A Comparative Study to Benchmark Cross-project Defect Prediction Approaches

Steffen Herbold, Alexander Trautsch, Jens Grabowski

Abstract—Cross-Project Defect Prediction (CPDP) as a means to focus quality assurance of software projects was under heavy investigation in recent years. However, within the current state-of-the-art it is unclear which of the many proposals performs best due to a lack of replication of results and diverse experiment setups that utilize different performance metrics and are based on different underlying data. Within this article, we provide a benchmark for CPDP. We replicate 24 approaches proposed by researchers between 2008 and 2015 and evaluate their performance on software products from five different data sets. Based on our benchmark, we determined that an approach proposed by Camargo Cruz and Ochimizu (2009) based on data standardization performs best and is always ranked among the statistically significant best results for all metrics and data sets. Approaches proposed by Turhan *et al.* (2009), Menzies *et al.* (2011), and Watanabe *et al.* (2008) are also nearly always among the best results. Moreover, we determined that predictions only seldom achieve a high performance of 0.75 recall, precision, and accuracy. Thus, CPDP still has not reached a point where the performance of the results is sufficient for the application in practice.

Index Terms—cross-project defect prediction, benchmark, comparison, replication

1 Introduction

Defect prediction as a means to focus quality assurance has been an often addressed research topic for a long time, with early work being performed in the nineties, e.g., by Basili *et al.* [1]. In the last decade, Cross-Project Defect Prediction (CPDP), i.e., the prediction of defects using training data from other projects, grew from being an ignored aspect into a quite large sub-topic of software defect prediction in general. A recent mapping study [2] found 50 publications between 2002 and 2015 that specifically address cross-project aspects. However, the survey also highlighted a lack of comparability and replication and could, therefore, not determine which approaches are best. The reasons for this are manifold, ranging from evaluations on different data sets or using different performance metrics, to different ways the training data is set up, e.g., using training data from multiple products together or using each product once for training and report the mean performance achieved. However, knowing which approaches work best is important for the further advancement of the state-of-the-art, because without such knowledge, it is hard to claim that (or even if) the state-of-the-art is advanced due to a new proposal.

Within this article, we perform a benchmark for the comparison of the state-of-the-art of CPDP approaches. The contributions of this benchmark are:

• S. Herbold, A. Trautsch, and J. Grabowski are with the University of Goettingen, Institute of Computer Science, Göttingen, Germany.
E-mail: {herbold,grabowski}@cs.uni-goettingen.de
alexander.trautsch@stud.uni-goettingen.de

- *Approach Comparison:* a systematic comparison and ranking of 24 CPDP approaches based on 85 software products from five different data sets.
- *Analysis Methodology:* an analysis methodology for the combination of rankings on different data sets with different performance metrics into a single overall ranking.
- *Replications:* the results of the benchmark are completely reproducible; all results and techniques are available online and as open source [2], [3].

Within the remainder of this article, we will first introduce the foundations in Section 2. We then proceed with the discussion of the related work in Section 3. Then, we introduce the methodology of our benchmark, including the research questions that we answer through the benchmark, the data used, as well as the performance metrics and evaluation strategy in Section 4. Afterwards, we present the results of the benchmark in Section 5 and discuss our findings, including lessons learned and threats to validity, in Section 6. Finally, we conclude the article in Section 7.

2 Foundations

Within this section, we introduce the definitions we use throughout this article, the general workflow of CPDP, as well as the performance metrics we require for for the discussion of the related work and our benchmark.

2.1 Definitions and Notations

Within this article, we define the terms software product, revision, and software project as follows:

- A software product is a specific revision of a software project. Hence, a software project may have multiple products, which are different versions of the software.

Additionally, we use the following notations to discuss the related work in Section 3.

- S denotes a software product used for training.
- S^* denotes the target product of the defect prediction.
- s denotes an instance of a product used for training S.
- s^* denotes an instance of the target product S^*.
- $m(s)$ denotes the metric values of a metric m of an instance s.
- $\hat{m}(s)$ denotes a transformation of a metric m of an instance s.
- $median(m(S))$, resp. $mean(m(S))$ denote the median, resp. mean value of the metric m for the product S.

2.2 General Workflow

Figure 1 depicts the general workflow of CPDP experiments. Within this article, we consider the setting of *strict CPDP* [4]. We have a data set with information about software products. One of these software products is selected as target product. The other products of the data sets are used for the defect prediction model. If other revisions of the target product exist in the data set, they are also discarded such that no information from the same project context remains.

Three other variants of CPDP studies can be found in the literature: *mixed CPDP*, Mixed-Project Defect Prediction (MPDP), and *pair-wise CPDP*. With mixed CPDP, old revisions of the target product are also allowed for training, thereby mixing the cross-project with the within-project context. While both *strict* and *mixed* CPDP experiments are common, they were always just referred to as CPDP in the past. The differentiation between *strict* and *mixed* was recently introduced by Herbold *et al.* [4]. MPDP goes one step further and even allows some labelled data from the target product itself [5]. In all of the above, multiple products are used together to train the defect prediction model. Pair-wise CPDP takes a different approach towards using the training data. Here, the CPDP approach is applied separately for each product in the data set as training data. Then, the mean or median of the performance of these pair-wise predictions is used to estimate the performance of the CPDP approach.

2.3 Performance Metrics

We reference the seven performance metrics within the discussion of the related work and our benchmark. The first six metrics are *recall*,[1] *precision*, *accuracy*, *F-measure*, *G-measure*, and *MCC*. The metrics are defined

1. instead of *recall*, *PD* or *tpr* are also used in the literature. *PD* stands for probability of detection and *tpr* for true positive rate.

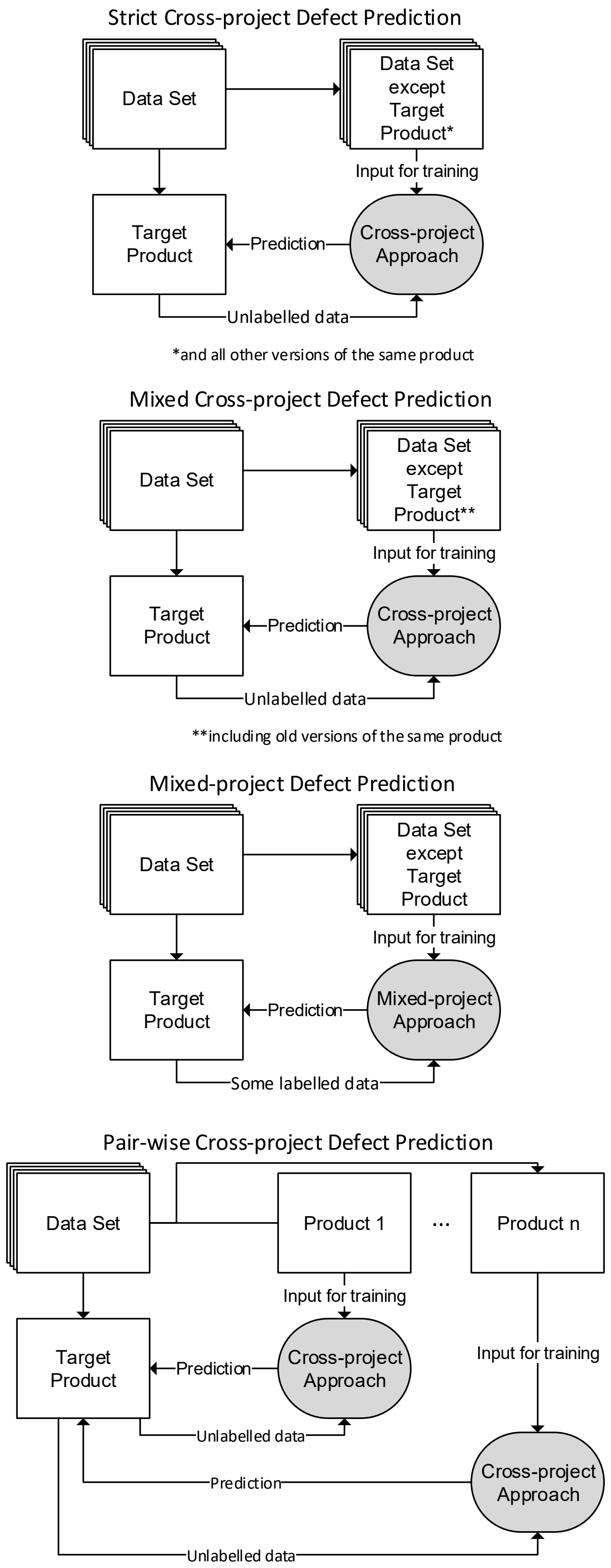

Fig. 1. Different types and general workflow of CPDP.

as

$$recall = \frac{tp}{tp + fn}$$
$$precision = \frac{tp}{tp + fp}$$
$$accuracy = \frac{tp + tn}{tp + fn + tn + fp}$$
$$F\text{-}measure = 2 \cdot \frac{recall \cdot precision}{recall + precision}$$
$$G\text{-}measure = 2 \cdot \frac{recall \cdot (1 - pf)}{recall + (1 - pf)}$$
$$MCC = \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn))}}$$

where tp, respectively, tn are the number of true positive, resp., true negative predictions, fp, resp., fn are the number of false positive, resp., false negative predictions, and the probability of false prediction $pf = \frac{fp}{tn+fp}$. The *recall* measures how many of the existing defects are found, the *precision* measures how many of the found results are actually defects, and the *accuracy* the percentage of correct predictions. The *F-measure* is the harmonic mean between *recall* and *precision* and the *G-measure* is the harmonic mean between *recall* and *pf*. *MCC* stands for Matthews Correlation Coefficient and it measures the correlation between the observed and predicted classifications with values in $[-1, 1]$. Positive values indicate a correlation, negative values indicate an inverse correlation, and values close to zero indicate no correlation at all.

The seventh metric is the Area Under the ROC Curve (AUC), which is defined using the Receiver Operating Characteristic (ROC). ROC is the curve of *pf* plotted versus the *recall*. The value is distributed between zero and one, a value of 0.5 indicates a performance similar to random guessing.

3 Related Work

The discussion of literature related to this article is two-fold: First, we discuss the literature on CPDP. Second, we consider other benchmarks for defect prediction and how they influenced our work.

3.1 Cross-Project Defect Prediction (CPDP)

Our discussion of CPDP is based on the mapping study by Herbold [2]. We divide the presentation into two parts: related work that was replicated as part of our benchmark and related work that was not replicated. Our criteria for replication were the following.

- Publication in or before December 2015.
- The approach for CPDP does not require any labeled data from the target product, i.e., Mixed-Project Defect Prediction (MPDP).
- The approach is based on software metrics that are based on the source code or the source code history. Other information about the training or target product, like the source code itself or context factors are not required.
- The approach works with the same metrics for the target and training data, i.e., using data from sources with different metrics is not considered.

Using these criteria, we determined 26 publications in which 24 approaches for CPDP were proposed which we replicated. These publications are discussed in tables 1–3, including a short description, adoptions for the replication if any where required, and an acronym which we use in the following to refer to the approach. In tables 4–5, we discuss the related work that was not replicated, including a short description and the reason for the exclusion.

3.2 Defect Prediction Benchmarks

To the best of our knowledge no benchmark on CPDP exists, yet. Hence, we cannot put the results of this benchmark or its setting directly in context with another CPDP benchmark. Instead, we discuss three benchmarks on Within-Project Defect Prediction (WPDP) and the procedures they used to rank results.

The first benchmark was performed by Lessmann *et al.* [60] and evaluated the impact of different classifiers on WPDP. Within their benchmark, the authors follow the proposal by Demšar [61] to use the Friedman test [62] to determine if the differences in performance between classifiers can be explained by randomness or are statistically significant. Then, they use the post-hoc Nemenyi test [63] to determine, which of the classifiers differ significantly. The second benchmark was performed by D'Ambros *et al.* [64] using different kinds of metrics and different classification models. The evaluations were also based on the Friedman test and the Nemenyi test.

The third benchmark was conducted by Ghotra *et al.* [65] performed with the same goal as the benchmark by Lessmann *et al.*, i.e., the comparison of the impact of classifiers on the performance of WPDP. However, in comparison to the benchmarks by Lessmann *et al.* and D'Ambros *et al.*, the authors did not follow the proposal by Demšar for the comparison of results. Instead, they used a different statistical test, i.e., the Scott-Knott hierarchical clustering. The Scott-Knott clustering is based on the results of an ANalysis Of VAriance (ANOVA) statistical test [66]. Based on the ANOVA results, clusters are determined, whereas the classifiers in each cluster are significantly different from the classifier in the other clusters. By sorting these clusters by their mean value, a ranking of the classifiers is created. Ghotra *et al.* argue that using Scott-Knott has the advantage that the generated clusters do not overlap, i.e., a fixed ranking is generated. This is not necessarily achieved by the Nemenyi test, where the results may overlap, which hinders a clear ranking of approaches. The drawback

TABLE 1
Related work on CPDP included in the benchmark (Table 1/3). Acronyms are defined following the authors and reference in the first column.

Publication	Short Description	Adoption for Benchmark
Khoshgoftaar *et al.*, 2008 [6] (Khoshgoftaar08)	The authors propose to use a classifier trained for each product in the training separately and then use the majority vote of the classifiers for each product for the classification. The authors also investigate other approaches, however, they determine that the majority vote is the best among the suggestions.	As is.
Watanabe *et al.*, 2008 [7] (Watanabe08)	The authors propose to standardize the target data as $\hat{m}_i(s^*) = \frac{m_i(s^*) \cdot mean(m_i(S))}{mean(m_i(S^*))}$ to improve the homogeneity between training and target data.	Standardization of the training data instead of the target data, i.e.,$\hat{m}_i(s) = \frac{m_i(s) \cdot mean(m_i(S^*))}{mean(m_i(S))}$.
Turhan *et al.*, 2009 [8] (Turhan09)	The authors propose to first transform the metric data with the logarithm and then apply a relevancy filter to the available training data based on the k-Nearest Neighbor (NN) algorithm. Through the relevancy filter, the k nearest instances to each instance in the target data are selected. The authors suggest to use $k = 10$.	As is. The original publication by Turhan *et al.* leaves room for interpretation when the logarithm is applied to the data, i.e. before the relevancy filter or after the relecvancy filter and only to the classification. We use the former interpretation, as we think that if skew is treated, it makes most sense to treat it for all skew-sensitive algorithms, which includes k nearest neighbor filtering.
Zimmermann *et al.*, 2009 [9] (Zimmermann09)	In addition to a large study on the feasibility of CPDP in general, the authors propose the training of a decision tree based on context factors and metric distributions that can be used to estimate which product is well suited for the defect prediction of another product. The decision trees can be used to improve either *recall*, *precision*, or *accuracy*.	We only consider the metric distributions and no context factors. Moreover, we use a single tree based on the *F-measure* as we require a single criterion for full automation of the approach. The *F-measure* is close to the original approach as it is the harmonic mean of *recall* and *precision*.
Camargo Cruz and Ochimizu, 2009 [10] (CamargoCruz09)	The authors propose to standardize the target and training data using the logarithm and the median of the training data as reference, i.e., $\hat{m}_i(s) = \log(1+m_i(s)) + median(\log(1+m_i(S))) - median(\log(1+m_i(S^{train})))$. The authors consider a single training product as reference.	We consider multiple products as training data. Therefore, we use the target data S^* instead of the training data S^{train} as reference point, i.e., $\hat{m}_i(s) = \log(1 + m_i(s)) + median(\log(1 + m_i(S))) - median(\log(1 + m_i(S^*)))$.
Liu *et al.*, 2010 [11] (Liu10)	The authors propose S-expression trees trained using a genetic program with a validation-and-voting strategy that uses parts of the training data for internal validation and the majority votes of multiple genetic programs for classification. Other approaches for the selection of the best result for the genetic program are also considered, but the authors determined validation-and-voting to be superior to them.	As is.
Menzies *et al.*, 2011 [12] and 2013 [13] (Menzies11)	Creation of a local model through clustering of the training data with the WHERE algorithm and afterwards classification of the results with the WHICH rule learning algorithm. Separate WHICH rules are created for each cluster to create local models.	As is, but also with other classifiers in addition to WHICH.
Ma *et al.*, 2012 [14] (Ma12)	The authors propose to apply data weighting based on the similarity using the concept of gravitation. The weights are calculated as $w_s = \frac{simatts_s}{(p - simatts_s + 1)^2}$, where $simatts$ are the number of attributes of an instance whose value is within the range of the target data and p the number of attributes. The weighting is used together with the Naïve Bayes algorithm and the approach is coined Transfer Naïve Bayes.	As is, but also with other classifiers in addition to Naïve Bayes.
Peters and Menzies, 2012 [15] (Peters12)	The authors propose a data privacy mechanism called MORPH that should not negatively affect the predictions results. With MORPH, all instances s are randomly modified using their nearest unlike neighbor s^{NUN}, i.e., the closest instance that has a different classification, such that $m_i(s) = m_i(s) + r \cdot (m_i(s) - m_i(s^{NUN}))$ with r a random value between 0.15 and 0.35.	As is.
Uchigaki *et al.*, 2012 [16] (Uchigaki12)	The authors propose a logistic ensemble, i.e., a collection of logistic regression models over the single attributes. The weighted majority vote of the single attribute models determine the overall classification. The weights are determined using the goodness of fit of the model.	Since the authors are vague on the definition of goodness of fit they are using, we decided to use *MCC*.
Canfora *et al.*, 2013 [17] and Canfora *et al.*, 2015 [18] (Canfora13)	The authors propose a multi-objective approach called MultiObjective DEfect Predictor (MODEP). They use a genetic program to estimate the coefficients of a logistic regression model that is optimal in terms of both cost and effectiveness, i.e., that finds as many defects for as little costs possible. The outcome of the training is a family of logistic regression models, each of which is optimal for a specific combination of effectiveness measured in *recall* and cost measured in Lines Of Code (LOC) that must be reviewed.	Since the approach yields a family of classifiers, but we require a single classifier as outcome, we use the approach with a fixed effectiveness with a *recall* of 0.7. Hence, we will select the classifier that has the minimal costs and at least a *recall* of 0.7.

TABLE 2
Related work on CPDP included in the benchmark (Table 2/3). Acronyms are defined following the authors and reference in the first column.

Publication	Short Description	Adoption for Benchmark
Peters *et al.*, 2013 [19] (Peters13)	The authors suggest the relevancy filter CLIFF that can be used together with the MORPH data privatization that was previously proposed. CLIFF uses a combination of binning for each metric and the conditional probability of the classifications for each bin. This way, the *power* of each instance is calculated and then the p percent of the instances with the highest power are selected. Afterwards, the data is privatized with MORPH.	We use $p = 40\%$ within the benchmark as this percentage yields the best results within the original study.
Herbold, 2013 [20] (Herbold13)	The author proposed to use the k-NN algorithm for the relevancy filtering of products. The k-NN uses the distances between the distributional characteristics of the products to determine the neighborhood. The authors suggest to select $k = \lfloor \frac{\#products}{2} \rfloor$ of the products for training. Additionally, the authors propose a strategy they refer to as equal weighting to treat a potential bias in the data. The approach works by weighting the training data such that the overall weight of the defect-prone instances is the same as for the non-defect-prone instances. The authors also consider relevancy filtering of products using the EM clustering algorithm, but this approach is outperformed by the k-NN algorithm.	As is.
Z. He *et al.*, 2013 [21] (ZHe13)	The authors propose a relevancy filter for products as well as an attribute selection approach using the separability of the training data from the target data. Concretely, they train a logistic regression model that tries to differentiate between a training product and the target product. The *accuracy* of this model is used as foundation for the separability. The authors suggest to select k products that cannot be separated easily from the target data, i.e., where the separation has a low *accuracy*. Then, the attributes with the highest information gain for the separation are removed, as they are likely different between the training and target data. Afterwards, undersampling is applied to the data of each project. Finally, the authors propose to use a bagging approach with one classifier trained for each of the k selected products and the majority vote as overall classification.	The authors use $k = 10$ products for their data selection, which is about one third of the data. Since some data sets we use have fewer products, we set $k = \lfloor \frac{\#products}{3} \rfloor$ for all data sets.
Nam *et al.*, 2013 [22] (Nam13)	The authors suggest a combination of standardization and Transfer Component Analysis (TCA), which then leads to the TCA+ approach. First, they determine which of five different standardization approaches should be selected: none, min-max normalization, Z-score standardization, Z-score standardization where all products are standardized based on all training data, and Z-score standardization where all products are standardized based on the target data. Then, the TCA is applied to determine a mapping of the training and target data to a homogeneous metric space.	As is.
Panichella *et al.*, 2014 [23] (Panichella14)	The authors propose a COmbined DEfect Predictor (CODEP), i.e., a classifier that is trained not directly on the training data, but indirectly on the outcome of other classifiers, which are trained on the training data. This way, CODEP combines the output of other classifiers into one meta-classifier. For the combination, the authors propose Logistic Regression and Bayesian Networks, as internal classifiers the authors propose Alternating Decision Trees, Bayesian Networks, Decision Tables, Logistic Regression, Multilayer Perceptrons, and RBF Networks.	As is.
Ryu *et al.*, 2014 [24] (Ryu14)	The authors propose a similarity based resampling approach that is used in combination with boosting. First, roughly 10% of the training data are selected randomly as hold-out data to evaluate the boosting results. Then, for each boosting iteration, the data is resampled based on the similarity. The similarity is determined using the data weighting approach proposed by Ma *et al.* [14]. The boosting itself is AdaBoost [25], with the difference being the similarity-based sampling.	As is.
P. He *et al.*, 2015 [26] (PHe15)	The authors propose to only use the best metrics for the training of prediction models. To determine the best metrics, the authors train a Within-Project Defect Prediction (WPDP) prediction model for each product in the training data. Then, they count how often each metric is used in the prediction models as a measure for how good an metric is and determine the best k metrics. Then, the optimal best metric set is calculated as a subset of pair-wise uncorrelated metrics of the best metrics that have the largest overlap with the subset of metrics that is determined by a Correlation-based Feature Subset (CFS) [27] metric selection.	As is, except that we automatically pick the optimal k based on the overlap with the CFS metrics, while this is done through visual analysis in the original publication. Moreover, if the optimal $k > 30$, we set $k = 30$ to limit the size of the power set that is used to determine the optimal subset.

TABLE 3
Related work on CPDP included in the benchmark (Table 3/3). Acronyms are defined following the authors and reference in the first column.

Publication	Short Description	Adoption for Benchmark
Peters *et al.*, 2015 [28] (Peters15)	The authors propose LACE2 as an extension of CLIFF and MORPH for privatization. With LACE2, the authors introduce a shared cache, in which data owners add their data one after the other. New data is only added, if it is not already represented in the data. To this aim, the authors propose a variant of the leader-follower algorithm. With this variant, the authors check how close a new instance is to its nearest unlike neighbor in the cache. Depending on the distance, LACE2 decides if an instance should be added. This way, not all data from all products is added to the cache, which automatically improves the privacy as fewer data needs to be shared.	As is.
Kawata *et al.*, 2015 [29] (Kawata15)	The authors propose to use the DBSCAN clustering algorithm [30] as a relevancy filter for the data. To this aim, the authors propose to cluster all training data together with the target data. All instances from the training data that are in the same cluster as any instance of the target data are used for training.	As is.
Y. Zhang *et al.*, 2015 [31] (YZhang15)	The authors propose the usage of the ensemble classifiers Average Voting and Maximum Voting, which internally use the classifiers Alternating Decision Trees, Bayesian Networks, Decision Tables, Logistic Regression, Multilayer Perceptrons, and RBF Networks. Moreover, the authors propose to use Bagging [32] and Boosting [25] for the classifiers Logistic Regression and Naïve Bayes.	As is.
Amasaki *et al.*, 2015 [33] (Amasaki15)	The authors propose a combination of attribute selection and relevancy filtering. First, the attributes are selected such that no attributes are remaining, whose value is not closest to any metric value. Then, the instances are filtered using the same principal, such that no instance is remaining, which is not the closest to any other instance. All this is performed on log-transformed data.	As is.
Ryu *et al.*, 2015 [34] (Ryu15)	The authors propose a relevancy filter based on the idea of string distances, which basically works on the number of metric values that are different. All entities that are not in the neighborhood of a target instance are removed from the training data. Before the relevancy filtering, the authors remove outliers using Mahalanobis distance [35]. Moreover, the authors use the LASER classification scheme [36], which wraps a machine learning classifier such that it first checks if a classification can be done using the immediate neighborhood of an instance, before applying a machine learning classifier.	As is.
Nam and Kim, 2015 [37] (Nam15)	The authors propose a fully automated unsupervised approach for defect prediction called CLAMI, which we included because it could render the need for cross-project data moot. CLAMI consists of two parts: Clustering and LAbeling (CLA) for the labeling of the training data using the metric data by counting how many attribute values are above the median for the attribute and Metric and Instances selection (MI) for the selection of a subset of attributes and instances that is consistent, i.e., such that all no attributes of any instance violate the labeling scheme.	As is.

of using ANOVA and Scott-Knott are the assumptions of ANOVA: the normality of the residuals of the distribution and homoscedasticity of the data. In comparison, Friedman and post-hoc Nemenyi test are non-parametric.

All three benchmarks have in common, that they base their rankings on single performance metrics, e.g., the *AUC* and on a single data set. In comparison, we consider a ranking using multiple metrics and multiple data sets within our benchmark.

4 Benchmark Methodology

Within this section, we describe the methodology that we followed to perform our benchmark. We will first formulate the research questions which we answer using this benchmark. Then, we describe the data sets that we use as foundation for our analysis. Afterwards, we discuss which machine learning classifiers were used for the predictions and how we established performance baselines. Furthermore, we describe the evaluation strategy for each of the research questions. Finally, we fix the scope of our benchmark through some additional remark regarding the limitations due to our replication of approaches.

4.1 Research Questions

With our benchmark, we want to answer the following five research questions:

- **RQ1:** Which CPDP approaches perform best in terms of *F-measure*, *G-measure*, *AUC*, and *MCC*?

TABLE 4
Related work on CPDP that was not included in the benchmark (Table 1/2).

Publication	**Short Description**	**Reason for Exclusion**
Briand *et al.*, 2002 [38]	An analysis if a prediction model trained for one project is applicable to another project.	No specific approach for CPDP proposed.
Nagappan *et al.*, 2006 [39]	A general analysis of the suitability of cross-project data for defect prediction. They determined that CPDP is possible with data from similar projects.	No specific approach for CPDP proposed.
Jureczko and Madeyski, 2010 [40]	Relevancy filtering of products using self-organizing maps.	Excluded because the clustering approach allows for products to not be clustered at all, hence, the approach would only be applicable to a subset of the data.
Turhan *et al.*, 2011 [5]	Augmentation of within-project training data with cross-project data by using a *k*-NN relevancy filter for the cross-project data to create a MPDP model.	MPDP is out of scope.
Rahman *et al.*, 2012 [41]	The authors propose the usage of the metric *AUCEC*, a cost-sensitive variant of *AUC* that takes the LOC that are reviewed into account.	No specific approach for CPDP proposed.
Turhan, 2012 [42]	An overview on the problems due to data set shift, i.e., differences between training and target products and a taxonomy how such problems can be handled.	No specific approach for CPDP proposed.
Z. He *et al.*, 2012 [43]	The authors study how well groups of three products are suited for CPDP of another product. Through a brute-force search, they determine the best case performance, i.e., the best set of three products for the training of another product. They use the results of this brute force search to define a decision tree for the selection of the best group of three products as an adoption of the decision tree approach proposed by Zimmermann *et al.* [9].	The decision tree approach proposed is an adoption of the decision tree proposed by Zimmermann *et al.* [9] and, thereby, already covered in the benchmark.
Peters *et al.*, 2013b [44]	A relevancy filter based on the *k*-NN algorithm that, in comparison to the work by Turhan *et al.* [8], selects the instances based on the neighborhoods of the training data instead of the target data.	The publication was withdrawn (http://de.slideshare.net/timmenzies/msr13-mistake (last checked: 2016-07-20)) and no new evidence was provided afterwards that the approach is working.
Kocaguneli *et al.*, 2013 [45]	A general discussion of the cross-project problem for defect prediction and effort prediction.	No specific approach for CPDP is proposed.
Turhan *et al.*, 2013 [46]	An extension of their previous work on MPDP [46], that focuses on the impact of using different quantities of within-project data.	MPDP is out of scope.
Singh *et al.*, 2013 [47]	Experiments on pair-wise CPDP without any specific approach.	No specific approach for CPDP is proposed.
F. Zhang *et al.*, 2014 [48] and F. Zhang *et al.*, 2015a [31]	The authors propose an approach for the transformation of training data based on the clustering of context factors. First, the software is clustered using the context factors, and in a second step, the metric values are transformed to the values 1, 2, ..., 10 depending on the decile they fall in within the cluster.	In comparison to Zimmermann *et al.* [9], who use context factors in addition to software metrics, approaches based on only context factors are out of scope.
Fukushima *et al.*, 2014 [49] and Kamei *et al.*, 2015 [50]	The authors discuss the application of CPDP in a Just In Time (JIT) setting, i.e., to predict defective changes. To this aim, the authors propose a relevancy filter based on the correlation of metrics between software products to use only the products with the strongest correlations for training. Additionally, the authors also consider to train bagging predicts, same as Khoshgoftaar *et al.* [6] and Z. He *et al.* [21].	JIT defect prediction is out of scope.
Mizuno and Hirata [51], 2014	The authors propose a text-based approach for defect prediction based on tokenizing the source code and differentiating between comments and code, and reducing the code to its basic structure.	Text classification based on the source code is out of scope.
Chen *et al.*, 2015 [52]	The authors propose an approach called Double Transfer Boosting, a boosting variant for MPDP that first selects similar data using *k*-NN relevancy filtering [8], then applies SMOTE for oversampling and uses data weighting after Ma *et al.* [14]. The boosting works with two data sets: a small within-project data set and a larger cross-project data set. The within-project data is favored during boosting with the intent to tailor the outcome to the target domain.	MPDP is out of scope.
Ryu *et al.*, 2015a [53]	The authors propose a MPDP approach for boosting of classifiers that internally uses some within-project data together with the cross-project data. The approach is based on their earlier work [24]. The major difference is that they allow for other classifiers than the Support Vector Machine (SVM) and use within-project data together with cross-project data.	MPDP is out of scope.

TABLE 5
Related work on CPDP that was not included in the benchmark (Table 2/2).

Publication	Short Description	Reason for Exclusion
Nam and Kim, 2015a [54]	The authors propose an approach to use data with different software metrics for CPDP. The approach is based on a pair-wise correlation analysis between all metrics of the training and target data. Using a correlation threshold to select candidate matches, the authors apply a bipartite matching algorithm to determine a set of highly correlated metric matches between the training and target data, in which all metrics occur only once. The matches are then used to train a classifier.	Data sources with different metric sets are out of scope.
Jing *et al.*, 2015 [55]	The authors propose an approach to use data with different metrics for CPDP. To this aim, they define the Unified Metric Representation (UMR), a combination of the metric sets of different domains, where all missing metric values are replaced with zeros. Then, the authors apply Canonical Correlation Analysis (CCA) to determine a transformation of the UMR that maximizes the correlation between the training and target data in order to create homogeneous training data.	Data sources with different metric sets are out of scope.
Cao *et al.*, 2015 [56]	The authors propose an approach where they combine outlier removal using the interquartile distances with TCA [22] to transform the training and target data such that they are homogeneous. Then, they propose to train a neural network with a data weighting strategy that takes a potential bias in the data into account.	The level of detail provided in the paper was not sufficient for replication because the specifics of the neural network definition, e.g., the layer structure, were missing.
Jureczko and Madeyski, 2015 [57]	The authors analyze how well products from different sources (open source, academic, proprietary) are suited for the prediction of defects from the other sources. The focus on the study is on which metrics are most important, depending on the data.	No specific approach for CPDP proposed.
Herbold, 2015 [3]	The authors proposed a benchmarking tool for CPDP built around WEKA [58].	No specific approach for CPDP proposed.
Altinger *et al.*, 2015 [59]	The authors investigated the feasibility of CPDP in the automotive domains and discovered through a correlation analysis and Principle Component Analysis (PCA) that fundamental assumptions required for defect prediction based on code metrics are not fulfilled, i.e., that there is an overlap between the defective regions in the training and target data.	No specific approach for CPDP proposed.

- **RQ2:** Does any CPDP approach consistently fulfill the performance criteria for successful predictions postulated by Zimmermann *et al.* [9], i.e., have at least 0.75 *recall*, 0.75 *precision*, and 0.75 *accuracy*?
- **RQ3:** What is the impact of using only larger products (> 100 instances) with a certain balance (at least 5% defective instances and at least 5% non-defective instances) on the benchmark results?
- **RQ4:** What is the impact of using a relatively small subset of a larger data set on the benchmark results?

The reason for **RQ1** is that in the current state of conducting CPDP experiments [2], all these performance metrics are frequently used and researchers have good arguments for and against any of these metrics. Since we are not aware of any conclusive argument with which we could decide for only a single of those four metrics, we instead decided to go the opposite way: determine the performance in terms of all the metrics and, thereby, combine the advantages of the different metrics. This gives a more holistic view on the performance of the CPDP approaches. The reason for **RQ2** is that we want to investigate how far CPDP has come. One popular result from their work from 2009 was, that Zimmermann *et al.* [9] found that less than three percent of CPDP prediction achieve this desired threshold. By revisiting this threshold, we want to determine if and how well this threshold performance is achieved by the current state of the art.

Subsetting by either only filtering out few projects (e.g., [43], [21], [26], [20], [33]) or working with a relatively small selection of products are both common in the defect prediction literature (e.g., [12], [13], [15], [17], [18], [19], [23], [28], [31], [54]). The analysis of **RQ3** and **RQ4** allows us to gain insights into how subsetting affects the overall performance determined by defect prediction experiments. There are good arguments both for increasing or decreasing performance due to subsetting. For example, a subset means less data for training, which may mean less generalization and, therefore, worse performance. On the other hand, a subset has less variance in the data, which could increase the performance.

4.2 Data

Our benchmark evaluates CPDP approaches on five publicly available data sets. We give an overview of the data sets, including the number of products and number of metrics. The complete list of metrics, products, and information about the number of instances and defect prone instances can be found in the appendix. Please note that the benchmarking of techniques, that combine data from different data sets (e.g., [54], [55]) is out of scope of this benchmark. Therefore, we describe each data set on its own.

4.2.1 JURECZKO / FILTERJURECZKO / SELECTEDJURECZKO

The first data set was donated by Jureczko and Madeyski [40].[2] The complete data set consists of 48 product releases of 15 open source projects, 27 product releases of six proprietary projects and 17 academic products that were implemented by students, i.e., 92 released products in total. As metrics, they collected 20 static product metrics for Java classes, as well as the number of defects that were found in each class. The defect labels are extracted from the SourceCode Management system (SCM) using a regular expression. We do not use the proprietary products in our benchmark to avoid mixing closed and open source data, which would add a potential threat to the validity of our results. Moreover, three of the academic products contain less than five defective instances, which is too few for reasonable analysis with machine learning. Hence, we use 62 open source and academic products, to which we refer as JURECZKO in the following. Additionally, we consider a second instance of this data set to which we refer to as FILTERJURECZKO for which we require at least 100 instances for each product and at least 5% of the data to be defective and at least 5% of the data to be non-defective. The FILTERJURECZKO data contains 39 open source products and two academic products. We use the FILTERJURECZKO data to evaluate the impact of such a filtering of products on the mean results in order to answer **RQ3**. The criteria are based on the filtering performed in the literature: Herbold [20] and Amasaki *et al.* [33] use only products with at least 100 instances, the selected projects by Z. He *et al.* [43], [21] and P. He *et al.* [26] also only contain larger products with at least 100 instances and only products with at least 5% of the data within each class. Furthermore, we use with SELECTEDJURECZKO a third combination of products from the overall data. The SELECTEDJURECZKO data contains 10 handpicked products from different open source projects. The reason for using the SELECTEDJURECZKO data is that using such a small subset is quite common in defect prediction literature (e.g., [12], [13], [15], [17], [18], [19], [23], [28], [31], [54]). Within this work, we use the same subset as it is used in a recent publication by Peters *et al.* [28]. The comparison of this subset with the JURECZKO data will allow us to estimate the impact on the mean results of using a smaller subset, that allows better insights into single results in order to answer **RQ4**.

2. The data is publicly available online: http://openscience.us/repo/defect/ck/ (last checked: 2017-05-19)

4.2.2 MDP

The second data set is the preprocessed version of the NASA Metrics Data Program (MDP) data provided by Shepperd *et al.* [67].[3] The data contains information about 12 products from 6 projects. The reason why we use the preprocessed version by Shepperd *et al.* is that Gray *et al.* [68] noted problems with the consistency of the originally published MDP data, which Shepperd *et al.* resolved. The projects in the data sets share 17 static source code metrics. Information on how the defect labels were created is not available. Within our benchmark, we use all 12 products from this data set and refer to this data set in the following as MDP.

4.2.3 AEEEM

The third data set was published by D'Ambros *et al.* [69][4] and contains data about five Java products from different projects. For all five products, 71 software metrics are available, including static product metrics, process metrics like the number of defects in previous releases, the entropy of code changes, and source code churn, as well as the weighted churn and of source code metrics (WCHU) and linearly decayed entropy of source code metrics (LDHH). The study by D'Ambros *et al.* concluded that using these variants of metrics yields the best performance.[5] The defect labels were extracted from the Issue Tracking System (ITS) of the projects. We use all five products within our benchmark and refer to this data in the following as AEEEM.[6]

4.2.4 NETGENE

The fourth data set was published by Herzig *et al.* [71][7] and contains data about four open source projects that follow strict and industry like development processes. The data contains a total of 465 metrics, including static product metrics, network metrics, as well as genealogy metrics, i.e., metrics related to the history of a file, e.g., the number of authors or the average time between changes. The defect labels are determined following the approach suggest by Zimmermann *et al.* [72]. We use all four products within our benchmark and refer to this data in the following as NETGENE.

3. The data is publicly available online: http://openscience.us/repo/defect/mccabehalsted/ (last checked: 2017-05-19)

4. The data is publicly available online: http://bug.inf.usi.ch/ (last checked: 2017-05-19)

5. Internal experiments confirmed this hypothesis for this benchmark. The results achieved with the AEEEM data without LDHH and WCHU, with only WCHU, and with only LDHH are part of the raw result data provided with our replication kit [70].

6. With this name we take pattern from Nam *et al.* [22], using the first letters of the products within the data set.

7. The data is publicly available online: https://hg.st.cs.uni-saarland.de/projects/cg_data_sets/repository (last (last checked: 2017-05-19)

4.2.5 RELINK

The fifth data set was published by Wu *et al.* [73][8] and contains defect information about three products from different projects. The data contains 60 static product metrics and three different defect labels for each module: 1) golden, with manually verified and not automatically labeled defect labels; 2) relink, with defect labels generated with their proposed approach; and 3) traditional heuristic, with an SCM comment based labeling. Within this benchmark, we use all three products with the golden labeling and refer to this data in the following as RELINK.

4.3 Classifiers and Baselines

Within our benchmark, we apply all 24 identified approaches (see in tables 1–3) to all data sets. Six of the approaches define a classification scheme, which we directly adopt:

- genetic program (GP) for Liu10;
- logistic ensemble (LE) for Uchigaki12;
- MODEP for Canfora13;
- CODEP with Logistic Regression (CODEP-LR) and CODEP with a Bayesian Network (CODEP-BN) for Panichella14;
- the boosted SVM (VCBSVM) for Ryu14; and
- average voting (AVGVOTE), maximum voting (MAXVOTE), bagging with a C4.5 Decision Tree (BAG-DT), bagging with Naïve Bayes (BAG-NB), boosting with a C4.5 Decision Tree (BOOST-DT), boosting with Naïve Bayes (BOOST-NB) for YZhang15.

For the 17 approaches Koshgoftaar08, Watanabe08, Turhan09, Zimmermann09, CamargoCruz09, Ma12, Peters12, Peters13, Herbold13, ZHe13, Nam13, PHe15, Peters15, Kawata15, Amasaki15, Ryu15, and Nam15 we use six classifiers in our benchmark. Table 6 lists the six classifiers, including a brief description and the reason for the selection. We used these classifiers together with all approaches, that did not propose a classifier, but a treatment for the data or something similar, i.e., the 17 listed above.

For Menzies11, we use the six classifiers from Table 6 and additionally the WHICH algorithm, that was used in the original publications [12], [13].

Additionally, we use four baselines in the benchmark listed in Table 7, including a brief description and the reason for the selection.

We repeat all approaches that contain a random component 10 times and use the mean value. These approaches are Liu10 and Canfora13 because of the genetic programs, Menzies11 due to the WHERE clustering, Peters12 and Peters13 because of MORPH, ZHe13 because of the undersampling, Ryu14 because of the internal sampling, Peters15 because of the ordering of products for building the cache and MORPH, and the baseline RANDOM.

Hence, we get a total of 450 results for each product we use in the benchmark due to 15×6=90 (six classifiers without repetitions including the baselines ALL and CV) + 4×6×10=240 (six classifiers with repetition) + 7×10=70 (Menzies11) + 10 (proposed classifiers without repetition and the baseline FIX) + 4×10=40 (proposed classifiers with repetition and the baseline RANDOM).

Unfortunately, we could not apply all approaches to all data sets, due to the required resource consumption or run time. We only include work in the benchmark that could be executed on a machine with 32 GigaByte (GB) Random Access Memory (RAM) and where the calculation of a single result required less than one day. The reason for this time restriction is two-fold. First, we need to consider this benchmark itself. The execution time of our experiment would not scale if we allowed longer runtimes. With a runtime of one day per project, the calculation of all results for the JURZECZKO data alone already requires 62 days for each classifier. With six classifiers, this increases two 372 days. Even if this is scaled with ten parallel processes, this still requires over 23 days. For one week of runtime per result, this goes up to over 162 days. The second reason is the practical applicability of the approach. We cannot assume that infinite time and memory is available for the task of defect prediction. The problem with CPDP is that the defect prediction model should be retrained every time the data changes as this could mean changes to the prediction model, i.e., the new model should be trained every time the software product, for which defects shall be predicted, changes. As a consequence, if one wanted to integrated CPDP into a nightly build cycle, even a runtime of one day would already be infeasible.

The following results, could not be obtained.

- We could not apply the approach Nam13 to the JURECZKO, FILTERJURECZKO, MDP, AEEEM,[9] and NETGENE. The reason for this is that the Transfer Component Analysis (TCA) required for their approach requires solving an eigenvalue problem over a non-symmetric non-sparse $n \times n$-matrix, with n the number of training instances. Even storing this matrix does not work for the MDP and NETGENE data with 32 GB RAM, for the JURECZKO, FILTERJURECZKO, and AEEEM the matrix fits within the memory, but solving the eigenvalue problem cannot be done in less than one day. Since the SELECTEDJURECZKO data is only used for comparison with the JURECZKO

8. The data is publicly available online: http://www.cse.ust.hk/~scc/ReLink.htm (last checked: 2017-05-19)

9. We realize that AEEEM was used in the original publication by Nam *et al.* [22]. However, in comparison to our work, Nam *et al.* only used 9 out of the 71 metrics we use in our work, which allowed solving the eigenvalue problem in much less time.

data, we also do not calculate the results for that data set.

- We could not apply the approach ZHe13 to the NETGENE data. The calculation of the separability between the products requires more than one day, due to the very high dimension of the data.

Hence, we have only 444 results for the products from the JURECZKO, FILTERJURECZKO, and MDP data sets, and 384 results for the products from the NETGENE data.

Thus we have

- 62×444=27,528 results for the JURECZKO data,
- 41×444=18,204 for the FILTERJURECZKO data,
- 10×444=4,440 for the SELECTEDJURECZKO data,
- 12×444=5,328 results for the MDP data,
- 5×444=2,220 results for the AEEEM data,
- 4×384=1,536 results for the NETGENE data, and
- 3×450=1,350 results for the RELINK data.

This gives us a total of 60,606 results[10].

4.4 Evaluation Strategy

For the evaluation of *RQ1*, we took pattern from the existing benchmarks on WPDP [60], [64], [65]. Our results did not hold the assumptions for ANOVA. The distribution of the performance metrics fulfilled neither the normality of the residuals assumption, nor homoscedasticity assumption. Thus, we could not use the ANOVA/Scott-Knott approach suggested by Ghotra *et al.* [65]. Therefore, we took pattern from Lessmann *et al.* [60] and D'Ambros *et al.* [64].

Hence, we adopted the guidelines by Demšar [61] for the statistical comparison of classifiers over multiple data sets. Demšar suggest the usage of a Friedman test [62] with a post-hoc Nemenyi test [63]. The Friedman test is a non-parametric statistical test which determines if there are statistically significant differences between three or more populations. In case the Friedman test determines that there are statistically significant differences between the populations, the post-hoc Nemenyi test can be applied to compare the different populations with each other. The Nemenyi test uses the concept of Critical Distances (CDs) between average ranks to define significant different populations. If the distance between two average ranks is greater than the CD, the two populations are significantly different. This concept can be visualized using CD diagrams, as shown in Figure 2.

10. The replication kit additional contains a total of 67,326 results. The additional results are the following. 2,250 results for AEEEM without the LDHH and WCHU metrics, 2,250 for AEEEM with only the LDHH metrics, and 2,220 for AEEEM with only WCHU metrics. The difference in results between the AEEEM metric combinations is due to the fact that Nam13 terminated in time for AEEEM without the LDHH and WCHU metrics and for AEEEM with only the LDHH metrics, but not for AEEEM with the WCHU metrics and AEEEM with the LDHH and WCHU metrics.

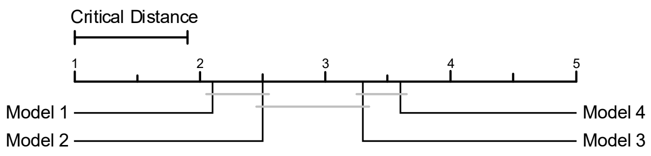

Fig. 2. Example for a critical distance after Demšar [61]. Models connected by gray lines are not significantly different.

In a normal CD diagram, the horizontal lines define all possible groups of non-significantly different populations, i.e., all populations whose average ranks are within the critical distance. As Figure 2 highlights, these groupings may be overlapping. Model 2 is both non-significantly different from Model 1 and Model 3. However, Model 1 and Model 3 are significantly different. This means the post-hoc Nemenyi test has one clear drawback: it does not create distinct ranks for the populations that are compared, but overlapping ranks. Under the assumption, that the distance between the best average ranks and the worst average ranks is greater than two time the CD, we can define three non-overlapping groups of populations:

- The populations that are within the CD of the best average ranking population (top rank 1).
- The populations that are within the CD of the worst average ranking population (bottom rank 3).
- The populations that are neither (middle rank 2).

Using these three groups, we get non-overlapping statistically signficantly different ranks from the results of the post-hoc Nemenyi test.

In the sense of Demšar's work, a data set is the counterpart to a software product in our benchmark. Moreover, Demšar does not take multiple performance metrics into account. Thus, in our case we actually have multiple sets of populations, to each of which we apply the Friedman test with the post-hoc Nemenyi test, i.e., for each data set (e.g., JURECZKO) and each metric (e.g., *AUC*). Thus, we get 20 rankings of the CPDP approaches (four metrics used for **RQ1** times five data sets).

The trivial approach would be to use the mean rank over the 20 rankings to get an overall ranking. However, this approach does not work since the number of approaches that are within a certain rank affects the results. For example, it makes a difference, if you are within the middle ranked cluster and there are only two results in the best ranked cluster, or if there are 20 results in the best ranked cluster.

In order to deal with this effect, we introduce the concept of the *rankscore* defined as one minus the percentage of approaches that is ranked higher, i.e.,

$$rankscore = 1 - \frac{\#\{\text{approaches ranked higher}\}}{\#\{\text{approaches}\} - 1}.$$

TABLE 6
Classifiers used in the benchmark.

Classifier (Abbreviation)	**Short Description**	**Reason for selection**
C4.5 Decision Tree (DT)	A tree structure where a logical decision based on a single metric is made within each node [74]. The leafs of the tree define the classification. Which attribute is used for the decision at the node is decided by the information gain.	Interpretable classifier, popular in the defect prediction literature.
Logistic Regression (LR)	A linear regression model of the logit function to estimate the likelihood of a classification [75]. For classification, logistic regression chooses the class with the highest likelihood.	Often successfully used for defect prediction and allows not only binary classification but also ranking of results.
Naïve Bayes (NB)	Estimates a score for each class based on a simplification of Bayes law [76].	Popular in the defect prediction literature.
Random Forest (RF)	A forest of random trees [77]. A random tree is a decision tree, for which the attributes over which the decisions are made are selected randomly. Hence, the complete structure of the tree is random.	One of the most powerful classification algorithms according to the machine learning literature [78], [6].
RBF Network (NET)	A type of artificial neural network with Radial Basis Functions (RBFs) as neurons [78], [79].	One of the most powerful classification algorithms according to the machine learning literature [78].
Support Vector Machine (SVM)	An optimization problem solver that determines a hyperplane that separates the positive from the negative samples. The hyperplane is determined in the kernel space of the data, i.e., a transformation of the data in a higher dimensional space using a kernel function. As kernel function, we use RBFs.	One of the most powerful classification algorithms according to the machine learning literature [78], [80].

TABLE 7
Baselines used in the benchmark.

Baseline	**Short Description**	**Reason for selection**
ALL	All products from the same data set except from the same product are used for training without any data treatment. We train a classifier with all data for all classifiers listed in Table 6.	The simplest possible approach towards CPDP: just take all data from a repository and let the classifier handle the rest.
CV	10x10 cross-validation for each product to build an WPDP baseline. We perform cross-validation with all classifiers listed in Table 6.	A comparison with WPDP. However, we note that cross-validation overestimates the performance, as determined by Tan *et al.* [81] who observed double-digit differences between cross-validation and real WPDP with only historic data and no data from the same revision.
RANDOM	Randomly classifies instances as defective with a probability of 0.5.	A trivial classification model which all approaches must beat. Random classification is also often used in machine learning research as a minimal criterion that approaches must outperform.
FIX	Classifies all entities as defective	Considering all entities as potentially defective is the null hypothesis that testers should work with, if no defect prediction model is available. Hence, all defect prediction models should be compared against this approach to ensure that they actually provide a practical gain.

The *rankscore* is within the interval $[0, 1]$, where one is a perfect ranking, if no approach is in a better cluster, and zero the worst ranking that is achieved if all other approaches are ranked better. Hence, the *rankscore* is not sensitive to the number of clusters and it creates a relative ranking of approaches for each metric. To create the overall ranking, we use the mean *rankscore* over all performance metrics and data sets. Hence, we rank approaches best if they perform well for all performance metrics on all data sets, failures to predict accurately measured using one of the four metrics on any of the five data sets will lead to lower rankings.

For **RQ2**, we consider the number of products that fulfill the success criterion of 0.75 *recall*, 0.75 *precision*, and 0.75 *accuracy*. For **RQ3**, we compare the results of the JURECZKO data to the results of the FILTERJURECZKO data. To this aim, we determine the difference in the mean performance between the two data sets for all results and consider the mean difference and the standard deviation. Moreover, we perform a Mann-Whitney-U test [82] to determine if the difference is statistically significantly different. For **RQ4**, we compare the results of the JURECZKO data to the SELECTEDJURECZKO data following the same approach as for **RQ3**.

For both Friedman test with post-hoc Nemenyi test and the Mann-Whitney-U test we use a significance threshold of $p < 0.05$.

4.5 Additional Remarks

Since the focus of our benchmark is on replication of proposed approaches and not the advancement or improvement of existing approaches, we tried to implement and execute the benchmark with parameters as close as possible to the original setups. Specifically, this means three things:

- We did not perform any treatment for the class imbalance problem, unless it was also done in the original work. For example, ZHe13 proposed to use undersampling, so we also used undersampling for that approach in our benchmark.
- We did not perform any skew treatment for variable distributions, unless it was also done in the original work. For example, Turhan09 proposed logarithm transformations in their work, so the metric data was also transformed in our replication of Turhan09.
- We did not perform any threshold optimization for threshold-sensitive classifiers. The reason for this is that none of related work performed such an optimization.[11]

We are aware that this decision to not perform these tasks in general may mean that classifiers underperform, as they are sensitive to such effects. For example, SVMs are very sensitive to the class imbalance problem, because they are often not well-calibrated. However, we deem the threat to the validity of this benchmark if we were to modify the approaches by, e.g., performing additional variable transformation to treat skew or additional data selection to treat the class imbalance is greater. In that case, it would not be clear if a good or bad performance is due to our modification, or due to the original approach.

The threshold optimization would only affect the results achieved with the LR model for classification, the other classifiers are not threshold-sensitive. Since the selection of the threshold must be done on the training data and none of the CPDP approaches proposes such a training data based threshold selection approach, we did not apply it either. A posteriory selection of the best possible threshold on the target data is not possible in practice, and was therefore not considered as an alternative.

5 RESULTS

Within this section, we present the results of our benchmark for each research question. We are aware that due to the scope of our benchmark and the data we collected, different research questions can also be addressed with the same data, e.g., the performance not over all metrics and data sets, but also for only a single metric, each data set on its own, etc. To facilitate such further insights, we provide a replication kit [70]. The replication kit contains

11. The only exception to this is to some degree Canfora13. However, they pick the optimal threshold for a given predefined performance target. We do not have such a performance target in our general benchmark.

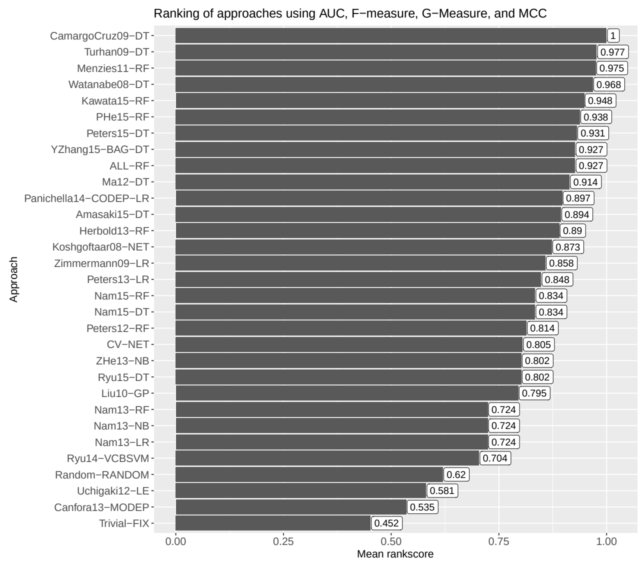

Fig. 3. Mean rank score over all data sets for the metrics *AUC*, *F-measure*, *G-measure*, and *MCC*. In case multiple classifiers were used (e.g., those listed in Table 6), we list only the result achieved with the best classifier.

- all source code used for the collection and evaluation of the results;
- a MySQL database dump with all raw results including additional metrics and the confusion matrix; and
- additional visualizations of the results, including box plots and CD diagrams for the single performance metrics and data sets.

5.1 RQ1: Which CPDP approaches perform best in terms of F-measure, G-measure, AUC, and MCC?

Figure 3 shows the mean *rankscore* for the four metrics and five data sets for all approaches. Table 8 shows detailed results including the mean values and rankings for each performance metric and each data set. Due to the number of results, we only list the best classifier for each approach. For example, we only list NB in case it outperformed DT, SVM, LR, etc.

The first thing we note for our result is that there are often many approaches ranked first, meaning that there are no statistically significant differences we could determine between many approaches. This is highlighted in Table 8 through bold-facing all top-ranking approaches for a single data set and performance metric according to the Friedmann test with post-hoc Nemenyi test. Still, there are some differences between the results.

CamargoCruz09-DT has the best mean *rankscore* with 1.00 (best possible score), followed by Turhan09-DT with 0.977, Menzies11-RF with 0.975, and Watanabe08-DT with 0.968. The difference between CamargoCruz09 and the three followers is very small.

TABLE 8
Mean results over all products with rankscores in brackets. Bold-faced values are top-ranking for the metric on the data set. For FILTERJURECZKO and SELECTEDJURECKO, we show the difference in the mean values to JURECZKO.

	JURECZKO				FILTERJURECZKO / SELECTEDJURECZKO				MDP			
	AUC	*F-measure*	*G-measure*	*MCC*	*AUC*	*F-measure*	*G-measure*	*MCC*	*AUC*	*F-measure*	*G-measure*	*MCC*
ALL-RF	**0.66 (1)**	**0.32 (1)**	**0.43 (1)**	**0.17 (1)**	-0.01 / 0.01	-0.01 / 0.01	-0.01 / 0.00	-0.02 / 0.01	**0.66 (1)**	**0.14 (1)**	**0.17 (1)**	**0.12 (1)**
Amasaki15-DT	**0.6 (1)**	**0.38 (1)**	**0.49 (1)**	**0.2 (1)**	0.00 / -0.01	-0.01 / 0.02	0.00 / 0.03	-0.02 / -0.01	**0.71 (1)**	**0.1 (1)**	**0.14 (1)**	**0.06 (1)**
CamargoCruz09-DT	**0.58 (1)**	**0.37 (1)**	**0.5 (1)**	**0.18 (1)**	0.00 / 0.00	0.00 / 0.04	0.01 / 0.03	-0.03 / 0.02	**0.7 (1)**	**0.09 (1)**	**0.12 (1)**	**0.09 (1)**
Canfora13-MODEP	0.52 (0.49)	**0.44 (1)**	**0.48 (1)**	**0.19 (1)**	0.01 / 0.03	0.04 / 0.09	0.04 / 0.14	0.00 / 0.08	0.5 (0.27)	**0.06 (1)**	**0.04 (1)**	-0.01 (0.48)
CV-NET	0.71 (0.49)	**0.49 (1)**	**0.46 (1)**	0.29 (0.37)	0.02 / 0.05	-0.02 / 0.00	-0.02 / 0.04	0.00 / 0.05	0.75 (0.58)	**0.18 (1)**	**0.19 (1)**	**0.18 (1)**
Herbold13-RF	**0.64 (1)**	**0.39 (1)**	**0.5 (1)**	**0.17 (1)**	0.00 / 0.03	-0.01 / 0.00	-0.01 / 0.01	-0.01 / 0.03	**0.66 (1)**	**0.13 (1)**	**0.21 (1)**	**0.09 (1)**
Kawata15-RF	**0.65 (1)**	**0.32 (1)**	**0.43 (1)**	**0.17 (1)**	-0.02 / 0.03	-0.01 / 0.03	0.00 / 0.02	-0.02 / 0.05	**0.68 (1)**	**0.11 (1)**	**0.14 (1)**	**0.09 (1)**
Koshgoftaar08-NET	**0.6 (1)**	**0.32 (1)**	**0.4 (1)**	0.23 (0.37)	-0.01 / 0.01	-0.02 / 0.05	-0.03 / 0.03	-0.01 / 0.04	0.51 (0.58)	0.05 (0.51)	0.05 (0.52)	**0.06 (1)**
Liu10-GP	**0.63 (1)**	0.51 (0.44)	**0.52 (1)**	**0.23 (1)**	-0.05 / -0.08	-0.02 / -0.02	-0.13 / -0.29	-0.07 / -0.09	**0.65 (1)**	0.27 (0.51)	0.52 (0.52)	0.17 (0.48)
Ma12-DT	**0.6 (1)**	**0.37 (1)**	**0.49 (1)**	**0.18 (1)**	-0.01 / 0.00	-0.01 / 0.03	0.00 / 0.04	0.00 / 0.03	**0.61 (1)**	**0.16 (1)**	**0.25 (1)**	**0.09 (1)**
Menzies11-RF	0.58 (0.49)	**0.32 (1)**	**0.43 (1)**	**0.15 (1)**	-0.01 / 0.01	-0.01 / 0.03	-0.01 / 0.02	-0.02 / 0.03	**0.54 (1)**	**0.14 (1)**	**0.2 (1)**	**0.11 (1)**
Nam13-NB	-	-	-	-	-	-	-	-	-	-	-	-
Nam15-DT	**0.66 (1)**	0.51 (0.44)	0.63 (0.42)	0.29 (0.37)	-0.02 / -0.01	-0.03 / 0.01	-0.02 / 0.01	-0.05 / -0.01	**0.62 (1)**	0.26 (0.51)	**0.35 (1)**	**0.13 (1)**
Nam15-RF	**0.66 (1)**	0.51 (0.44)	0.63 (0.42)	0.29 (0.37)	-0.02 / -0.01	-0.03 / 0.01	-0.02 / 0.01	-0.05 / -0.01	**0.62 (1)**	0.26 (0.51)	**0.35 (1)**	**0.13 (1)**
Panichella14-CODEP-LR	**0.59 (1)**	**0.3 (1)**	**0.38 (1)**	**0.2 (1)**	0.02 / 0.04	0.04 / 0.1	0.06 / 0.12	0.02 / 0.05	0.51 (0.58)	**0.06 (1)**	**0.06 (1)**	**0.08 (1)**
Peters12-RF	**0.63 (1)**	**0.3 (1)**	**0.38 (1)**	**0.15 (1)**	0.00 / 0.01	-0.01 / 0.06	-0.01 / 0.07	0.00 / 0.03	**0.61 (1)**	**0.09 (1)**	**0.12 (1)**	**0.07 (1)**
Peters13-LR	0.72 (0.49)	0.17 (0.44)	0.2 (0.42)	**0.16 (1)**	-0.03 / -0.09	0.02 / 0.12	0.02 / 0.14	0.01 / 0.04	0.72 (0.58)	0.04 (0.51)	0.04 (0.52)	**0.05 (1)**
Peters15-DT	**0.57 (1)**	**0.35 (1)**	**0.46 (1)**	**0.15 (1)**	-0.01 / 0.03	0.01 / 0.06	0.01 / 0.07	0.01 / 0.05	0.55 (0.58)	**0.14 (1)**	**0.2 (1)**	**0.09 (1)**
PHe15-RF	**0.64 (1)**	**0.31 (1)**	**0.43 (1)**	0.13 (0.37)	-0.02 / 0.04	0.00 / 0.08	0.01 / 0.06	0.00 / 0.1	**0.63 (1)**	**0.08 (1)**	**0.11 (1)**	**0.07 (1)**
Random-RANDOM	0.5 (0.13)	**0.37 (1)**	**0.49 (1)**	0.00 (0.08)	0.00 / 0.00	0.00 / 0.00	0.01 / 0.01	0.00 / 0.00	0.51 (0.58)	**0.18 (1)**	0.5 (0.52)	0.01 (0.48)
Ryu14-VCBSVM	**0.6 (1)**	0.46 (0.44)	**0.5 (1)**	**0.18 (1)**	-0.01 / 0.03	-0.02 / 0.03	0.02 / 0.09	-0.02 / 0.05	0.56 (0.58)	0.24 (0.51)	**0.22 (1)**	**0.07 (1)**
Ryu15-DT	0.57 (0.49)	**0.29 (1)**	**0.37 (1)**	**0.14 (1)**	0.00 / 0.00	0.02 / 0.03	0.04 / 0.05	-0.01 / 0.00	0.53 (0.58)	**0.1 (1)**	**0.15 (1)**	**0.07 (1)**
Trivial-FIX	0.5 (0.13)	**0.48 (1)**	0.00 (0.13)	0.00 (0.08)	0.00 / 0.00	-0.01 / -0.01	0.00 / 0.00	0.00 / 0.00	0.5 (0.27)	**0.21 (1)**	0.00 (0.52)	0.00 (0.48)
Turhan09-DT	**0.59 (1)**	**0.36 (1)**	**0.47 (1)**	**0.19 (1)**	0.01 / 0.00	0.00 / 0.03	0.01 / 0.04	-0.01 / -0.01	**0.62 (1)**	**0.16 (1)**	**0.24 (1)**	**0.11 (1)**
Uchigaki12-LE	0.74 (0.49)	0.08 (0.44)	0.09 (0.42)	0.1 (0.37)	-0.03 / 0.01	0.01 / 0.11	0.01 / 0.12	0.02 / 0.11	0.77 (0.27)	**0.07 (1)**	**0.08 (1)**	**0.09 (1)**
Watanabe08-DT	**0.59 (1)**	**0.37 (1)**	**0.5 (1)**	0.13 (0.37)	0.00 / 0.03	-0.02 / 0.04	-0.04 / 0.04	-0.02 / 0.07	**0.67 (1)**	**0.11 (1)**	**0.18 (1)**	**0.11 (1)**
YZhang15-BAG-DT	**0.67 (1)**	**0.37 (1)**	**0.48 (1)**	**0.22 (1)**	-0.02 / 0.02	-0.03 / 0.02	-0.02 / 0.03	-0.04 / 0.00	**0.69 (1)**	**0.15 (1)**	**0.19 (1)**	**0.11 (1)**
ZHe13-NB	**0.62 (1)**	0.46 (0.44)	**0.52 (1)**	**0.23 (1)**	-0.01 / 0.02	-0.03 / -0.01	0.02 / 0.05	-0.02 / 0.04	**0.61 (1)**	0.26 (0.51)	0.47 (0.52)	0.18 (0.48)
Zimmermann09-LR	**0.62 (1)**	**0.39 (1)**	**0.45 (1)**	**0.17 (1)**	-0.02 / 0.03	-0.07 / -0.02	-0.1 / 0.00	-0.05 / 0.03	**0.56 (1)**	**0.05 (1)**	**0.07 (1)**	**0.06 (1)**

	AEEEM				NETGENE				RELINK			
	AUC	*F-measure*	*G-measure*	*MCC*	*AUC*	*F-measure*	*G-measure*	*MCC*	*AUC*	*F-measure*	*G-measure*	*MCC*
ALL-RF	**0.71 (1)**	**0.31 (1)**	**0.36 (1)**	**0.28 (1)**	0.71 (0.51)	**0.28 (1)**	**0.32 (1)**	0.28 (0.46)	0.73 (0.57)	**0.53 (1)**	**0.55 (1)**	**0.36 (1)**
Amasaki15-DT	0.55 (0.55)	**0.33 (1)**	**0.47 (1)**	**0.22 (1)**	**0.56 (1)**	**0.28 (1)**	**0.49 (1)**	**0.13 (1)**	**0.65 (1)**	0.57 (0.46)	0.57 (0.43)	0.39 (0.44)
CamargoCruz09-DT	**0.58 (1)**	**0.35 (1)**	**0.46 (1)**	**0.24 (1)**	**0.53 (1)**	**0.27 (1)**	**0.44 (1)**	**0.15 (1)**	**0.65 (1)**	**0.54 (1)**	**0.61 (1)**	**0.31 (1)**
Canfora13-MODEP	0.49 (0.12)	0.16 (0.43)	0.18 (0.48)	0.00 (0.07)	0.5 (0.51)	0.00 (0.39)	0.00 (0.1)	0.00 (0.46)	0.5 (0.57)	0.15 (0.46)	0.1 (0.43)	0.00 (0.44)
CV-NET	0.78 (0.12)	**0.35 (1)**	**0.38 (1)**	**0.31 (1)**	0.74 (0.51)	**0.3 (1)**	**0.34 (1)**	0.27 (0.46)	0.7 (0.57)	**0.53 (1)**	**0.57 (1)**	**0.33 (1)**
Herbold13-RF	**0.7 (1)**	0.37 (0.43)	**0.48 (1)**	**0.25 (1)**	0.73 (0.51)	0.4 (0.39)	**0.5 (1)**	0.34 (0.46)	**0.7 (1)**	**0.44 (1)**	**0.47 (1)**	**0.25 (1)**
Kawata15-RF	**0.69 (1)**	**0.29 (1)**	**0.35 (1)**	**0.24 (1)**	0.68 (0.51)	**0.29 (1)**	**0.34 (1)**	0.27 (0.46)	**0.7 (1)**	**0.45 (1)**	**0.48 (1)**	**0.27 (1)**
Koshgoftaar08-NET	**0.62 (1)**	**0.35 (1)**	**0.42 (1)**	0.31 (0.5)	**0.54 (1)**	**0.25 (1)**	**0.4 (1)**	**0.06 (1)**	**0.66 (1)**	**0.52 (1)**	**0.55 (1)**	**0.34 (1)**
Liu10-GP	**0.6 (1)**	**0.36 (1)**	**0.41 (1)**	0.18 (0.5)	**0.53 (1)**	**0.27 (1)**	**0.18 (1)**	**0.06 (1)**	0.56 (0.57)	0.59 (0.46)	0.29 (0.43)	**0.18 (1)**
Ma12-DT	**0.62 (1)**	**0.32 (1)**	**0.46 (1)**	**0.21 (1)**	**0.64 (1)**	**0.32 (1)**	0.5 (0.37)	**0.19 (1)**	**0.62 (1)**	0.58 (0.46)	**0.57 (1)**	0.34 (0.44)
Menzies11-RF	**0.58 (1)**	**0.29 (1)**	**0.35 (1)**	**0.22 (1)**	**0.57 (1)**	**0.22 (1)**	**0.31 (1)**	**0.16 (1)**	**0.64 (1)**	**0.51 (1)**	**0.56 (1)**	**0.3 (1)**
Nam13-NB	-	-	-	-	-	-	-	-	**0.69 (1)**	0.39 (0.46)	0.4 (0.43)	**0.25 (1)**
Nam15-DT	**0.67 (1)**	**0.41 (1)**	0.64 (0.48)	**0.26 (1)**	**0.57 (1)**	**0.25 (1)**	**0.49 (1)**	**0.12 (1)**	**0.64 (1)**	0.56 (0.46)	**0.5 (1)**	**0.32 (1)**
Nam15-RF	**0.67 (1)**	**0.41 (1)**	0.64 (0.48)	**0.26 (1)**	**0.57 (1)**	**0.25 (1)**	**0.49 (1)**	**0.12 (1)**	**0.64 (1)**	0.56 (0.46)	**0.5 (1)**	**0.32 (1)**
Panichella14-CODEP-LR	**0.63 (1)**	0.36 (0.43)	**0.46 (1)**	0.3 (0.5)	**0.54 (1)**	**0.17 (1)**	**0.27 (1)**	**0.07 (1)**	**0.66 (1)**	**0.51 (1)**	**0.54 (1)**	0.35 (0.44)
Peters12-RF	0.54 (0.55)	0.16 (0.43)	0.2 (0.48)	0.12 (0.5)	**0.56 (1)**	**0.27 (1)**	**0.17 (1)**	**0.03 (1)**	**0.62 (1)**	0.22 (0.46)	0.24 (0.43)	0.11 (0.44)
Peters13-LR	**0.71 (1)**	**0.28 (1)**	**0.36 (1)**	**0.25 (1)**	**0.55 (1)**	**0.24 (1)**	**0.37 (1)**	**0.08 (1)**	**0.65 (1)**	**0.41 (1)**	**0.45 (1)**	**0.27 (1)**
Peters15-DT	0.51 (0.55)	**0.28 (1)**	**0.41 (1)**	0.15 (0.5)	**0.6 (1)**	**0.27 (1)**	**0.38 (1)**	**0.17 (1)**	**0.62 (1)**	**0.55 (1)**	**0.55 (1)**	**0.3 (1)**
PHe15-RF	**0.64 (1)**	**0.29 (1)**	**0.35 (1)**	**0.22 (1)**	**0.58 (1)**	0.15 (0.39)	**0.23 (1)**	**0.08 (1)**	**0.71 (1)**	**0.49 (1)**	**0.5 (1)**	**0.31 (1)**
Random-RANDOM	0.51 (0.55)	0.27 (0.43)	**0.51 (1)**	0.01 (0.07)	0.5 (0.51)	**0.23 (1)**	**0.5 (1)**	0.00 (0.46)	0.5 (0.57)	**0.43 (1)**	**0.49 (1)**	0.00 (0.03)
Ryu14-VCBSVM	0.52 (0.55)	**0.26 (1)**	0.1 (0.48)	0.09 (0.5)	0.5 (0.51)	0.00 (0.39)	0.00 (0.1)	0.00 (0.46)	0.6 (0.57)	**0.54 (1)**	**0.42 (1)**	**0.2 (1)**
Ryu15-DT	0.51 (0.55)	0.07 (0.43)	0.1 (0.48)	0.02 (0.5)	**0.59 (1)**	**0.29 (1)**	**0.43 (1)**	**0.17 (1)**	0.57 (0.57)	**0.34 (1)**	**0.39 (1)**	0.12 (0.44)
Trivial-FIX	0.5 (0.12)	**0.31 (1)**	0.00 (0.08)	0.00 (0.07)	0.5 (0.51)	**0.26 (1)**	0.00 (0.1)	0.00 (0.46)	0.5 (0.57)	**0.56 (1)**	0.00 (0.08)	0.00 (0.44)
Turhan09-DT	0.56 (0.55)	**0.3 (1)**	**0.44 (1)**	**0.2 (1)**	**0.53 (1)**	**0.24 (1)**	**0.44 (1)**	**0.12 (1)**	**0.63 (1)**	**0.53 (1)**	**0.54 (1)**	**0.35 (1)**
Uchigaki12-LE	0.77 (0.12)	0.1 (0.43)	0.1 (0.48)	0.18 (0.5)	0.71 (0.51)	**0.26 (1)**	0.00 (0.1)	0.00 (0.46)	0.75 (0.57)	0.34 (0.46)	**0.34 (1)**	**0.27 (1)**
Watanabe08-DT	**0.57 (1)**	**0.33 (1)**	**0.48 (1)**	**0.23 (1)**	**0.57 (1)**	**0.27 (1)**	**0.45 (1)**	**0.14 (1)**	**0.6 (1)**	**0.49 (1)**	**0.55 (1)**	**0.29 (1)**
YZhang15-BAG-DT	**0.71 (1)**	**0.32 (1)**	**0.41 (1)**	**0.24 (1)**	0.69 (0.51)	**0.33 (1)**	**0.46 (1)**	0.24 (0.46)	0.73 (0.57)	**0.42 (1)**	**0.45 (1)**	**0.29 (1)**
ZHe13-NB	**0.59 (1)**	**0.37 (1)**	**0.3 (1)**	**0.18 (1)**	-	-	-	-	**0.65 (1)**	0.58 (0.46)	0.62 (0.43)	**0.3 (1)**
Zimmermann09-LR	**0.6 (1)**	**0.3 (1)**	**0.37 (1)**	**0.2 (1)**	0.47 (0.51)	0.15 (0.39)	0.15 (0.37)	-0.03 (0.46)	**0.62 (1)**	**0.48 (1)**	**0.43 (1)**	0.16 (0.44)

Each of the three followers in only once not top-ranked, the differences are only due to the fact that a different amount of approaches is part of the top ranks in these cases, leading to variations in the *rankscore*. Overall, ten approaches have a mean *rankscore* of at least 0.9, i.e., are on average in the best 10% of results: the four listed above, as well as Kawata15-RF with 0.948, PHe15-RF with 0.938, Peters15-DT with 0.931, YZhang15-BAT-DT and ALL-RF with 0.927, and Ma12-DT with 0.914. Of the top ten, Kawata15-RF and PHe15-RF are also close to the top and only twice ranked worse than CamargoCruz09-DT, the others three times.

The WPDP with cross validation CV-NET is not the best overall approach, but only ranked in the lower mid-field with a mean rank score of 0.805. This indicates that CPDP can outperform WPDP in this setting, where multiple performance measures and data sets are considered.

The baseline ALL-RF is part of the results which are on average in the best 10% of results. This outperforms 16 of the proposed approaches for all classifiers: Ma12, Panichella14, Amasaki15, Herbold13, Koshgoftaar08, Zimmermann09, Peters13, Nam15, Peters12, ZHe13, Ryu15, Liu10, Nam13, Ryu14, Uchigaki12, and Canfora13. The baselines RANDOM and FIX are at the bottom of the ranking. We note that we made an assumption for the last ranked result, Canfora13-MODEP, i.e., the selection of a classifier with at least 70% *recall*. Another choice here might drastically change the performance of this approach, since it actually yields a family of classifiers.

Answer RQ1: CamargoCruz09-DT performs best among the compared CPDP approaches and even outperforms cross-validation. However, the differences to other approaches are very small, including the difference to the baseline ALL-RF. The baseline ALL-RF is ranked higher than sixteen of the CPDP approaches.

TABLE 9
Approaches and products were any classifier fulfilled the criterion by Zimmermann *et al.* [9]. Italic means only fulfilled by a baseline.

Product	Data Set	Fulfilling approaches
berek	JURECZKO	ALL, Amasaki15, CamargoCruz09, CV, Herbold13, Kawata15, Koshgoftaar08, Ma12, Menzies11, Panichella14, Peters12, Peters13, Peters15, PHe15, Turhan09, YZhang15, Zimmermann09
ckjm	JURECZKO	Herbold13, Liu10, Zimmermann09
log4j-1.2	*JURECZKO*	*CV, FIX*
openintents	RELINK	Amasaki15, YZhang15
pbeans1	JURECZKO	CV, Herbold13, FIX, ZHe13
pdftranslator	JURECZKO	CV, Liu10, Nam15
poi-1.5	*JURECZKO*	*CV*
poi-2.5	*JURECZKO*	*CV*
poi-3.0	JURECZKO	Canfora13, CV, Liu10, Nam15
sklebagd	JURECZKO	CamargoCruz09, CV, Herbold13, Liu10, Nam15, Peters12, Peters15, PHe15, Ryu15, Turhan09, ZHe13
termoproject	JURECZKO	Ryu14
velocity-1.4	*JURECZKO*	*CV, FIX*
velocity-1.5	*JURECZKO*	*CV*
wspomaganiepi	JURECZKO	Canfora13, CV, Herbold13, Liu10, Ma12, Menzies11, Nam15, Peters12, Ryu15, Turhan09, ZHe13, Zimmermann09
xalan-2.7	*JURECZKO*	*CV, FIX*
xerces-1.4	*JURECZKO*	*CV*
zuzel	JURECZKO	Amasaki15, Nam15, Peters15, PHe15, Turhan09, Zimmermann09

5.2 RQ2: Does any CPDP approach consistently fulfill the performance criteria for successful predictions postulated by Zimmermann *et al.* [9], i.e., have at least 0.75 recall, 0.75 precision, and 0.75 accuracy?

Table 9 shows the products and CPCD approaches where any classifier fulfills the criterion by Zimmermann *et al.* [9]. Overall, for only 10 out of 86 products any CPDP approach fulfills the criterion. The baseline CV fulfills the criterion for 13 out of 86 products with any classifier. The CV-RF is the best classifier for CV with 11 out of 86 products, i.e., for 12.79% of the products. Of the CPDP approaches, Turhan09 and Nam15 perform best with 4 products, i.e., 4.65%. This performance is the same as the trivial FIX baseline, which also fulfills the criterion for 4 products. Turhan09 and Nam15 both fulfull the criterion for the products sklebagd, wspomaganiepi, and zuzel from the JURECZKO data, i.e., three of the four products are overlapping. Interestingly, FIX predicts well on four other products. Hence, none of the research could advance the state-of-the-art to such a degree, that metric based predictions can fulfill this criterion. Actually, the trivial baseline FIX could not even be outperformed. Even with the CV as baseline, i.e., with WPDP it is not possible to meet this performance criterion consistently. In total, only 317 of the 37,824 results we collected for our benchmark for the data sets JUREZCKO, MDP, AEEEM, NETGENE, and RELINK fulfill the criterion, i.e., for 0.83% of results. If we exclude the baselines FIX, CV, and RANDOM, this number drops to 268, i.e., to 0.73%. In order to see if relaxing the criterion helps, we also evaluated the results for the relaxed criterion of 0.7 *recall* and 0.5 *precision* as proposed by He *et al.* [43]. Now, CV-DT fulfills the criterion for 19 products, i.e., 22.09% of the

TABLE 10
Products where any CPDP approach fulfills the criterion by Zimmermann *et al.* [9] and the *precision* of the trivial prediction FIX.

Product	Data Set	#Inst.	#Appr.	*precision* FIX
berek	JURECZKO	43	36.7	0.37
ckjm	JURECZKO	10	3.4	0.50
openintents	RELINK	56	2	0.39
pbeans1	JURECZKO	26	5	0.77
pdftranslator	JURECZKO	33	4.4	0.45
poi-3.0	JURECZKO	442	5.4	0.64
sklebagd	JURECZKO	20	16	0.60
termoproject	JURECZKO	42	1	0.31
wspomaganiepi	JURECZKO	18	23.3	0.67
zuzel	JURECZKO	29	7	0.45

TABLE 11
Products where no CPDP approach fulfills the criterion by Zimmermann *et al.* [9] but the trivial baseline FIX does.

Product	Data Set	#Inst.	*precision* FIX
log4j-1.2	JURECZKO	205	0.92
velocity-1.4	JURECZKO	196	0.75
xalan-2.7	JURECZKO	909	0.99

products, followed by Liu10-GP with 18.8 products[12] of the 86 products, i.e., for 21.86% and FIX with 18 products, i.e., 20.93%. Hence, the success rate goes up to about one in five, but is still on the level of trivial predictions. The total number of results, including the baselines that fulfill the relaxed criterion is 1,755, i.e., for 4.62% of the results. If we exclude the baselines this number drops to 1,639, i.e., for 4.49% of the results.

Due to the strong performance of the trivial predictions, we also changed our focus from approaches to products and investigated for which products the criterion by Zimmermann *et al.* was fulfilled. Table 10 list the 10 products from all data sets for which the criterion was fulfilled at least once by an CPDP approach, i.e., not including the baselines FIX, CV, and RANDOM. The table also shows the *precision* achieved with the trivial baseline FIX. Please note that for FIX *precision* equals the *accuracy* and the *recall* is always one. Hence, for the baseline FIX the criterion is fulfilled if *precision* $>= 0.75$. The 10 products include nine out of 21 products with less than 100 instances. Thus, the approaches from the CPDP literature only fulfill the criterion for one product (poi-3.0) out of 65 products from all data sets that are not very small. Table 11 lists three products within the data, where the FIX baseline fulfills the criterion, but none of the CPDP approaches does. Thus, three products within the data can be predicted sufficiently accurate with a trivial approach, but by none of the CPDP approaches.

12. The fraction is possible due to the 10 repetitions. For one product the prediction fulfilled the criterion in 8 out of 10 replications.

TABLE 12
Difference between the results achieved with the JURECZKO and SELECTEDJURECZKO data. Bold-faced values are significantly higher.

	JURECZKO	SELECTEDJURECZKO	*p-value*
AUC	0.64	0.65	0.297
F-Measure	0.32	**0.36**	0.001
G-Measure	0.38	**0.43**	0.003
MCC	0.18	**0.21**	< 0.001

Answer RQ2: No approach allows for predictions that fulfill this criterion consistently, not even WPDP with CV. Only for small products the criterion is fulfilled, but also irregularly. Even the relaxed criterion of 0.7 *recall* and 0.5 *precision* is not fulfilled consistently. Moreover, even if trivial prediction can fulfill the criterion by Zimmermann *et al.*, the CPDP approaches do not.

5.3 RQ3: What is the impact of using only larger products on the benchmark results?

Table 8 shows the difference between the JURECZKO data and the FILTERJURECZKO data. The comparison between the JURECZKO and FILTERJUREZCKO data using the Mann-Whitney-U shows no significant differences between the two data sets in the metrics *AUC*, *F-measure*, *G-measure*, *MCC*, and *AUCEC*. The largest difference between the two sets is for the *G-measure* with a mean difference of $\mu = 0.02$ and a standard deviation of $\sigma = 0.02$.

Answer RQ3: There is almost no difference between using all of the JURECZKO data and using the FILTERJURECZKO data that does not contain small products.

5.4 RQ4: What is the impact of using a relatively small subset of a larger data set on the benchmark results?

Table 8 shows the difference between the JURECZKO data and the SELECTEDJURECZKO data and Table 12 the overall mean values for all approaches and the *p-value* of the Mann-Whitney-U tests performed. The comparison revealed statistically significant difference for all metrics except *AUC*. For *F-measure*, *G-measure*, and *MCC* the mean performance is significantly higher with the SELECTEDJURECZKO data. The deviation between the JURECZKO and SELECTEDJURECZKO data has the highest value for the *G-measure* with about 0.05 higher mean value, i.e., a performance overestimation by five percent.

Answer RQ4: There are statistically significant differences between the JUREZCKO data and the SELECTEDJURECZKO data. Our findings indicate that performance may increase by up to 5%.

6 Discussion

Within this section, we discuss our results, lessons that we learned from the benchmark, and threats to validity.

6.1 Insights from the Benchmark

The results of the benchmark were quite surprising for us. First of all, we did not expect that with CamargoCruz09 one of the first approaches to be proposed would be at the top of our ranking. Additionally, with Turhan09 another seven year old approach and with Watanabe08 even an eight year old approach are also part of the top ranked approaches. Hence, three approaches that are at least seven years old are within the four top ranked results. What is even more surprising is that Turhan09 was actually replicated quite a few times already and researchers frequently concluded that their approaches outperform the proposed k-Nearest Neighbor (NN) relevancy filter. However, our results show that if different performance metrics and data sets are considered this is not statistically significantly the case, and the opposite is true. Regarding the similarly high ranking of CamargoCruz09 and Watanabe08, we note that both approaches use standardization. Hence, simple standardization of the data already works very well.

Another finding is that different classifiers perform best for different approaches. DT (9 times) and RF (8 times) and are most often among the best results, however, all others except the SVM are also present. Furthermore, we note that for many approaches the differences between the classifiers were rather small. Details on the classifier performance can be found in the additional results published along with this benchmark (see Section 5). We also note that the SVM often yields trivial classifiers. This is due to the fact that most approaches do not perform imbalance treatment and SVMs are very sensitive to imbalance. However, for ZHe13 (which treats the bias with undersampling) and for Herbold13 (which treats the bias with equal weighting) the SVM also performs quite well but not best. This indication is in line with the suggestion by Hall *et al.* [83], who determined in their literature review on defect prediction, that treating a bias in the data is often neglected, but has a potentially big impact on the results. However, further investigations in this direction are required to draw firm conclusions for CPDP. We also note that the evaluation of differences between classifiers is not the focus of this benchmark and requires further research to see, e.g., how the findings of classifier comparison studies (e.g., [60], [65]) apply to our results.

The approach Nam2015, which is actually unsupervised defect prediction and, therefore, could render the need for CPDP ranks in the mid-field overall, with a *rankscore* of 0.834. This means that many CPDP approaches outperform this unsupervised approached. However, Nam2015 is among the best when it comes to fullfilling Zimmermann *et al.*'s performance criterion [9] and the *rankscore* of 0.834 is also not devastating. This means that in principle, unsupervised approaches can be a challenger for CPDP and should also be considered in future benchmarks for CPDP approaches. Moreover, it should be analyzed under which circumstances the unsupervised approaches work well. Our results indicate that they may be well suited for small projects, as indicated by the fullfillment of Zimmermann *et al.*'s performance criterion on four small products shows. However, a detailed study to generate further insights into this problem is required.

Our findings regarding **RQ2** revealed three things. The first is the obvious: the approaches fail to achieve the postulated prediction goal. Menzies *et al.* [84] already discussed possible ceiling effects of defect prediction models if only static code attributes are used. The AEEEM and NETGENE data also include other metric types. Both data sets contain process metrics, the NETGENE data also network metrics. Both types of information do not help to improve the models to such a degree that they fulfill the criterion. Hence, further investigations into a potential ceiling effect, also for other metric types, are warranted. The second is more important for future studies: small products do not allow for good conclusions. Apparently, getting a CPDP approach to work on a small product is much simpler than for larger products. To our mind, there are two possible explanations for this: either the fewer instances in the small products lead to a bigger random chance of fulfilling the performance goals or smaller products are easier to predict. We believe it is a mixture of both and, thereby, urge researchers to be cautious when drawing conclusions based on predictions on very small data products. The third is that one should always check if trivial approaches already yield sufficient performance. The check if the trivial prediction of everything being defective fulfills the criterion revealed that trivial predictions are actually possible for some products, where the CPDP approaches failed.

Moreover, our benchmark revealed the need for using multiple data sources and performance metrics. If we would have used only one performance metric and one data set, our results may have looked quite different. For example, if we only would have used the metric *AUC* and the JURECZKO data, the approach Uchigaki12-LE would be among the best results. However, this approach performs poorly in terms of all other performance metrics and is, therefore, ranked very low in our benchmark. Additionally, our evaluation regarding **RQ3** and **RQ4** revealed that subsetting of data is possible without changing the results drastically (**RQ3**), but small subsets may change the results (**RQ4**).

6.2 Lessons Learned

We learned three lessons from our benchmark, which we, as a (cross-project) defect prediction community, should try to respect to ensure progress towards solving the open problems in the state-of-the-art.

Replicate! Always replicate the results of previous studies. Do not just compare yourself against baselines like cross validation, but directly against approaches proposed by other researchers. And do not just compare yourself against one or two large publications, but all competitors. Our results show that it is not sufficient to just discuss related work, but that actual comparisons are required to demonstrate if and how the state-of-the-art is advanced. Otherwise it is easy to overlook promising approaches and hard to claim that your own work actually advances the state-of-the-art. As motivation for this, consider that to the best of our knowledge, the approach by Camargo Cruz *et al.* [10] was never replicated before. In case a comparison against all competitors is not possible for some reason, we suggest to compare yourself at least to Camargo Cruz *et al.* (2009) [10], Turhan *et al.* (2009) [8], Menzies *et al.* (2011, 2013) [12], [13], and Watanabe *et al.* (2008) [7], as each of them is for over 95% of performance metrics and data sets among the statistically significant best results. If the same data sets as in our benchmark are used, you may directly compare yourself to the results we achieved.

Share! While sharing data is quite common within our community, sharing implementations is not. For almost none of the replicated approaches, we could find a publicly available implementation online. This meant that we had to put a lot of effort into implementing approaches, which were already implemented. Sharing implementations can help the community a lot with replicating results. Recently, cloud platforms that enable sharing of implementations, data, and results at the same time were proposed as a solution [85]. Another approach could be the sharing of R packages [86].

Diversify! Our study shows that different metrics yield different results, hence, depending on the point of view, different approaches might perform best. Hence, we suggest that during the evaluation of approaches multiple perspectives should be considered, e.g., through multiple performance metrics covering different aspects, different data sets that cover different aspects of reality, or different potential applications. Such evaluations give a more holistic view on the capabilities of suggested approaches and techniques, from which we as a community will benefit in the long term.

6.3 Threats to Validity

Our benchmark has several threats to its validity. We distinguish between internal validity, construct validity, and external validity.

6.3.1 Internal Validity

Since our focus was on replicating existing work as is without changes or tuning of parameters, we do not see any internal threats to the validity of our results.

6.3.2 Construct Validity

The construction of our benchmark may influence the results and the findings. The performance metrics that were used might be unsuitable. A different set of metrics may lead to different results. Additionally, the ranking based on the Friedmann test with post-hoc Nemenyi test has a strong impact on the overall evaluation. A different statistical test may lead to a different ranking. Furthermore, the way we combined different rankings using the *rankscore* influences the overall ranking of our benchmark. A different approach may lead to another ranking. The data itself may also be noisy and contain mislabelled instances [40], [68] which would influence the results. Finally, even though we tested our implementation and re-used existing libraries where possible, we cannot exclude that our implementations contains bugs or that we misunderstood implementation details from the publications that we replicated. However, we tested all approaches on small data sets and checked the consistency of the results. Moreover, we double-checked for all approaches that the implementations match the descriptions in the publications. Additionally, we compared the results of our benchmark to the results of the original publications, in case the similar data was used as a sanity check. This check also revealed no inconsistencies.

6.3.3 External Validity

Our main concern regarding the external validity of the results is that the data that we used might not be representative for software in general. Although we used quite a large corpus of data within this study, with products from five different data sets, it were still only 85 products. Hence, we cannot rule out that the effect is random and depends purely on the analyzed data and the benchmark results may change drastically in the case that other data is used.

Additionally, our data did not contain any social metrics or context factors. Approaches that utilize such information should be considered for future benchmarks and may outperform the approaches presented in this benchmark.

7 Conclusion

Within this article, we presented the results of the replication of 24 approaches for CPDP and their ranking based on the performance on five data sets measured with four performance metrics. A standardization approach based on the median value of the target and training data in combination with taking the logarithm of metric values proposed by Camargo Cruz and Ochimizu [10] was determined to perform best, as it always was in the group of results with the statistically significant best performance. The approach is followed by other approaches proposed by Turhan *et al.* [8], Menzies *et al.* [12], [13], and Watanabe *et al.* [7]. However, none of the replicated approaches yields a consistently high performance according to the performance criterion by Zimmermann *et al.* [9], i.e., a performance of 0.75 *recall*, *precision*, and *accuracy*. Additionally, we determined that the performance does not significantly change if a (still large) subset of products is used, where each product has at least 100 classes, in comparison to using all products. The impact on the mean performance was almost non-existent. On the other hand, using a rather small subset of products can improve the mean results statistically significantly and by several percent. Therefore, selection of subsets from a large data set should always be done carefully and based on good reasons.

In the future, we will extend our benchmark with new approaches proposed by other researchers. Moreover, we hope that other researchers benefit from our replication of techniques and would be happy if our implementations can help with their future research on CPDP. Moreover, we plan to scale up our benchmark using new and larger data, such that the ranking is not based only on 85 software products, but on a large body of products with hundreds, or thousands of products that can be seen as representative for a large part of software. Finally, we will use our insights into techniques and their implementations to advance the state-of-the-art of CPDP with new techniques with the hope to achieve the performance goal postulated by Zimmermann *et al.* [9] consistently and be able to provide a good defect prediction tool for software engineers. As part of these efforts, we will also benchmark modifications of the proposed state of the art. Concretely, we will re-visit all approaches and extend them with a treatment to address the class imbalance problem following the advice by Hall *et al.* [83]. Additionally, a recent study by Tantithamthavorn *et al.* [87] determined that automated hyperparameter tuning can improve cross-project defect prediction models. We will adopt their approach for all classification models, and if possible also hyperparameters of the CPDP approaches themselves, e.g., the neighborhood size of the relevancy filter proposed by Turhan *et al.* [8].

We will also revisit the results of our benchmark to provide further insights. Currently, our benchmark provides lots of raw result data, together with a statistical analysis which approaches perform best, but rather few visualizations that allow deeper insights into the benchmark results. As part of our future work, we will extend this with further visual evaluation tools, e.g., to allow insights not only on the overall results on a data set, but also specifically for each product in a data set.

References

[1] V. Basili, L. Briand, and W. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751–761, 1996.

[2] S. Herbold, "A systematic mapping study on cross-project defect prediction," *CoRR*, vol. abs/1705.06429, 2017. [Online]. Available: https://arxiv.org/abs/1705.06429

[3] ——, "CrossPare: A Tool for Benchmarking Cross-Project Defect Predictions." The 4th International Workshop on Software Mining (SoftMine), 2015.

[4] S. Herbold, A. Trautsch, and J. Grabowski, "Global vs. local models for cross-project defect prediction," *Empirical Software Engineering*, pp. 1–37, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10664-016-9468-y

[5] B. Turhan, A. Tosun, and A. Bener, "Empirical evaluation of mixed-project defect prediction models," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, Aug 2011, pp. 396–403.

[6] T. M. Khoshgoftaar, P. Rebours, and N. Seliya, "Software quality analysis by combining multiple projects and learners," *Software Quality Journal*, vol. 17, no. 1, pp. 25–49, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11219-008-9058-3

[7] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter language reuse," in *Proc. 4th Int. Workshop on Predictor Models in Softw. Eng. (PROMISE)*. ACM, 2008.

[8] B. Turhan, T. Menzies, A. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Eng.*, vol. 14, pp. 540–578, 2009.

[9] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proc. the 7th Joint Meet. Eur. Softw. Eng. Conf. (ESEC) and the ACM SIGSOFT Symp. Found. Softw. Eng. (FSE)*. ACM, 2009, pp. 91–100.

[10] A. E. Camargo Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in *Proc. 3rd Int. Symp. on Empirical Softw. Eng. and Measurement (ESEM)*. IEEE Computer Society, 2009.

[11] Y. Liu, T. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *Software Engineering, IEEE Transactions on*, vol. 36, no. 6, pp. 852–864, Nov 2010.

[12] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. Cok, "Local vs. global models for effort estimation and defect prediction," in *Proc. 26th IEEE/ACM Int. Conf. on Automated Softw. Eng. (ASE)*. IEEE Computer Society, 2011.

[13] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local versus global lessons for defect prediction and effort estimation," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 822–834, June 2013.

[14] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Inf. Softw. Technology*, vol. 54, no. 3, pp. 248 – 256, 2012.

[15] F. Peters and T. Menzies, "Privacy and utility for defect prediction: Experiments with morph," in *Software Engineering (ICSE), 2012 34th International Conference on*, June 2012, pp. 189–199.

[16] S. Uchigaki, S. Uchida, K. Toda, and A. Monden, "An ensemble approach of simple regression models to cross-project fault prediction," in *Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, Aug 2012, pp. 476–481.

[17] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *Proc. 6th IEEE Int. Conf. Softw. Testing, Verification and Validation (ICST)*, 2013.

[18] ——, "Defect prediction as a multiobjective optimization problem," *Software Testing, Verification and Reliability*, vol. 25, no. 4, pp. 426–459, 2015. [Online]. Available: http://dx.doi.org/10.1002/stvr.1570

[19] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing privacy and utility in cross-company defect prediction," *Software Engineering, IEEE Transactions on*, vol. 39, no. 8, pp. 1054–1068, Aug 2013.

[20] S. Herbold, "Training data selection for cross-project defect prediction," in *Proc. 9th Int. Conf. on Predictive Models in Softw. Eng. (PROMISE)*. ACM, 2013.

[21] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in *Proc. 7th Int. Symp. on Empirical Softw. Eng. and Measurement (ESEM)*, 2013.

[22] J. Nam, S. Pan, and S. Kim, "Transfer defect learning," in *Software Engineering (ICSE), 2013 35th International Conference on*, May 2013, pp. 382–391.

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

[24] D. Ryu, O. Choi, and J. Baik, "Value-cognitive boosting with a support vector machine for cross-project defect prediction," *Empirical Software Engineering*, vol. 21, no. 1, pp. 43–71, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10664-014-9346-4

[25] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119 – 139, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002200009791504X

[26] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170 – 190, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584914002523

[27] M. A. Hall, "Correlation-based feature subset selection for machine learning," Ph.D. dissertation, University of Waikato, Hamilton, New Zealand, 1998.

[28] F. Peters, T. Menzies, and L. Layman, "Lace2: Better privacy-preserving data sharing for cross project defect prediction," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 1, May 2015, pp. 801–811.

[29] K. Kawata, S. Amasaki, and T. Yokogawa, "Improving relevancy filter methods for cross-project defect prediction," in *Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence (ACIT-CSI), 2015 3rd International Conference on*, July 2015, pp. 2–7.

[30] M. Ester, H. peter Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Int. Conf. Knowledge Discovery and Data Mining*, 1996.

[31] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, vol. 2, July 2015, pp. 264–269.

[32] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[33] S. Amasaki, K. Kawata, and T. Yokogawa, "Improving cross-project defect prediction methods with data simplification," in *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*, Aug 2015, pp. 96–103.

[34] D. Ryu, J.-I. Jang, and J. Baik, "A hybrid instance selection using nearest-neighbor for cross-project defect prediction," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 969–980, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11390-015-1575-5

[35] P. C. Mahalanobis, "On the generalised distance in statistics," in *Proceedings National Institute of Science, India*, vol. 2, no. 1, Apr. 1936, pp. 49–55. [Online]. Available: http://ir.isical.ac.in/dspace/handle/1/1268

[36] B. Raman and T. R. Ioerger, "Enhancing learning using feature and example selection," Technical Report, Department of Computer Science, Texas A&M University, 2003.

[37] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, Nov 2015, pp. 452–463.

[38] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 706–720, Jul 2002.

[39] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proc. ACM/IEEE 28th Int. Conf. on Softw. Eng.*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 452–461.

[40] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proc. 6th Int. Conf. on Predictive Models in Softw. Eng. (PROMISE)*. ACM, 2010.

[41] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of cross-project defect prediction," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng. (FSE)*. ACM, 2012.

[42] B. Turhan, "On the dataset shift problem in software engineering prediction models," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 62–74, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9182-8

[43] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Softw. Eng.*, vol. 19, pp. 167–199, 2012.

[44] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 409–418. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487161

[45] E. Kocaguneli, B. Cukic, and H. Lu, "Predicting more from less: Synergies of learning," in *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on*, May 2013, pp. 42–48.

[46] B. Turhan, A. T. Misirli, and A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," *Information & Software Technology*, vol. 55, no. 6, pp. 1101–1118, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2012.10.003

[47] P. Singh, S. Verma, and O. P. Vyas, "Article: Cross company and within company fault prediction using object oriented metrics," *International Journal of Computer Applications*, vol. 74, no. 8, pp. 5–11, July 2013, full text available.

[48] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards Building a Universal Defect Prediction Model," in *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*. ACM, 2014.

[49] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 172–181. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597075

[50] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Software Engineering*, pp. 1–35, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10664-015-9400-x

[51] O. Mizuno and Y. Hirata, "A cross-project evaluation of text-based fault-prone module prediction," in *Empirical Software Engineering in Practice (IWESEP), 2014 6th International Workshop on*, Nov 2014, pp. 43–48.

[52] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Negative samples reduction in cross-company software defects prediction," *Information and Software Technology*, vol. 62, pp. 67 – 77, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584915000348

[53] D. Ryu, J.-I. Jang, and J. Baik, "A transfer cost-sensitive boosting approach for cross-project defect prediction," *Software Quality Journal*, pp. 1–38, 2015a. [Online]. Available: http://dx.doi.org/10.1007/s11219-015-9287-1

[54] J. Nam and S. Kim, "Heterogeneous defect prediction," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 508–519. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786814

[55] X. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 496–507. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786813

[56] Q. Cao, Q. Sun, Q. Cao, and H. Tan, "Software defect prediction via transfer learning based neural network," in *Reliability Systems Engineering (ICRSE), 2015 First International Conference on*, Oct 2015, pp. 1–10.

[57] M. Jureczko and L. Madeyski, "Cross–project defect prediction with respect to code ownership model: An empirical study," *e-Informatica Software Engineering Journal*, vol. 9, pp. 21–35, 2015.

[58] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[59] H. Altinger, S. Herbold, J. Grabowski, and F. Wotawa, *Testing Software and Systems: 27th IFIP WG 6.1 International Conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, November 23-25, 2015, Proceedings*. Cham: Springer International Publishing, 2015, ch. Novel Insights on Cross Project Fault Prediction Applied to Automotive Software, pp. 141–157. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25945-1_9

[60] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, July 2008.

[61] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248548

[62] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940. [Online]. Available: http://www.jstor.org/stable/2235971

[63] P. Nemenyi, "Distribution-free multiple comparison," Ph.D. dissertation, Princeton University, 1963.

[64] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Softw. Engg.*, vol. 17, no. 4-5, pp. 531–577, Aug. 2012. [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9173-9

[65] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 1, May 2015, pp. 789–800.

[66] R. A. Fisher, "The correlation between relatives on the supposition of mendelian inheritance," *Philosophical Transactions of the Royal Society of Edinburgh*, vol. 52, pp. 399–433, 1918.

[67] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, 2013.

[68] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, *The misuse of the NASA metrics data program data sets for automated software defect prediction*. IET, 2011, pp. 96–103.

[69] M. D'Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," in *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE Computer Society, 2010.

[70] S. Herbold, "sherbold/replication-kit-tse-2017-benchmark: Release of the replication kit," May 2017. [Online]. Available: https://doi.org/10.5281/zenodo.581178

[71] K. Herzig, S. Just, A. Rau, and A. Zeller, "Predicting defects using change genealogies," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, Nov 2013, pp. 118–127.

[72] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, ser. PROMISE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–. [Online]. Available: http://dx.doi.org/10.1109/PROMISE.2007.10

[73] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 15–25. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025120

[74] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[75] D. R. Cox, "The regression analysis of binary sequences (with discussion)," *J Roy Stat Soc B*, vol. 20, pp. 215–242, 1958.

[76] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*. Prentice hall Upper Saddle River, 2003, vol. 2.

[77] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010933404324

[78] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 161–168. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143865

[79] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," DTIC Document, Tech. Rep., 1988.

[80] T. van Gestel, J. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. de Moor, and J. Vandewalle, "Benchmarking Least Squares Support Vector Machine Classifiers," *Machine Learning*, vol. 54, no. 1, pp. 5–32, 2004.

[81] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 99–108. [Online]. Available: http://dl.acm.org/citation.cfm?id=2819009.2819026

[82] H. B. Mann and D. R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," *The Ann. of Math. Stat.*, vol. 18, no. 1, pp. pp. 50–60, 1947.

[83] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1109/TSE.2011.103

[84] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proc. 4th Int. Workshop on Predictor Models in Softw. Eng. (PROMISE)*. ACM, 2008.

[85] F. Trautsch, S. Herbold, P. Makedonski, and J. Grabowski, "Adressing problems with external validity of repository mining studies through a smart data platform," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 97–108. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901753

[86] L. Madeyski and B. Kitchenham, "Reproducible research–what, why and how," Technical report, WrUT, Report PRE W08/2015/P-02, 2015.

[87] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th International Conference on Software Engineering*. ACM, 2016.

[88] B. Henderson-Sellers, *Object-oriented Metrics; Measures of Complexity*. Prentice-Hall Inc., 1996.

[89] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA: Elsevier Science Inc., 1977.

APPENDIX A
TABLE OF ACRONYMS

ANOVA	ANalysis Of VAriance
AST	Abstract Syntax Tree
AUC	Area Under the ROC Curve
Ca	Afferent Coupling
CBO	Coupling Between Objects
CCA	Canonical Correlation Analysis
CD	Critical Distance
Ce	Efferent Coupling
CFS	Correlation-based Feature Subset
CLA	Clustering and LAbeling
CODEP	COmbined DEfect Predictor
CPDP	Cross-Project Defect Prediction
DBSCAN	Density-Based Spatial Clustering
DCV	Dataset Characteristic Vector
DTB	Double Transfer Boosting
fn	false negative
fp	false positive
GB	GigaByte
HL	Hosmer-Lemeshow
ITS	Issue Tracking System
JIT	Just In Time
LCOM	Lack of COhession between Methods
LOC	Lines Of Code
MDP	Metrics Data Program
MI	Metric and Instances selection
MODEP	MultiObjective DEfect Predictor
MPDP	Mixed-Project Defect Prediction
NN	Nearest Neighbor
PCA	Principle Component Analysis
RAM	Random Access Memory
RFC	Response For a Class
SCM	SourceCode Management system
SVM	Support Vector Machine
TCA	Transfer Component Analysis
tn	true negative
tp	true positive
RBF	Radial Basis Function
ROC	Receiver Operating Characteristic
UMR	Unified Metric Representation
VCB	Value-Cognitive Boosting
WPDP	Within-Project Defect Prediction

APPENDIX B
DATA SET DETAILS

B.1 JURECZKO / FILTERJURECZKO / SELECTED-JURECZKO Data

The following metrics are part of the JURECZKO/FILTERJURECZKO data:

- WMC: weighted method count, number of methods in a class
- DIT: depth of inheritance tree
- NOC: number of children
- CBO: coupling between objects, number of classes coupled to a class
- RFC: response for class, number of different methods that can be executed if the class receives a message
- LCOM: lack of cohesion in methods, number of methods not related through the sharing of some of the class fields
- LCOM3: lack of cohesion in methods after [88]
- NPM: number of public methods
- DAM: data access metric, ratio of private (protected) attributes to total number of attributes in the class
- MOA: measure of aggregation, number of class fields whose types are user defined classes
- MFA: measure of functional abstraction, ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class
- CAM: cohesion among methods of class, relatedness of methods based upon the parameter list of the methods
- IC: inheritance coupling, number of parent classes to which the class is coupled
- CBM: coupling between methods, number of new/redefined methods to which all the inherited methods are coupled
- AMC: average method complexity
- Ca: afferent couplings
- Ce: efferent couplings
- CC: cyclomatic complexity
- Max(CC): maximum cyclomatic complexity among methods
- Avg(CC): average cyclomatic complexity among methods

For a detailed explanation see [40].

B.2 MDP Data

The following metrics are part of the MDP data. This is the common subset of metrics that is obtained by all projects within the MDP data set:

- LOC_TOTAL: total lines of code
- LOC_EXECUTABLE: exectuable lines of code
- LOC_COMMENTS: lines of comments
- LOC_CODE_AND_COMMENT: lines with comments or code
- NUM_UNIQUE_OPERATORS: number of unique operators
- NUM_UNIQUE_OPERANDS: number of unique operands
- NUM_OPERATORS: total number of operators
- NUM_OPERANDS: total number of operands
- HALSTEAD_VOLUME: Halstead volume (see [89])
- HALSTEAD_LENGTH: Halstead length (see [89])
- HALSTEAD_DIFFICULTY: Halstead difficulty (see [89])
- HALSTEAD_EFFORT: Halstead effort (see [89])

TABLE 13
Software products from the JURECZKO / FILTERJURECZKO data, including their total number of classes and the number of defect-prone classes. Products that are not part of FILTERJURECZKO are italic. Products that are part of the SELECTEDJURECZKO data are boldface.

Product	#Classes	#Defect-prone	% Defect-prone
ant 1.3	125	20	16%
ant 1.4	178	40	22%
ant 1.5	293	32	11%
ant 1.6	351	92	26%
ant 1.7	745	166	22%
arc	234	27	12%
berek	43	16	37%
camel 1.0	339	11	4%
camel 1.2	608	216	36%
camel 1.4	872	145	17%
camel 1.6	965	188	19%
ckjm	10	5	50%
e-learning	64	5	8%
forrest 0.7	29	5	17%
ivy 1.1	111	63	57%
ivy 1.4	241	16	7%
ivy 2.0	352	40	11%
jedit 3.2	272	90	33%
jedit 4.0	306	75	25%
jedit 4.1	312	79	25%
jedit 4.2	367	48	13%
jedit 4.3	492	11	2%
kalkulator	27	6	22%
log4j 1.0	135	34	25%
log4j 1.1	109	37	34%
log4j 1.2	205	189	92%
lucene 2.0	195	91	47%
lucene 2.2	247	144	58%
lucene 2.4	340	203	60%
nieruchomosci	27	10	37%
pbeans 1	26	20	77%
pbeans 2	51	10	20%
pdftranslator	33	15	45%
poi 1.5	237	141	59%
poi 2.0	314	37	12%
poi 2.5	385	248	64%
poi 3.0	442	281	64%
redaktor	176	27	15%
serapion	45	9	20%
skarbonka	45	9	20%
sklebagd	20	12	60%
synapse 1.0	157	16	10%
synapse 1.1	222	60	27%
synapse 1.2	256	86	34%
systemdata	65	9	14%
szybkafucha	25	14	56%
termoproject	42	13	31%
tomcat	858	77	9%
velocity 1.4	196	147	75%
velocity 1.5	214	142	66%
velocity 1.6	220	78	35%
workflow	39	20	51%
wspomaganiepi	18	12	67%
xalan 2.4	723	110	15%
xalan 2.5	803	387	48%
xalan 2.6	885	411	46%
xalan 2.7	909	898	99%
xerces initial	162	77	48%
xerces 1.2	440	71	16%
xerces 1.3	453	69	15%
xerces 1.4	588	437	74%
zuzel	29	13	45%
Total	17681	6062	34%

TABLE 14
Software products from the MDP data that are part of the study, including their total number of modules and the number of defect-prone modules.

Product	#Modules	#Defect-prone	% Defect-prone
CM1	344	42	12%
JM1	9593	1759	18%
KC1	2096	325	16%
KC3	200	36	18%
MC1	9277	68	1%
MC2	127	44	35%
MW1	264	27	10%
PC1	759	61	8%
PC2	1585	16	1%
PC3	1125	140	12%
PC4	1399	178	13%
PC5	17001	503	3%
Total	43770	3199	7%

- HALSTEAD_ERROR_EST: Halstead Error, also known as Halstead Bug ((see [89]))
- HALSTEAD_PROG_TIME: Halstead Pro
- BRANCH_COUNT: Number of branches
- CYCLOMATIC_COMPLEXITY: Cyclomatic complexity (same as CC in the JSTAT data)
- DESIGN_COMPLEXITY: design complexity

B.3 AEEEM Data

The following metrics are part of the AEEEM data:

- *CBO: coupling between objects*
- *DIT: depth of inheritance tree*
- *fanIn: number of other classes that reference the class*
- *fanOut: number of other classes referenced by the class*
- *LCOM: lack of cohesion in methods*
- *NOC: number of children*
- *RFC: response for class*
- *WMC: weighted method count*
- *NOA: number of attributes*
- *NOAI: number of attributes inherited*
- *LOC: lines of code*
- *NOM: number of methods*
- *NOMI: number of methods inherited*
- *NOPRA: number of private attributes*
- *NOPRM: number of private methods*
- *NOPA: number of public attributes*
- *NOPM: number of public methods*
- NR: number of revisions
- NREF: number of times the file has been refactored
- NAUTH: number of authors
- LADD: sum of lines added
- max(LADD): maximum lines added
- avg(LADD): average lines added
- LDEL: sum of lines removed
- max(LDEL): maximum lines deleted
- avg(LDEL): average lines deleted
- CHURN: sum of code churn
- max(CHURN): maximum code churn
- avg(CHURN): average code churn

TABLE 15
Software products from the AEEEM data that are part of the study, including their total number of classes and the number of defect-prone classes.

Product	#Classes	#Defect-prone	% Defect-prone
lucene	691	64	9%
pde	1497	209	14%
mylyn	1862	245	13%
eclipse	997	206	21%
equinox	324	129	40%
Total	5371	893	16%

TABLE 16
Software products from the NETGENE data that are part of the study, including their total number of classes and the number of defect-prone classes.

Product	#Classes	#Defect-prone	% Defect-prone
httpclient	361	205	57%
jackrabbit	542	225	42%
lucene	1671	346	11%
rhino	253	109	43%
Total	2827	885	31%

- AGE: age of the file
- WAGE: weighted age of the file

The italic metrics are included in three different variants: with their actual metric values, as well as with the weighted churn (WCHU) and linear decayed entropy (LDHH). For a detailed explanation see [69].

B.4 NETGENE Data

Due to the large number of metrics, we ask the reader to consider the original publication for the list of the 465 metrics [71].

B.5 RELINK Data

The following metrics are part of the RELINK data:

- AvgCyclomatic: average cyclomatic complexity for all nested functions or methods
- AvgCyclomaticModified: average modified cylcomatic complexity for all nested functions and methods
- AvgCyclomaticStrict: average strict cyclomatic complexity for all nested functions or methods
- AvgEssential: average essential complexity for all nested functions or methods
- AvgLine: average number of lines for all nested functions or methods
- AvgLineBlank: average number of blank lines for all nested functions or methods
- AvgLineCode: average number of lines containing source code for all nested functions or methods
- AvgLineComment: average number of lines containing comment for all nested functions or methods
- CountClassBase: number of immediate base classes
- CountClassCoupled: number of other classes coupled to
- CountClassDerived: number of immediate subclasses
- CountDeclClass: number of classes
- CountDeclClassMethod: number of class methods
- CountDeclClassVariable: number of class variables
- CountDeclFunction: number of functions
- CountDeclInstanceMethod: number of instance methods
- CountDeclInstanceVariable: number of instance variables
- CountDeclInstanceVariablePrivate: number of private instance variables
- CountDeclInstanceVariableProtected: number of protected instance variables
- CountDeclInstanceVariablePublic: number of public instance variables
- CountDeclMethod: number of local methods
- CountDeclMethodAll: number of methods, including inherited ones
- CountDeclMethodConst: number of local const methods
- CountDeclMethodFriend: number of local friend methods
- CountDeclMethodPrivate: number of local private methods
- CountDeclMethodProtected: number of local protected methods
- CountDeclMethodPublic: number of local public methods
- CountInput: number of classing subprograms plus global variables read
- CountLine: number of all lines
- CountLineBlank: number of blank lines
- CountLineCode: number of lines containing source code
- CountLineCodeDecl: number of lines containing declarative source code
- CountLineCodeExe: number of lines containing exectuable source code
- CountLineComment: number of lines containing comment
- CountLineInactive: number of inactive lines
- CountLinePreprocessor: number of preprocessor lines
- CountOutput: number of called subprograms plus global variables
- CountPath: number of possible paths not counting abnormal exits or gotos
- CountSemicolon: number of semicolons
- CountStmt: number of statements
- CountStmtDecl: number of declarative statements
- CountStmtExe: number of executable statements

TABLE 17
Software products from the RELINK data that are part of the study, including their total number of classes and the number of defect-prone classes.

Product	#Classes	#Defect-prone	% Defect-prone
Apache HTTP	194	98	51%
OI Safe	56	22	39%
ZXing	399	118	30%
Total	649	238	37%

- Cyclomatic: cyclomatic complexity
- CyclomaticModified: modified cyclomatic complexity
- CyclomaticStrict: strict cyclomatic complexity
- Essential: essential complexity
- Knots: measure of overlapping jumps
- MaxCyclomatic: maximum cylcomatic complexity of all nested functions or methods
- MaxCyclomaticModified: maximum modified cyclomatic complexity of all nested functions or methods
- MaxCyclomaticStrict: maximum strict cyclomatic complexity of all nested functions or methods
- MaxEssentialKnots: maximum knots after structured programming constructs have been removed
- MaxInheritanceTree: maximum depth of class in inheritance tree
- MaxNesting: maximum level of control constructs
- MinEssentialKnots: minimum knots after structured programming constructs have been removed
- PercentLackOfCohesion: 100% minus the average cohesion for package entities
- RatioCommentToCode: ratio of comment lines to code lines
- SumCyclomatic: sum of cyclomatic complexity for all nested functions and methods
- SumCyclomaticModified: sum of modified cyclomatic complexity for all nested functions and methods
- SumCyclomaticStrict: sum of strict cyclomatic complexity for all nested functions and methods
- SumEssential: sum of essential complexity of all nested functions and methods

For a detailed explanation see the Understand website (https://scitools.com/ (last checked: 2016-07-20)).

Steffen Herbold is a PostDoc and substitutional head of the research group Software Engineering for Distributed Systems at the Institute of Computer Science of the Georg-August-Universität Göttingen. He received his doctorate in 2012 from the Georg-August-Universitat Göttingen. Dr. Herbolds research interests is empirical software engineering, especially the application of data science methods. His research includes work on defect prediction, repository mining infrastructures, usage-based testing, model-based testing, as well as scaling analysis in cloud environments.

Alexander Trautsch is a M.Sc. student at the Georg-August-Universitat Göttingen. Alexander received his B.Sc. degree from the Georg-August-University in Göttingen in 2015. As part of his studies, Alexander is working on cross-project defect prediction, e.g., the implementation of local models, genetic algorithms, and heterogeneous approaches.

Jens Grabowski is professor at the Georg-August-Universitat Göttingen. He is vice director of the Institute of Computer Science and is heading the Software Engineering for Distributed Systems Group at the Institute. Prof. Grabowski is one of the developers of the standardized testing languages TTCN-3 and UML Testing Profile. The current research interests of Prof. Grabowski are directed towards model-based development and testing, managed software evolution, and empirical software engineering.