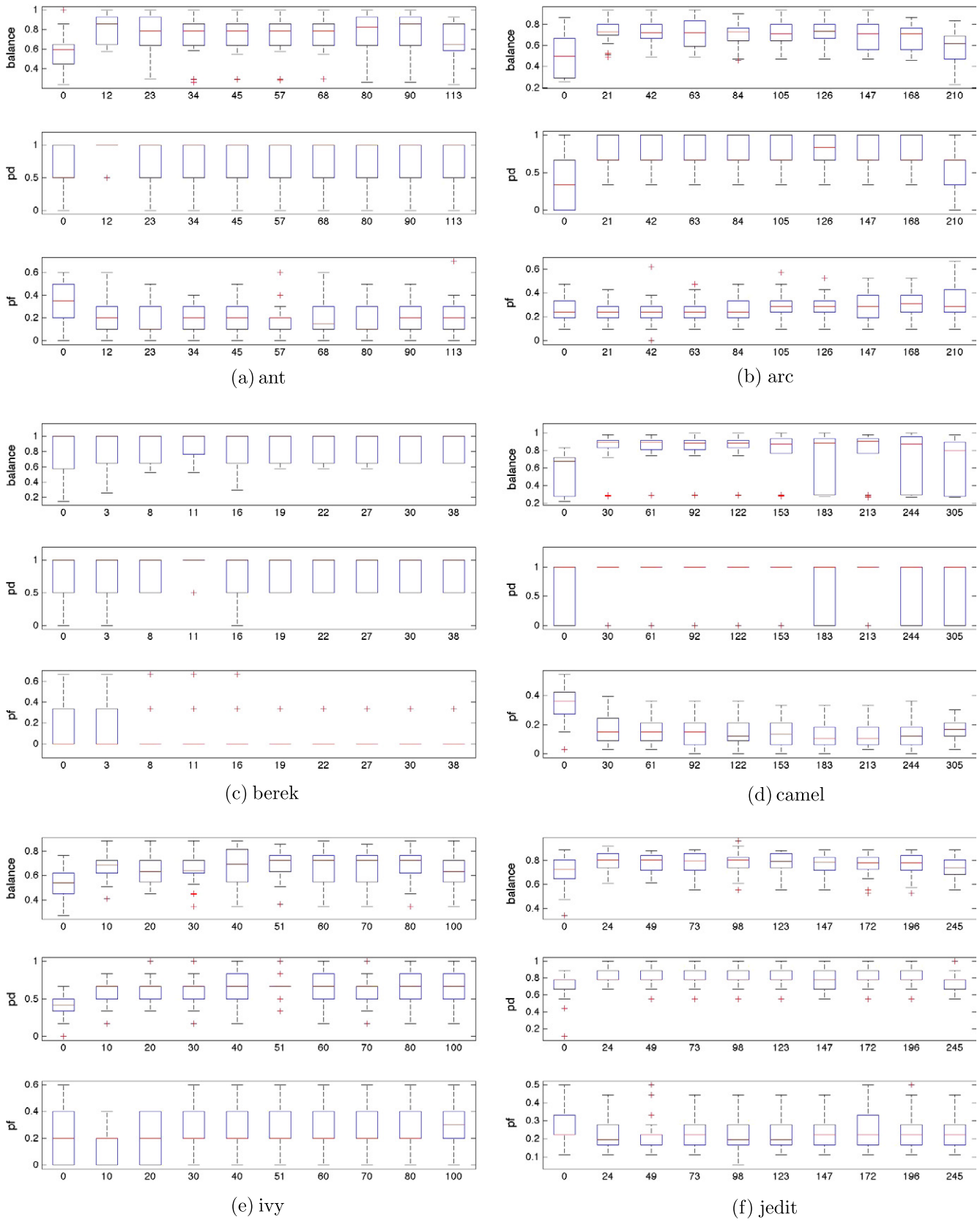

Fig. 3. Box plots for models from CP, WP(k%) + CP with different sizes of WP, to WP.

repositories for building or tuning models for other projects. We noticed that existing studies focus on the two ends of the spectrum, that is using either within or cross project data, which lead to the motivation behind this paper. We investigated the case

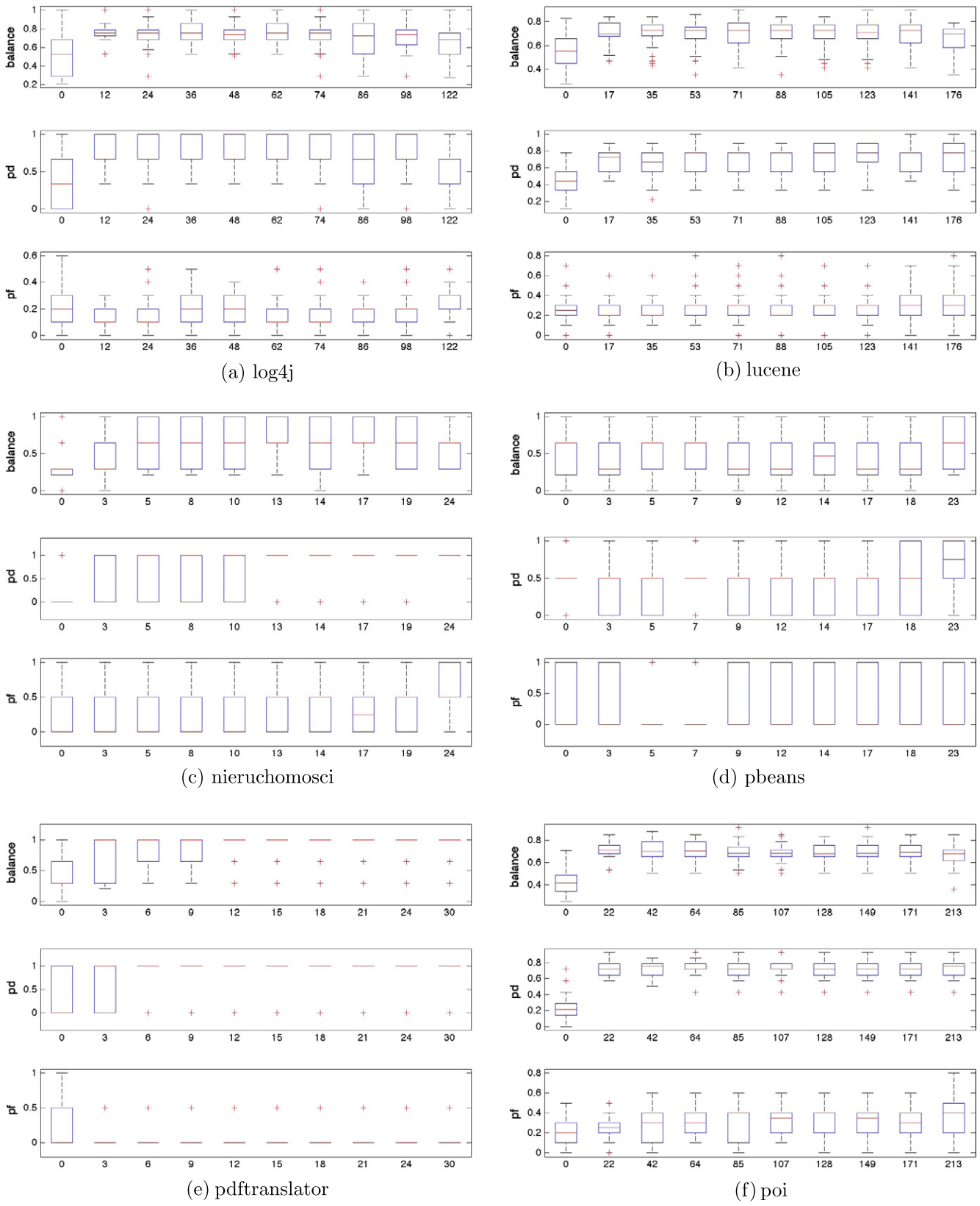

Fig. 4. Box plots for models from CP, WP(k%) + CP with different sizes of WP, to WP (Continued).

where models are constructed from a mix of within and cross project data, and checked for any improvements to within project defect predictions after adding data from other projects.

We conclude that the extra effort associated with collecting data from other projects is not feasible when there is already a within project defect predictor utilizing full project history.

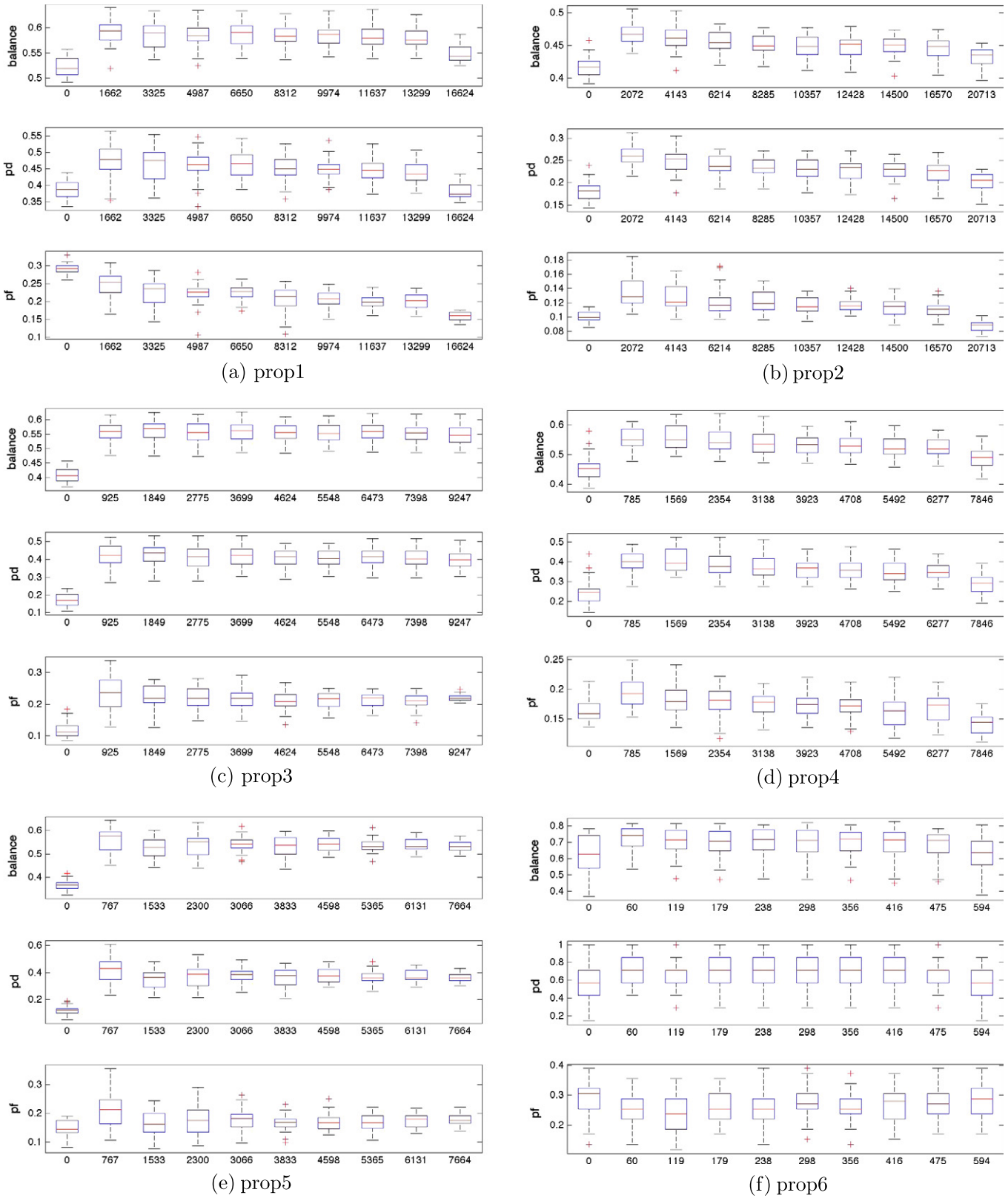

Fig. 5. Box plots for models from CP, WP(k%) + CP with different sizes of WP, to WP (Continued).

However, when there is limited project history (e.g. 10% of all development activity), mixed project predictions are justifiable as they perform as good as full within project models.

We see two potential future directions. In order to address the data scarcity problem to build defect prediction models in software engineering, one potential path could be employing complex

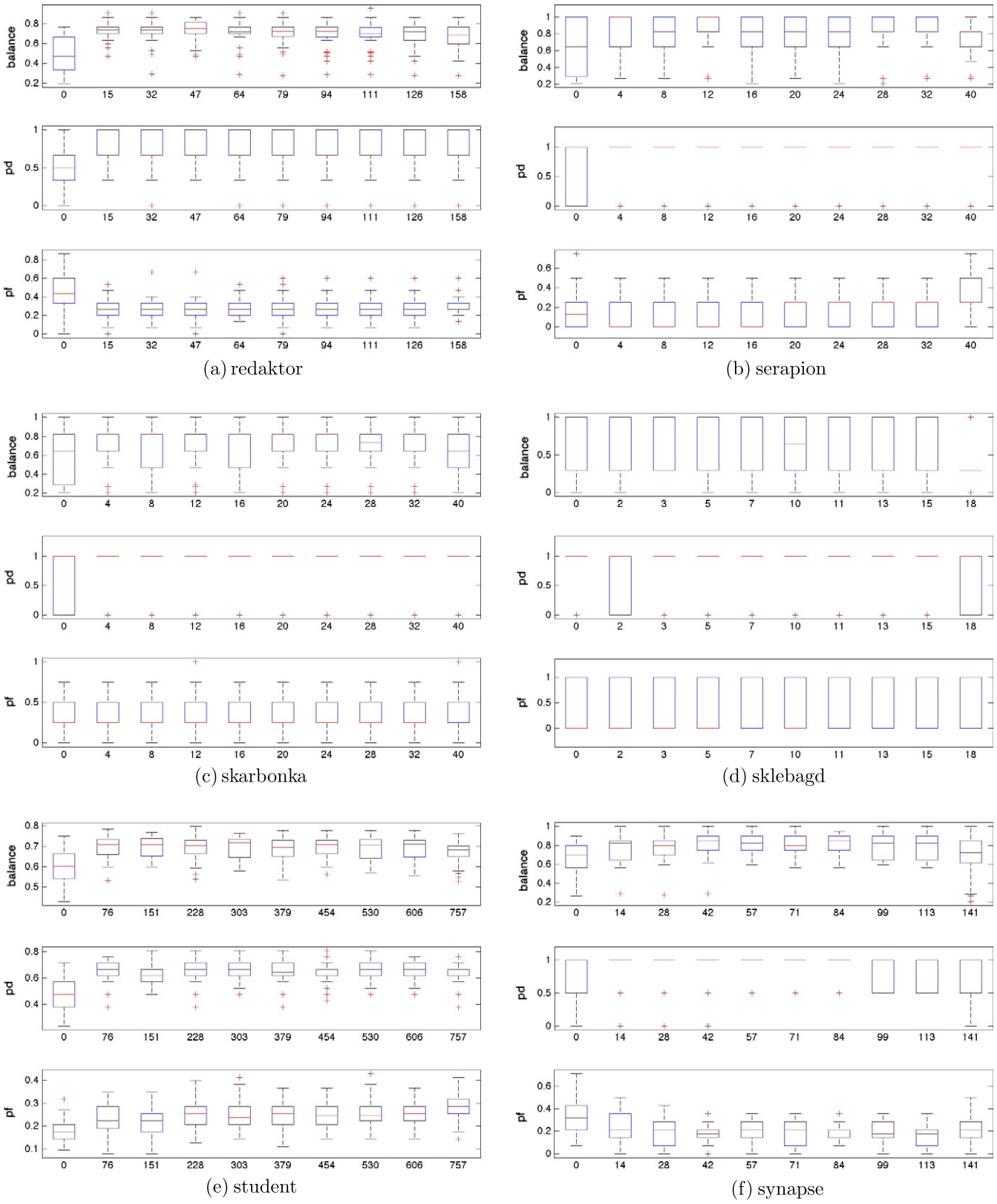

Fig. 6. Box plots for models from CP, WP(k%) + CP with different sizes of WP, to WP (Continued).

algorithms that use imputation techniques similar to recommender systems. In order to employ such algorithms either limited in-house data, cross project data or a combination can be used to impute the missing data. Another potential path may be to include other data sources (if they are readily available or relatively easier to collect) and metrics such as social networks, version control systems or issue repositories to build the prediction models.

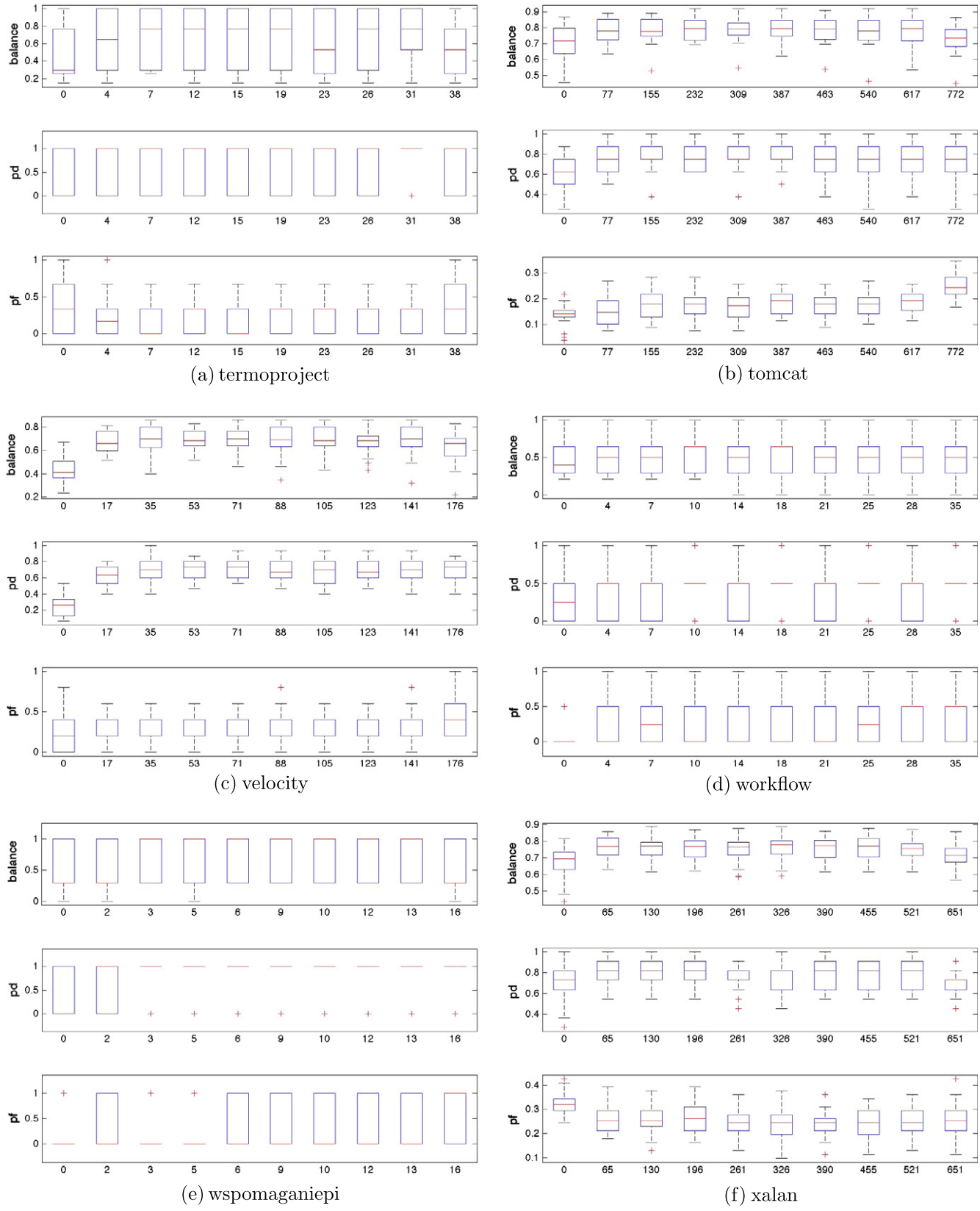

Fig. 7. Box plots for models from CP, WP(k%) + CP with different sizes of WP, to WP (Continued).

Acknowledgments

This research is supported in part by (i) TEKES under Cloud-SW Project and the Academy of Finland with Grant Decision No. 260871 (in Finland), (ii) NSERC Discovery Grant No. 402003-2012 (in Canada), and (iii) Turkish State Planning Organization (DPT) under the Project Number 2007K120610 (in Turkey). This work has been initiated when Dr. Ayşe Tosun Mısırlı was with the Dept. of

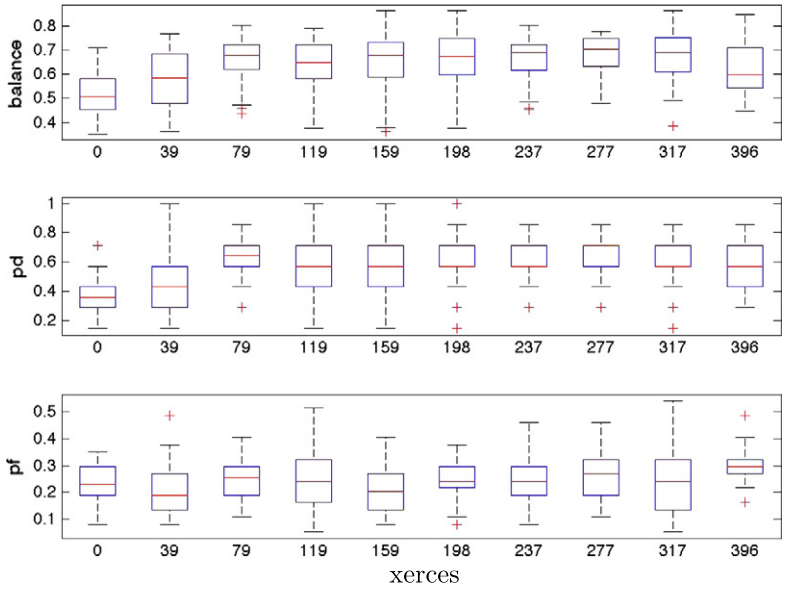

Fig. 8. Box plots for models from CP, WP(k%) + CP with different sizes of WP, to WP (Continued).

Computer Engineering at Boğaziçi University and completed when she moved to her current position. Authors would like to thank the anonymous reviewers for their insightful and constructive comments which have significantly improved the manuscript.

Appendix A

A.1. Summary of significance test results for RQ1

Mann Whitney U-test results are summarized for single and multi version projects in Tables 8–11 respectively.

A.2. Summary of significance test results for RQ2

Mann–Whitney U-tests are summarized in three tables for student (Table 12), open source (Table 13) and proprietary (Table 14) projects respectively. Each column indicates the results of moving from WP to WP(k%) + CP (k from 10% to 80%) models when k% WP data is included in the WP(k%) + CP model. Tables are sorted according to bal from *decreased, same* to *increased*.

A.3. Box plots for RQ2 experiments

Figs. 3–8 present box plots over 30 iterations of WP, CP, and WP(k%) + CP models for datasets according to alphabetical order. A boxplot for a dataset has three sub-figures for pd, pf and bal respectively. From left to right, WP data is changed from 0% (full CP model) to 90% (full WP model) in the increments of 10, and the size of WP data used in each experiment is presented in the X axis.

References

[1] L. Huang, B.W. Boehm, How much software quality investment is enough: a value-based approach, IEEE Softw. 23 (5) (2006) 88–95.
[2] M. Kläs, H. Nakao, F. Elberzhager, J. Münch, Support planning and controlling of early quality assurance by combining expert judgment and defect data – a case study, Empirical Softw. Eng. 15 (4) (2010) 423–454.
[3] T. Hall, S. Beecham, D. Bowes, D. Gray, S. Counsell, A systematic review of fault prediction performance in software engineering, IEEE Trans. Softw. Eng. 99 (PrePrints) (2011).
[4] E.J. Weyuker, T.J. Ostrand, R.M. Bell, Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models, Empirical Softw. Eng. 13 (5) (2008) 539–559.
[5] A. Tosun, A.B. Bener, B. Turhan, T. Menzies, Practical considerations in deploying statistical methods for defect prediction: a case study within the Turkish telecommunications industry, Inform. Softw. Technol. 52 (11) (2010) 1242–1257.
[6] B.A. Kitchenham, E. Mendes, G.H. Travassos, Cross versus within-company cost estimation studies: a systematic review, IEEE Trans. Softw. Eng. 33 (5) (2007) 316–329.
[7] B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D.J. Reifer, B. Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.
[8] C.J. Lokan, T. Wright, P.R. Hill, M. Stringer, Organizational benchmarking using the ISBSG data repository, IEEE Softw. 18 (5) (2001) 26–32.
[9] B. Turhan, A.T. Misirli, A.B. Bener, Empirical evaluation of mixed-project defect prediction models, in: EUROMICRO-SEAA, IEEE, 2011, pp. 396–403.
[10] L.C. Briand, W.L. Melo, J. Wst, Assessing the applicability of fault-proneness models across object-oriented software projects, IEEE Trans. Softw. Eng. 28 (7) (2002) 706–720.
[11] B. Turhan, T. Menzies, A.B. Bener, J.S.D. Stefano, On the relative value of cross-company and within-company data for defect prediction, Empirical Softw. Eng. 14 (5) (2009) 540–578.
[12] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, in: H. van Vliet, V. Issarny (Eds.), ESEC/SIGSOFT FSE, ACM, 2009, pp. 91–100.
[13] A.E.C. Cruz, K. Ochimizu, Towards logistic regression models for predicting fault-prone code across software projects, in: ESEM, 2009, pp. 460–463.
[14] B. Turhan, A.B. Bener, T. Menzies, Regularities in learning defect predictors, in: M.A. Babar, M. Vierimaa, M. Oivo (Eds.), PROFES, Lecture Notes in Business Information Processing, vol. 6156, Springer, 2010, pp. 116–130.
[15] A. Nelson, T. Menzies, G. Gay, Sharing experiments using open-source software, Softw. Pract. Exper. 41 (3) (2011) 283–305.
[16] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE '10, ACM, New York, NY, USA, 2010, pp. 9:1–9:10.
[17] Y. Liu, T.M. Khoshgoftaar, N. Seliya, Evolutionary optimization of software quality modeling with multiple repositories, IEEE Trans. Softw. Eng. 36 (6) (2010) 852–864.
[18] V.B. Kampenes, T. Dybå, J.E. Hannay, D.I.K. Sjøberg, Systematic review: a systematic review of effect size in software engineering experiments, Inf. Softw. Technol. 49 (11–12) (2007) 1073–1086.
[19] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, B. Turhan, The promise repository of empirical software engineering data, 2012.
[20] T.J. McCabe, A complexity measure, IEEE Trans. Softw. Eng. 2 (4) (1976) 308–320.
[21] M.H. Halstead, Elements of Software Science, Elsevier, 1977.
[22] M. Jureczko, D. Spinellis, Using object-oriented design metrics to predict software defects, in: Proceedings of the Fifth International Conference on Dependability of Computer Systems, Monographs of System Dependability, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Poland, 2010, pp. 69–81.
[23] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Trans. Softw. Eng. 20 (6) (1994) 476–493.
[24] B. Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity, Prentice-Hall, 1996.

[25] J. Bansiya, C.G. Davis, A hierarchical model for object-oriented design quality assessment, IEEE Trans. Softw. Eng. 28 (1) (2002) 4–17.
[26] M.-H. Tang, M.-H. Kao, M.-H. Chen, An empirical study on object-oriented metrics, in: IEEE METRICS., IEEE Computer Society, 1999, pp. 242–249.
[27] R. Martin, OO design quality metrics – an analysis of dependencies, in: Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics. OOPSLA'94, 1994.
[28] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (1) (2007) 2–13.
[29] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, Y. Jiang, Implications of ceiling effects in defect predictors, in: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, PROMISE '08, ACM, New York, NY, USA, 2008, pp. 47–54.
[30] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Softw. Eng. 34 (4) (2008) 485–496.
[31] X. Wu, V. Kumar, J.R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A.F.M. Ng, B.L. 0001, P.S. Yu, Z.-H. Zhou, M. Steinbach, D.J. Hand, D. Steinberg, Top 10 algorithms in data mining, Knowl. Inform. Syst. 14 (1) (2008) 1–37.
[32] T. Menzies, A. Dekhtyar, J.S.D. Stefano, J. Greenwald, Problems with precision: a response to comments on data mining static code attributes to learn defect predictors, IEEE Trans. Softw. Eng. 33 (9) (2007) 637–640.
[33] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.
[34] M. Lumpe, R. Vasa, T. Menzies, R. Rush, B. Turhan, Learning better inspection optimization policies, Int. J. Softw. Eng. Knowl. Eng. 22 (5) (2012) 621–644.
[35] J. Krinke, Identifying similar code with program dependence graphs, in: Proc. Working Conf. Reverse Engineering (WCRE), IEEE Computer Society Press, 2001, pp. 301–309.
[36] D. Bowes, T. Hall, D. Gray, Comparing the performance of fault prediction models which report multiple performance measures: recomputing the confusion matrix, in: Proceedings of the 8th International Conference on Predictive Models in Software Engineering, PROMISE '12, ACM, New York, NY, USA, 2012, pp. 109–118.
[37] V.R. Basili, F. Shull, F. Lanubile, Building knowledge through families of experiments, IEEE Trans. Softw. Eng. 25 (4) (1999) 456–473.