



On effort-aware metrics for defect prediction

Jonida Çarka¹ · Matteo Esposito¹ · Davide Falessi¹

Accepted: 31 May 2022 / Published online: 6 August 2022
© The Author(s) 2022

Abstract

Context Advances in defect prediction models, aka classifiers, have been validated via accuracy metrics. Effort-aware metrics (EAMs) relate to benefits provided by a classifier in accurately ranking defective entities such as classes or methods. PofB is an EAM that relates to a user that follows a ranking of the probability that an entity is defective, provided by the classifier. Despite the importance of EAMs, there is no study investigating EAMs trends and validity.

Aim The aim of this paper is twofold: 1) we reveal issues in EAMs usage, and 2) we propose and evaluate a normalization of PofBs (aka NPofBs), which is based on ranking defective entities by predicted defect density.

Method We perform a systematic mapping study featuring 152 primary studies in major journals and an empirical study featuring 10 EAMs, 10 classifiers, two industrial, and 12 open-source projects.

Results Our systematic mapping study reveals that most studies using EAMs use only a single EAM (e.g., PofB20) and that some studies mismatched EAMs names. The main result of our empirical study is that NPofBs are statistically and by orders of magnitude higher than PofBs.

Conclusions In conclusion, the proposed normalization of PofBs: (i) increases the realism of results as it relates to a better use of classifiers, and (ii) promotes the practical adoption of prediction models in industry as it shows higher benefits. Finally, we provide a tool to compute EAMs to support researchers in avoiding past issues in using EAMs.

Keywords Defect prediction · Accuracy metrics · Effort-aware metrics

Communicated by: Yasutaka Kamei

✉ Jonida Çarka
jonida.carka@students.uniroma2.eu

Matteo Esposito
m.esposito@ing.uniroma2.it

Davide Falessi
falessi@ing.uniroma2.it

¹ University of Rome Tor Vergata, Rome, Italy

1 Introduction

The manner in which defects are introduced into code, and the sheer volume of defects in software, are typically beyond the capability and resources of most development teams (Ghotra et al. 2017; Kamei et al. 2012; Kondo et al. 2019; Tantithamthavorn et al. 2019). Defect prediction models aim to identify software artifacts that are likely to be defective (Menzies et al. 2010; Ohlsson and Alberg 1996; Ostrand and Weyuker 2004; Ostrand et al. 2005; Turhan et al. 2009; Weyuker et al. 2010). The main purpose of defect prediction is to reduce the cost of testing, analysis, or code review by prioritizing developers' efforts on specific artifacts such as commits, methods, or classes.

Past studies investigated how to use defect prediction models, aka classifiers, to predict the defectiveness of different types of entities including commits (Fan et al. 2021; Giger et al. 2012; Herbold 2019; Herbold et al. 2020; Huang et al. 2019; Kondo et al. 2020; McIntosh and Kamei 2018; Pascarella et al. 2019, 2020; Rodríguez-Pérez et al. 2020; Tu et al. 2020), classes (Amasaki 2020; Bangash et al. 2020; Bennin et al. 2018, 2019; Chen et al. 2020; Chi et al. 2017; Dalla Palma et al. 2021; Herbold et al. 2017, 2018, 2019; Hosseini et al. 2019; Jiarpakdee et al. 2020; Jing et al. 2017; Kamei et al. 2016; Kondo et al. 2019; Lee et al. 2016; Liu et al. 2017; Morasca and Lavazza 2020; Mori and Uchihira 2019; Nucci et al. 2018; Palomba et al. 2019; Peters et al. 2019; Qu et al. 2021a; Shepperd et al. 2018; Song et al. 2019; Tantithamthavorn et al. 2016c, 2019, 2020; Tian et al. 2015; Yan et al. 2017; Yu et al. 2019; Zhang et al. 2016, 2017) or methods (Pascarella et al. 2019) by leveraging, for example, product metrics (Basili et al. 1996; Gyimóthy et al. 2005; Khoshgoftaar et al. 1996; Nagappan and Ball 2005; Hassan 2009), process metrics (Moser et al. 2008), knowledge from where previous defects occurred (Ostrand et al. 2005; Kim et al. 2007), information about change-inducing fixes (Kim et al. 2008; Fukushima et al. 2014) and, recently, deep learning techniques to automatically engineer features from source code elements (Wang et al. 2016).

Accuracy metrics are important to validate the extent to which classifiers are accurate and would support potential users. The aim of this paper is to focus on a particular family of accuracy metrics called effort-aware metrics (EAM) (Jiang et al. 2013; Rahman et al. 2012). EAMs relate to the ranking, as provided by a classifier, of candidate defective entities (Mende and Koschke 2009). The better the classifier, the higher the number of defective entities a developer can identify within a given rank. The different EAMs vary in the thresholds used to stop analyzing the ranking and ranking criteria. For instance, *PofBx* is defined as the percentage of bugs that a developer can identify by inspecting the top x percent of lines of code (Chen et al. 2017).

Table 1 reports an example of *PofB20*. The dataset in Table 1 is 1790 LOC, and it features seven entities, three of which are defective. Looking at *PofB20*, a user following the rank of classes predicted to have the highest probability of defectiveness, as provided by a DMP, would stop at 20% of the dataset and hence would analyze only up to 358 LOC. Thus, the user would analyze only the first two entities, finding one-third of the defective entities. Thus, in this example, *PofB20* is 33%.

To better motivate the need to normalize the ranking according to the size of the ranked entities, we present here the same entities seven entities of Table 1 (Section 1) but with a normalized ranking. Specifically, Table 2 differently to Table 1, ranks the entities according to their predicted probability to be defective divided by their size rather than according to their predicted probability only. Looking at the normalization of *PofB20* (Table 2), a user, following the rank of classes based on the predicted probability to be defective, as

Table 1 Example of a dataset to compute an EAM metric called PofB20

PofB ranking				
ID	Size	Predicted probability	Actual	
Entity 1	120	0.89	Defective	
Entity 2	100	0.78	Not Defective	
Entity 3	250	0.73	Defective	
Entity 4	80	0.71	Defective	
Entity 5	140	0.59	Not Defective	
Entity 6	500	0.53	Not Defective	
Entity 7	600	0.51	Not Defective	

provided by a classifier, would stop at 20% of the dataset and hence would analyze only up to 358 LOC. Thus, by following a normalized ranking, the user would analyze the first three entities, finding two-thirds of the defective entities. Thus, in this example of seven entities, the normalization increased PofB20 from 33% to 66%.

Despite the importance of EAMs, there is no study investigating EAMs trends and validity. The aim of this paper is twofold: 1) we reveal issues in EAMs usage, and 2) we propose and evaluate a normalization of PofB called NPofB. NPofB measures the ranking effectiveness when the ranking is normalized by the size of the ranked, and possibly defective, entities. In this paper we provide the following contributions:

1. We reveal trends in EAMs usage. Our systematic mapping study, featuring 152 primary studies, in major software engineering journals, reveals a few issues, including that most studies using EAMs use only a single EAM (e.g., PofB20) and that some studies mismatched EAMs names.
2. We suggest normalizing the PofBs. The idea behind normalization is that the user will follow a ranking based on the probability that an entity is defective, as provided by the classifier, normalized (i.e., divided) by the size of this entity.
3. We validate the normalization of PofBs. By analyzing ten PofBs, ten classifiers, two industrial projects and 12 open-source projects, we show that the normalization increases the PofBs statistically and by orders of magnitude. This result means that: 1)studies reporting PofBs, rather than the proposed normalization (i.e., NPofB), underestimate the benefits of using classifiers for ranking defective classes. 2)The normalization increases the realism of PofBs due to better use of classifiers.

Table 2 Example of a dataset to compute the normalized PofB

Normalized PofB ranking				
ID	Size	Predicted probability	Predicted/Size	Actual
Entity 4	80	0.71	0.008875	Defective
Entity 2	100	0.78	0.0078	Not Defective
Entity 1	120	0.89	0.007416667	Defective
Entity 5	140	0.59	0.004214286	Not Defective
Entity 3	250	0.73	0.00292	Defective
Entity 6	500	0.53	0.00106	Not Defective
Entity 7	600	0.51	0.00085	Not Defective

4. We show that the proposed normalization changes the ranking of classifiers. Specifically, when considering the same dataset, in most cases, the best classifier for a PofB resulted as different from the best classifier for that normalized PofB.
5. We show that multiple PofBs are needed to support a comprehensive understanding of classifiers accuracy.
6. We provide a tool to compute EAMs to support researchers in: 1) avoiding extra effort in EAMs computation as there is no available tool to compute EAMs, 2) increasing results reproducibility, and 3) increasing results validity by avoiding EAMs misnaming and, 4) increasing results generalizability by avoiding single EAM usage.

The remainder of this paper is structured as follows. Section 2 discusses the related literature, focusing in particular on accuracy metrics for classifiers. Section 3 reports the design, Section 4 the results, and Section 5 a discussion of our study. Section 6 presents our ACUME tool. Section 7 provides the threats to validity of our investigation. Finally, Section 8 concludes the paper and outlines directions for future work.

2 Related Work

2.1 Accuracy Metrics

Accuracy metrics evaluate the ability of a classifier to provide correct classifications. Examples of accuracy metrics include the following:

- True Positive (TP): The class is actually defective and is predicted to be defective.
- False Negative (FN): The class is actually defective and is predicted to be non-defective.
- True Negative (TN): The class is actually non-defective and is predicted to be non-defective.
- False Positive (FP): The class is actually non-defective and is predicted to be defective.
- **Precision:** $\frac{TP}{TP+FP}$.
- **Recall:** $\frac{TP}{TP+FN}$.
- **F1-score:** $\frac{2*Precision*Recall}{Precision+Recall}$.
- **AUC** (Area Under the Receiving Operating Characteristic Curve) (Powers 2007) is the area under the curve, of true positive rate versus false positive rate, that is defined by setting multiple thresholds. AUC has the advantage of being threshold-independent.
- **MCC** (Matthews Correlation Coefficient) is commonly used in assessing the performance of classifiers dealing with unbalanced data (Matthews 1975), and is defined as: $\frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$. Its interpretation is similar to correlation measures, i.e., $MCC < 0.2$ is considered to be low, $0.2 \leq MCC < 0.4$ —fair, $0.4 \leq MCC < 0.6$ —moderate, $0.6 \leq MCC < 0.8$ —strong, and $MCC \geq 0.8$ —very strong.
- **Gmeasure:** $\frac{2*Recall*(1-pf)}{Recall+(1-pf)}$ is the harmonic mean between recall and probability of false alarm (pf), which denotes the ratio of the number of non-defective modules that are wrongly classified as defective to the total number of non-defective modules as $\frac{FP}{FP+TN}$ (Chen et al. 2020)

A drawback of the metrics above is that they somehow assume that the costs associated with testing activities are the same for each entity, which is not reasonable in practice. For

example, costs for unit testing and code reviews are roughly proportional to the size of the entity under test.

2.2 Effort-Aware Metrics

The rationale behind EAM is that they focus on effort reduction gained by using classifiers (Mende and Koschke 2009).

In general, there are two types of EAM: normalized by size or not normalized by size. The most known not-normalized EAM is called *PofB* (Chen et al. 2017; Tu et al. 2020; Wang et al. 2020; Xia et al. 2016) which is defined as the proportion of defective entities identified by analyzing the first $x\%$ of the code base as ranked according to their probabilities, as provided by the prediction model, to be defective. The better the ranking, the higher the *PofB*, the higher the support provided during testing. For instance, a method having a *PofB*10 of 30% means that 30% of defective entities have been found by analyzing 10% of the codebase by using the ranking provided by the method.

Since the *PofB*X of a perfect ranking is still costly, it is interesting to compare the ranking provided by a prediction model with a perfect ranking; this helps understanding how the prediction model performed compared to a perfect model. Therefore, Mende and Koschke (2009), as inspired by Arisholm et al. (2007), proposed *Popt* which measures the ranking accuracy provided by a prediction model by taking into account how it is worse than a perfect ranking and how it is better than a random ranking. *Popt* is defined as the area Δ_{opt} between the optimal model and the prediction model. In the optimal model, all instances are ordered by decreasing fault density, and in the predicted model, all instances are ordered by decreasing predicted defectiveness. The equation of computing *Popt* is shown below, where a larger *Popt* value means a smaller difference between the optimal and predicted model: $Popt = 1 - \Delta_{opt}$ (Yu et al. 2019).

Popt and *PofB* are two different metrics describing two different aspects of the accuracy of a model. *Popt* and *PofB* rank entities in two different ways: *Popt* according to bug density (i.e., bug probability divided by entity size), *PofB* according to bug probability. Therefore, the ranking of classifiers provided by *Popt* and *PofB* might differ. Finally, *Popt* is more realistic than *PofB* as the ranking is based on density rather than probability. However, *Popt* is harder to interpret than *PofB* as a classifier with the double of *Popt* does not provide the double of benefits to its user. Thus, in this paper, we try to bring the best of *PofB* and *Popt* by proposing a new EAM metric that ranks entities similarly to both *Popt* and *PofB*.

In the following we provide a description of additional EAMS.

- *Norm(Popt)*: is introduced by Feng et al. (2021) and coincides with *Popt*20
- *PCI@20%* and *PMI@20%*: have been introduced by (Huang et al. 2019) and Chen et al. (2021) respectively and they represent the Proportion of Changes Inspected and Proportion of Modules Inspected, respectively, when 20% LOC are inspected. Note that these metrics are about the ranking of modules in general rather than about the ranking of defective modules. The idea behind these two similar metrics is that context switches shall be minimized to support effective testing. Specifically, a larger *PMI@20%* indicates that developers need to inspect more files under the same volume of LOC to inspect. Thus bug prediction models should strive to reduce *PMI@20%* while trying to increase *Popt* (Qu et al. 2021b) at the same time.
- *PFI@20%*: has been introduced by Qu et al. (2021b) and it coincides with *PMI@20* (Chen et al. 2021) when the module is a file.

- *IFA*: “returns the number of initial false alarms encountered before the first real defective module is found” (Chen et al. 2021). This effort-aware performance metric has been considerably influenced by previous work on automatic software fault location (Kochhar et al. 2016). When IFA is high then there are many false positives before detecting the first defective module. (Chen et al. 2021).
- *Peffort*: has been introduced by D’Ambros et al. (2012) and it is similar to our proposed NPofB. Peffort uses the LOC metric as a proxy for inspection effort. Peffort evaluates a ranking of entities based on the number of predicted defects divided by size whereas our NPofB evaluates a ranking of entities based on the predicted defectiveness divided by size.

2.3 Evaluations

As EAMs drive and impact the results of prediction models evaluations, it is important to discuss studies about how to evaluate prediction models. The evaluation of prediction models performed in studies has been largely discussed.

Many papers explicitly criticized specific empirical evaluations. For instance, Herbold (2017) criticized the use of the ScottKnottESD test in Tantithamthavorn et al. (2016c).

Shepperd et al. (2014) found that the choice of classifier has less impact on results than the researcher group. Thus, they suggest conducting blind analysis, improve reporting protocols, and conduct more intergroup studies. Tantithamthavorn et al. (2016b) replied for a possible explanation for the results aside from researcher’ bias; however, after a few months Shepperd et al. (2018) concluded that the problem of researcher’ bias remains.

Zhang and Zhang (2007) criticized Menzies et al. (2007b) because, due to the small percentage of defective modules, their results are not satisfactory for practical use. Zhang and Zhang (2007) suggest using accuracy metrics, such as Recall and Precision, instead of *pd* or *pf*. Menzies et al. (2007a) replied that it is often required to lower precision to achieve higher recall and that there are many domains where low precision is useful. Menzies et al. (2007a), in contrast to Zhang and Zhang (2007), advised researchers to avoid the use of precision metric; they suggest the use of more stable metrics (i.e., recall (*pd*) and false alarm rates) for datasets with a large proportion of negative (i.e. not defective) instances.

Falessi et al. (2020) reports on the importance of preserving the order of data between the training and testing set. Afterward, the same issue was deeply discussed in Flint et al. (2021) Thus, results are unrealistic if the underlying evaluation does not preserve the order of data.

Falessi et al. (2022) show that dormant defects impact classifiers’ accuracy and hence its evaluation. Specifically, an entity, such as a class or method used in the training/testing set, can be labeled in the ground-truth as defective only after the contained defect is fixed. Since defects can sleep for months or years (Ahluwalia et al. 2019; Chen et al. 2014) then the entity erroneously seems to be not defective until the defect it contains is fixed. Thus, Ahluwalia et al. (2019) suggest to ignore the most recent releases to avoid that dormant defects impact classifiers’ accuracy.

Shepperd et al. (2013) commented on the low extent to which published analyses based on the NASA defect datasets are meaningful and comparable.

Very recently Morasca and Lavazza (2020) proposed a new approach and a new performance metric (the Ratio of Relevant Areas) for assessing a defect proneness model by taking into account only parts of a ROC curve. They also show the differences and how more reliable and less misleading their metric is compared to the existing ones.

3 Study Design

In this paper we investigate the following research questions:

- **RQ1: Which EAMs are used in software engineering journal papers?** In this research question we investigate the trends in EAMs usage, i.e., which and how many EAMs are used in past studies. We are also interested in understanding if the same study uses multiple EAMs and if the EAMs are consistently defined and computed across different studies.
- **RQ2: Does the normalization improve PofBs?** In this research question we investigate if the normalization of PofBs brings higher accuracy. Higher accuracy means that if we analyze a percent of lines of code of the possibly defective entities, we cover a high number of defective entities following a ranking that is based on both the entities likelihood (to be defective) and its size rather than a ranking that is based only on the entities likelihood. If the normalization of PofBs brings higher accuracy, then studies reporting EAMs, unlike our normalized EAMs, underestimate the benefits of using a classifier for ranking defective classes. Moreover, the normalized EAMs shall be considered more realistic than EAMs since they relate to better classifiers.
- **RQ3: Does the ranking of classifiers change by normalizing PofBs?** In this research question we are interested in understanding if the best classifier of a PofB is also the best classifier of NPofB; i.e. if a classifier results as best in PofB10 then it might not be the best in NPofB10. Suppose the normalization changes the ranking of classifiers. In that case, past studies using PofB are misleading, i.e., past studies might not identify the classifier providing the highest benefit to the user in ranking defective classes.
- **RQ4: Does the ranking of classifiers change across normalized PofBs?** In this research question we are interested in understanding if multiple NPofBs are needed to support a comprehensive understanding of classifier accuracy. In other words, we want to know if different NPofBs rank classifiers in the same way. If different NPofBs rank classifiers differently, then results related to a single NPofB cannot be generalized to the overall ranking effectiveness provided by classifiers; i.e. if a classifier resulted as best in NPofB20 then it might not be the best in NPofB10.

3.1 RQ1: Which EAMs are Used in Software Engineering Journal Papers?

To investigate the trends in EAMs usage, we carried out a mapping study (MS) in the first semester of 2021 by following the Kitchenham and Charters guidelines (Kitchenham and Charters 2007).

We performed the MS by applying the following query in the tile:

(bug OR defect) AND (prediction OR estimation)

To make the MS feasible to our effort constraints we focused on the top five journals in the software engineering areas: IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, Empirical Software Engineering and Measurement, Journal of Systems and Software, and Information and Software Technology.

We excluded conferences since they pose space constraints. Specifically, we wanted to be sure that a limited use of EAM was a deliberate design decision of the authors rather than a decision to meet the (conference) space constraints.

Our search provided us a set of about 179 papers. Then we applied the following exclusion criteria:

- Comments and answer to comments kind of papers.
- Systematic and mapping study kind of papers.
- Practitioners' opinions kind of papers.
- Studies about models predicting things other than defectiveness such as ticket resolution time.

After applying the exclusion criteria, we focused the remainder of the MS on 152 primary studies.

Once we applied the above-mentioned exclusion criteria for each paper, we checked the name of the EAMs used and their definition (i.e., how it was computed). Thus, we started from an empty list of EAMs and we improved the list as we analyzed the papers. The data extracting and synthesis of all papers have been performed by both authors independently after a period of training on a small set of papers. The results of the authors perfectly coincided.

3.2 RQ2: Does the Normalization Improve PofBs?

In general, EAMs try to measure the ranking effectiveness of prediction models. The rationale behind EAMs is to measure the effort required by testers to find a specific percent of defects by following a ranked set of entities possibly containing defects. Since the testing effort varies according to the size of the entities under test, we had the intuition that the ranking of entities, is more effective if it takes into consideration both the likelihood of the entity to be defective and also its size. Therefore, in this paper we propose and validate a new EAM that measures the ranking effectiveness of prediction models when the ranking is normalized by the size of the ranked entities; i.e., it measures the effectiveness of an effort-aware ranking.

To investigate if the normalization increases PofBs we perform an empirical study based on within-project across-release class-level defect prediction. Specifically, we observe if the PofB of the same classifier on the same dataset increases after the normalization. As datasets we use the same two industry projects and 12 open-source projects we successfully used in a recent study (Falessi et al. 2020). The 12 open-source projects have been originally used by Tantithamthavorn et al. (2016c) which in turn have been selected from a set of 101 publicly-available defect datasets.

We refer to the recent study for details about the size and characteristics of the projects.

3.2.1 Independent Variable

The independent variable of this research question is the presence or absence of normalization in computing PofBs. In this study, we use the term, feature, to refer to the input (e.g., CHURN) of a classifier. Our independent variable is the normalization of the ranking by size as this is what we conjecture influences the ranking effectiveness. We note that in some studies, that are different from the present one, the features are the independent variables.

3.2.2 Dependent Variables

The dependent variable of this research question is the score of PofB with and without the normalization. As PofBs we considered the spectrum from 10 to 90 with a step of 10. We neglected PofB0 since this is always zero and PofB100 since this is always 100. We also considered the AveragePofB as computed as the average between the PofBs from 0 to 100 with a step of 10. Thus we considered ten different PofBs.

In addition to comparing the two scores, with versus without the normalization, in this paper we observe the relative gain provided by the normalization as defined as $\frac{(NPofB - PofB)}{PofB}$ where NPofB represents the normalized score of PofB.

3.2.3 Measurement Procedure

For each project, we:

1. Perform preprocessing:
 - Normalization: we normalize the data with log10 as performed in a related study (Jiang et al. 2008; Tantithamthavorn et al. 2019).
 - Feature Selection: we filter the independent variables described above by using the correlation-based feature subset selection (Ghotra et al. 2017; Hall 1998; Kondo et al. 2019). The approach evaluates the worth of a subset of features by considering the individual predictive capability of each feature, as well as the degree of redundancy between different features. The approach searches the space of feature subsets by a greedy hill-climbing augmented with a backtracking facility. The approach starts with an empty set of features and performs a forward search by considering all possible single feature additions and deletions at a given point.
 - Balancing: we apply SMOTE (Agrawal and Menzies 2018; Chawla et al. 2002) so that each dataset is perfectly balanced.
2. Create the Train and Test datasets by adopting the above walk-forward validation technique. Specifically, our context is the within-project across-release class-level defect prediction. As a measurement procedure, we adopt the walk-forward validation technique suggested in a recent study (Falessi et al. 2020). In this technique, the project is first organized in releases. Afterwards, there is a loop for $n = 2, n + 1, \dots, up\ to\ n = max\ releases$ where the data of the initial $n-1$ releases is used as training set, and the data of the last n release is used as testing set. This technique has the advantage of preserving the order of data and hence avoiding that data from the future is used to predict data in the past. Moreover, the technique is fully replicable as there is no random mechanism. The disadvantage is that it requires the project to have at least two releases. The random aspects in our classifiers, if any, are controlled by seeds that are used as a parameter, i.e., input, of the classifiers. Therefore, our classifiers are deterministic rather than stochastic, i.e., our results coincide over multiple runs on the same train-test pair. Thus, there is no need to perform a sensitivity analysis of our results. Our set of 14 projects, analyzed via a walk-forward technique, leads to a total of 71 datasets (i.e., 71 specific combinations of training and testing sets). For instance, since KeymindA consists of five releases, then walk-forward on KeymindA leads to 4 datasets. Again, we forward the reader to the previous study for further details about the datasets (Falessi et al. 2020).
3. Compute predicted probability of defectiveness of each class by using each of the ten classifiers.
4. Compute PofBs and NPofBs.

As classifiers we used the ones used in a previous study (Falessi et al. 2020):

- Decision Table: Two major parts: schema, the set of features included in the table, and a body, labeled instances defined by features in the schema. Given an unlabeled instance, try matching instance to record in the table (Kohavi 1995).

- IBk: Also known as the k-nearest neighbor's algorithm (k-NN), which is a non-parametric method. The classification is based on the majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (Altman 1992). K-nearest neighbors classifier run with $k = 1$ (Aha and Kibler 1991).
- J48: Generates a pruned C4.5 decision tree (Quinlan 1993).
- KStar: Instance-based classifier using some similarity function. Uses an entropy-based distance function (Cleary and Trigg 1995).
- Naive Bayes: Classifies records using estimator classes and applying Bayes theorem (John and Langley 1995) i.e., it assumes that the contribution of an individual feature towards deciding the probability of a particular class is independent of other features in that project instance (McCallum and Nigam 1998).
- SMO: John Platt's sequential minimal optimization algorithm for training a support vector classifier (Platt 1998)
- Random Forest: Ensemble learning creating a collection of decision trees. Random trees correct for overfitting (Breiman 2001).
- Logistic Regression: It estimates the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables. The estimation is performed through the logistic distribution function (Le Cessie and Van Houwelingen 1992).
- BayesNet: Bayesian networks (BNs), also known as belief networks (or Bayes nets for short), belong to the family of probabilistic graphical models (GMs). These graphical structures are used to represent knowledge about an uncertain domain. In particular, each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables. These conditional dependencies in the graph are often estimated by using known statistical and computational methods. Hence, BNs combine principles from graph theory, probability theory, computer science, and statistics (Ben-Gal 2008).
- Bagging: Probably the most well-known sampling approach. Given a training set, bagging generates multiple bootstrapped training sets and calls the base model learning algorithm with each of them to yield a set of base models (Kotsiantis et al. 2005).

3.2.4 Analysis Procedure

We compare the value of PofBs of the same classifier on the same dataset, with versus without the normalization.

Since our data strongly deviate from normality, the hypotheses of this research question are tested using the Wilcoxon signed-rank test (Wilcoxon 1945). The test is paired since the compared distributions, with versus without normalization, are related to the identical objects (i.e., the score of the same ten classifiers, ten classifiers, on the same 71 datasets). We also use the Cliff's delta (paired) to analyze the effect size (Grissom and Kim 2005). Table 3 presents the standard interpretation (Vargha and Delaney 2000) of Cliff's delta (paired) effect size.

3.3 RQ3: Does the Ranking of Classifiers Change by Normalizing PofBs?

Since the normalization of PofBs results in higher accuracy (RQ2), it is interesting to understand the validity of past studies since they do not normalize PofBs. Suppose the

Table 3 Interpretation of Cliff's delta effect size

Cliff's delta value	Interpretation
≥ 0.11	Small
≥ 0.28	Medium
≥ 0.43	Large

normalization changes the ranking of prediction models. In that case, past studies using PofBs are misleading, i.e., past studies might not identify the prediction model providing the highest benefit to the user.

The dependent variable is the rank of classifiers. The independent variable is the presence or absence of the PofB normalization.

In this research question, we leverage RQ2 results, i.e., the accuracy of 10 classifiers over 72 datasets grouped in 14 projects. To compare the rankings, we use the Spearman's rank correlation (Spearman, 1904) between the ranking of classifiers provided by the same PofB, with versus without the normalization, in each of the 72 datasets. To compare the rankings we use the Spearman's rank correlation (Spearman 1904) between the ranking of classifier provided by the same PofB, with versus without the normalization. Table 4 presents the standard interpretation (Akoglu 2018) of Spearman's ρ .

We also compare, for each dataset and PofB, if the best classifier coincides after the normalization.

3.4 RQ4: Does the Ranking of Classifiers Change Across Normalized PofBs?

Since past studies used a very limited set of EAMs (RQ1), it is interesting to understand if it is a valid design decision to use a limited set of NPofBs. Suppose different NPofBs rank classifiers differently. In that case, the results related to a single NPofB cannot be generalized to the overall ranking effectiveness provided by prediction models; i.e. if a prediction model resulted as best in NPofB20 then it might not be the best in NPofB10. Thus, to understand if the use of multiple NPofBs is needed, we need to understand if there is a difference among NPofBs. We measure the difference among NPofBs as the difference among their rankings. To compare the rankings we use the Spearman's rank correlation (Spearman 1904) between the ranking of classifiers provided by each pair of NPofBx, with x in the range [10, 90]. As in RQ3, in this research question we leverage RQ2 results. Specifically, each classifier, in each of the 72 datasets, has a ranking in the range [1,10] (as we used 10 classifiers) with a specific PofBx. We compute the Spearman's values across each combination of NPofBx, with x in the range [10, 90]. We also compare, for each dataset, the proportion of ten NPofBs sharing the same classifier as best.

Table 4 Interpretation of Rho Spearman's values

ρ Spearman's value	Interpretation
< 0.6	Fair
< 0.8	Moderate
< 0.9	Very strong
$= 1$	Perfect

4 Study Results

4.1 RQ1: Which EAMs are Used in Software Engineering Journal Papers?

Table 5 reports the EAMs used in software engineering journal papers. According to Table 5 the most used EAM is AveragePopt and PofB20.

Table 6 reports the number of EAMs used in past studies. According to Table 6 the majority of the studies used no EAM and hence ignored to validate the model according to their impact on effort. Moreover, the majority of studies using EAMs used a single EAM (i.e., 12 out of 20).

Table 7 reports the number of studies correctly or incorrectly naming EAMs according to their original definitions (Chen et al. 2017; Mende and Koschke 2009). According to Table 7 seven out of 20 studies incorrectly named EAM.

4.2 RQ2: Does the Normalization Improve PofBs?

Figure 1 reports ten PofBs, and their normalization, of 10 classifiers over the 72 datasets grouped in 14 projects. Figure 2 reports the gain achieved by normalizing a specific PofB metric. According to Fig. 2 the normalization increases the performance of the median classifier of all PofBs in all 14 projects.

Table 8 reports the average gain, across datasets and classifiers, in normalizing a specific EAM. According to Table 8 the relative gain doubles when decreasing the PofB metric; e.g., the relative gain in PofB10 is double than PofB20, which is the double of PofB30.

Table 8 reports the statistical test results comparing a PofB before and after the normalization. According to Table 8, the normalization significantly improves all ten PofB metrics. Therefore, we can reject H10 for all ten EAMs. Moreover, the effect size resulted as large for all ten EAMs.

4.3 RQ3: Does the Ranking of Classifiers Change by Normalizing PofBs?

Table 9 reports the correlation between the same PofB before and after the normalization. According to Table 9 the correlation is only fair between the same PofB before and after the normalization in nine out of 10 PofBs.

Figure 3 reports the proportion of times the same PofB metrics, with and without normalization, identifies the same classifier as best. According to Fig. 3, the proportion of times the same PofB metrics, with and without normalization, identifies the same classifier as best changes across projects and PofBs. Specifically, in half of the datasets, the best classifier changes in all PofBs after the normalization. Table 10 summarizes Fig. 3 by reporting in average per specific PofBs, the proportion of times the same PofB metrics, with and without normalization, identifies the same classifier as best. According to Table 10, in all ten PofBs the proportion of times that the same PofB metrics, with and without normalization, identifies the same classifier as best is less than half. Thus, the best classifier is more likely to change than to coincide when considering PofB after the normalization.

Table 5 EAM used in past studies

Popt20	Popt30	Popt40	AveragePopt	PCI20	PofB20	Peffort	IFA	Norm(Popt)	PMI@20%	PFI@20%
7	1	1	8	1	7	1	4	1	1	1

Table 6 Number of EAM used in past studies

# EAM	0	1	2	3	4	≥ 5
# PS	132	12	5	2	1	0

Table 7 Number of studies correctly or incorrectly naming EAM

# PS Correctly naming EAM	13
# PS Incorrectly naming EAM	7

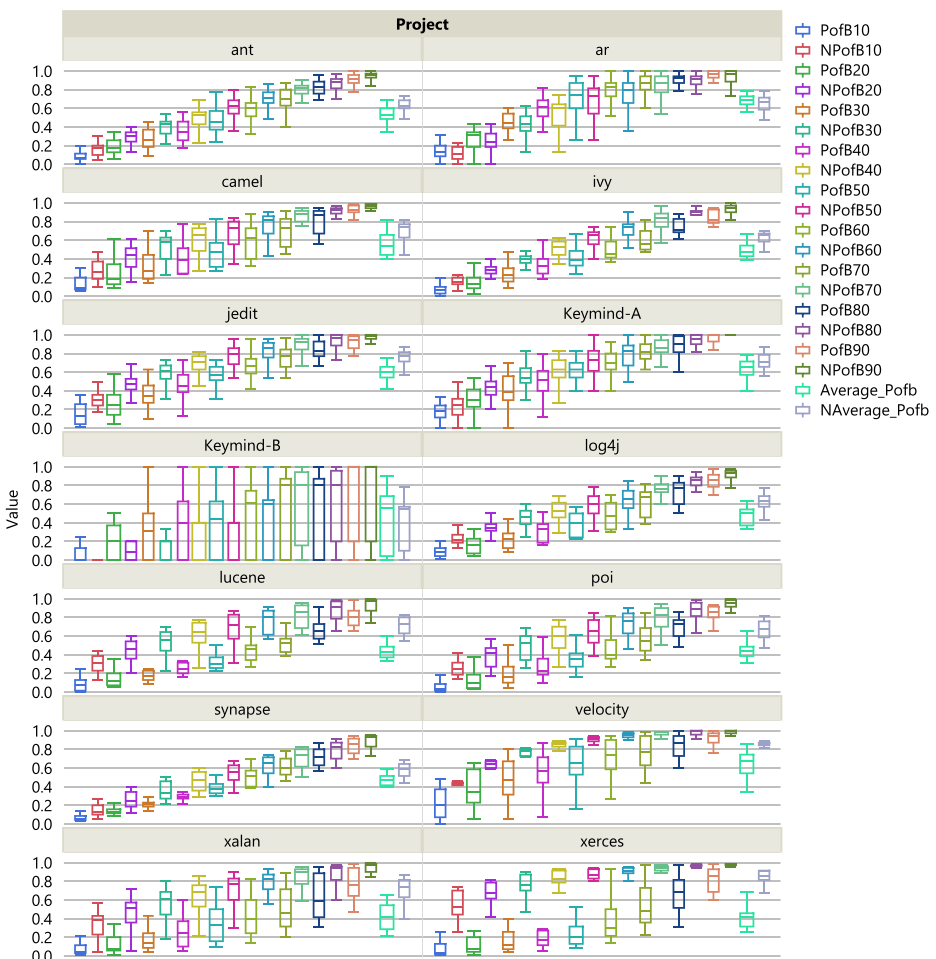


Fig. 1 PofBs, and their normalization (NPofBs), of 10 classifiers over the 14 projects

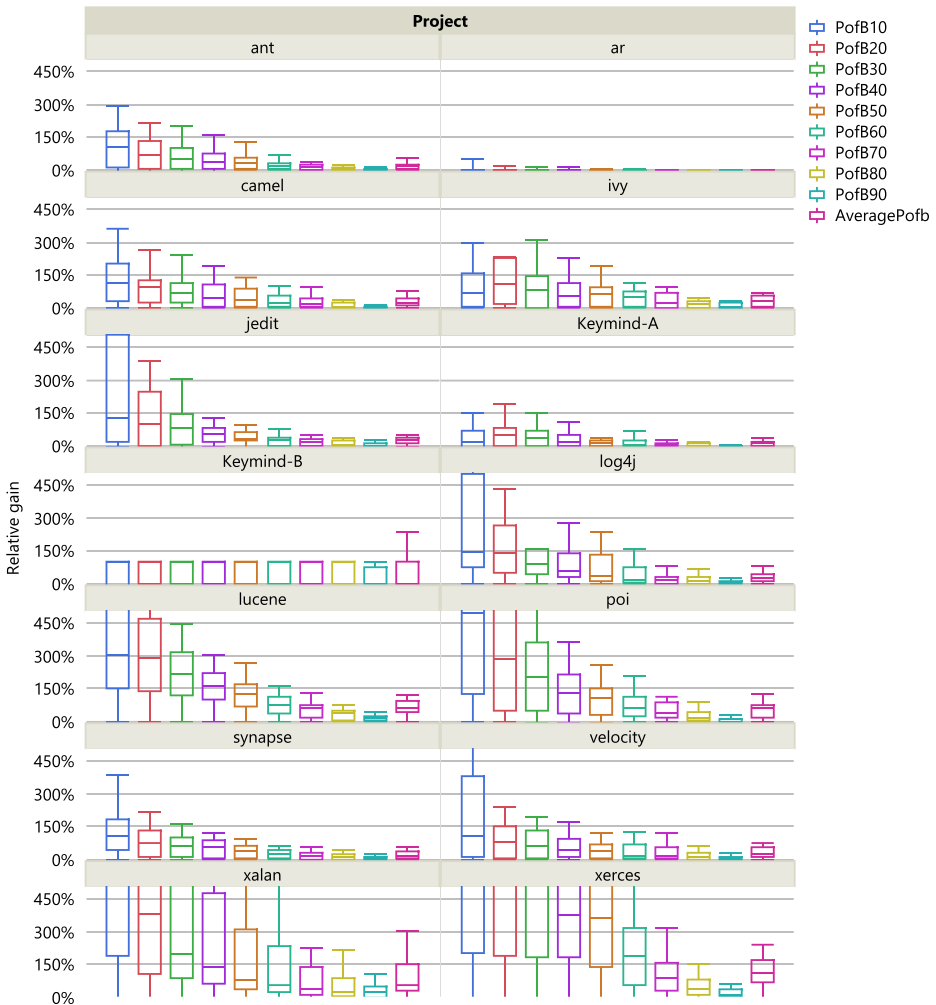


Fig. 2 Average by project of the relative gain in normalizing a specific PofB

4.4 RQ4: Does the Ranking of Classifiers Change Across Normalized PofBs?

Table 12 in the appendix reports the correlation among each couple of normalized PofBs. To better summarize Table 12, Fig. 4 reports the distribution of the correlations among each couple of NPofBs. Table 11 reports the frequency of interpretations of correlation values among couples of NPofBs. According to Table 11 no couple of NPofBs is perfectly correlated. Moreover, only 30 out of 45 couples of NPofBs are very strongly correlated. In conclusion, Table 11 shows that the rankings of classifiers are far to be identical across different NPofBs.

Figure 5 reports the distribution of the proportion of times the best classifier for a dataset coincides across NPofBs. According to Fig. 5, in only five out of 41 datasets the best classifier for a dataset coincides across the ten PofBs. Thus, in about 88% of the cases, the best classifier varies across NPofBs.

Table 8 Relative gain, statistical test and Cliff's delta results comparing a PofB metric with its normalization

Compared metrics	Relative gain	Test statistic S	Pvalue	Cliff's delta
PofB10 vs. NPofB10	414%	31267	<0.0001	0.5804
PofB20 vs. NPofB20	224%	32089	<0.0001	0.5726
PofB30 vs. NPofB30	147%	33804	<0.0001	0.5588
PofB40 vs. NPofB40	107%	34122	<0.0001	0.5497
PofB50 vs. NPofB50	78%	34119	<0.0001	0.5217
PofB60 vs. NPofB60	51%	33533	<0.0001	0.4965
PofB70 vs. NPofB70	35%	33700	<0.0001	0.4695
PofB80 vs. NPofB80	23%	32107	<0.0001	0.4318
PofB90 vs. NPofB90	11%	31094	<0.0001	0.3540
Average_Pofb vs. NAverage_Pofb	38%	34446	<0.0001	0.5515

5 Discussion

This section discusses our main results, the possible explanations for the results, implications, and guidelines for practitioners and researchers.

5.1 RQ1: Which EAMs are Used in Software Engineering Journal Papers?

The main result of RQ1 is that EAMs are used in a minority of defect prediction studies, i.e., 20 out of 152 software engineering journal papers. One possible reason is that EAMs do not make much sense in Just-in-time prediction studies, i.e., in studies predicting the defectiveness of commits. As a matter of fact, in the JIT context, the user is envisioned to consider the defectiveness prediction just after each single commit, and hence a JIT classifier cannot help the user in ranking the possibly defective entities (i.e., commits). However, we observed many JIT studies using EAMs and many non-JIT studies, i.e., studies predicting a class's defectiveness or method, not using EAMs. One possible reason for the low EAMs usage in non-JIT studies is the absence of a tool for EAMs computation. A further possible reason for the low EAMs adoption could be the lack of awareness about the importance of EAMs to evaluate the realistic benefits of using prediction models.

Table 9 Correlation between the same PofB before and after the normalization

Compared metrics	Rho	Pvalue	Intepretation
PofB10 vs. NPofB10	0.121	<0.0001	Fair
PofB20 vs. NPofB20	0.112	<0.0001	Fair
PofB30 vs. NPofB30	0.219	<0.0001	Fair
PofB40 vs. NPofB40	0.246	<0.0001	Fair
PofB50 vs. NPofB50	0.316	<0.0001	Fair
PofB60 vs. NPofB60	0.349	<0.0001	Fair
PofB70 vs. NPofB70	0.410	<0.0001	Fair
PofB80 vs. NPofB80	0.440	<0.0001	Fair
PofB90 vs. NPofB90	0.607	<0.0001	Moderate
Average_PofB vs. NAverage_PofB	0.245	<0.0001	Fair

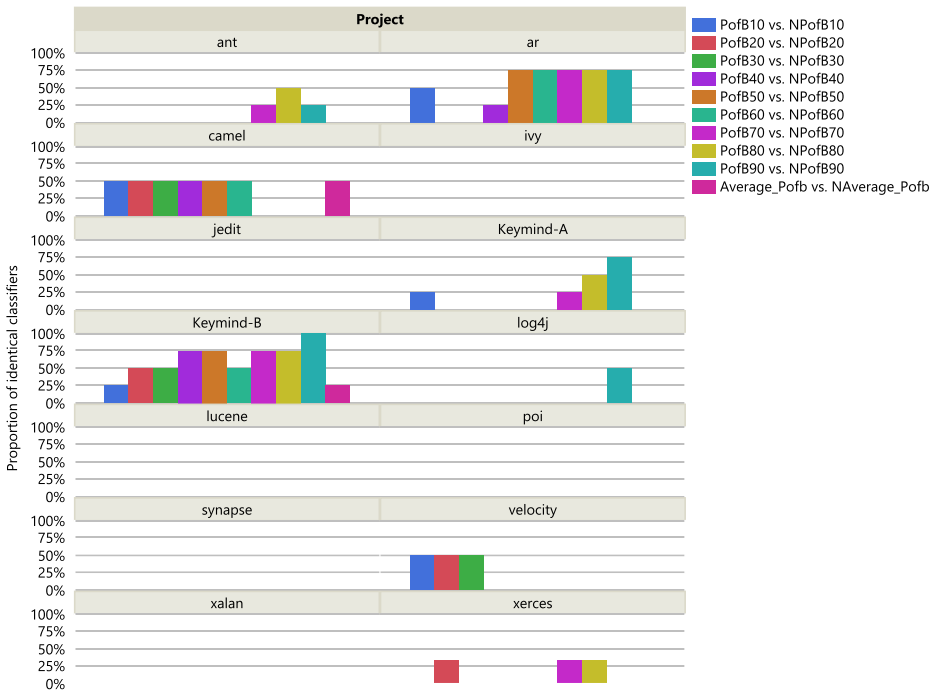


Fig. 3 Proportion of times the same PofB metric, with and without normalization, identifies the same classifier as best

Another important result of RQ1 is that some EAMs are misnamed and that only one study used more than one EAM. Again, one possible reason for this result is the absence of a tool for EAMs computation.

The main implication of RQ1 is that a tool to automate EAMs computation would have supported a broader and more correct use of EAMs.

Table 10 Proportion of times the same PofB metrics, with and without normalization, identifies the same classifier as best

Compared metrics	Proportion of equivalent best classifiers
PofB10 vs. NPofB10	15%
PofB20 vs. NPofB20	12%
PofB30 vs. NPofB30	10%
PofB40 vs. NPofB40	12%
PofB50 vs. NPofB50	17%
PofB60 vs. NPofB60	15%
PofB70 vs. NPofB70	22%
PofB80 vs. NPofB80	27%
PofB90 vs. NPofB90	29%
Average_PofB vs. NAverage_PofB	5%

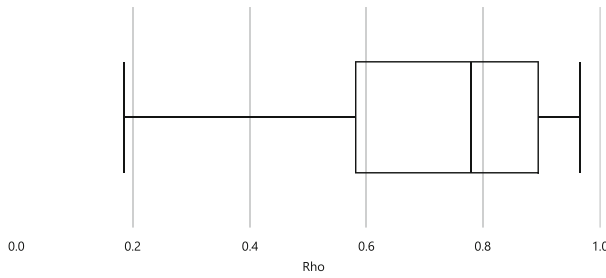


Fig. 4 Distribution of correlations among each couple of NPofBs

Table 11 Frequency of interpretations of correlations among couples of NPofBs

Interpretation	#
Inverse	0
Fair	7
Moderate	8
Very strong	30
Perfect	0

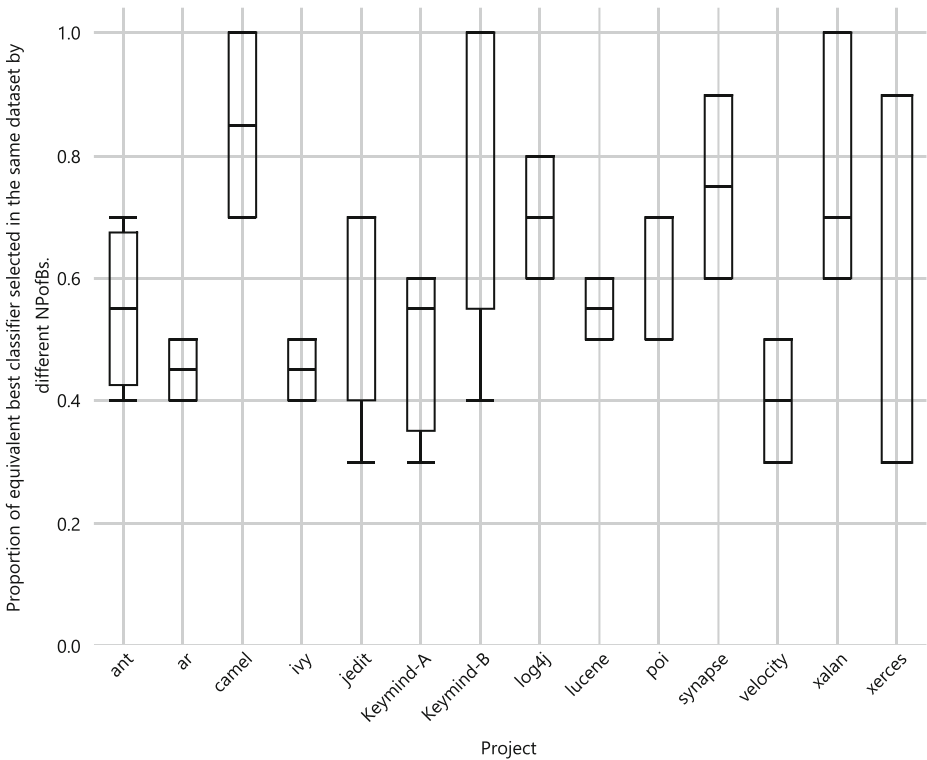


Fig. 5 Distribution of the proportion of datasets where the best classifier coincides across NPofBs

5.2 RQ2: Does the Normalization Improve PofBs?

The main result of RQ2 is that the proposed normalization increases statistically and of orders of magnitude the PofBs. While the improvement is statistically significant on all 10 PofBs, we can see that the normalization increased the different PofBs differently. Specifically, the relative gain provided by the normalization resulted perfectly inversely correlated with the percent of analyzed LOC related in the specific PofB; the highest gain was observed in PofB10. This result can be explained by the fact that the ranking quality loses benefit while the number of analyzed entities are high. In our case, it is obvious that when considering PofB90 many rankings can result equally beneficial to the user as long as the 10% of the not analyzed LOC are shared across such rankings. Thus, a better ranking is more visible in PofB10 than in PofB90.

We also note that the relative gain was higher in some datasets, i.e., xerces, than others, i.e., ar. The most probable reason is that the range of the relative gain is large when the accuracy without the normalization is low. Specifically, the PofBs in ar are much higher than in xerces. In other words, the normalization in ar has less chance of improving the PofBs since it is already high.

The main implication of RQ2 is that we need to use the normalized PofBs (aka NPofBs) rather than PofBs. The NPofBs are more realistic than PofBs as they relate to better use of classifiers. Moreover, studies reporting PofBs, rather than its normalization, underestimate the benefits of using classifiers for ranking defective classes and hence might have hindered the practical adoption of defect classifiers.

5.3 RQ3: Does the Ranking of Classifiers Change by Normalizing PofBs?

The main result of RQ3 is that the normalization changes the ranking of classifiers. Specifically, the correlation is only fair between the same PofB before and after the normalization in nine out of 10 PofBs. Moreover, in more than half of cases a classifier resulting as best with a PofB is not best with its normalization. The main implication of RQ2 is that past studies using the not normalized version of PofB likely highlighted a classifier as best despite another classifier brings the highest benefit in ranking defective classes to the user. Hence, RQ3 results call for replication of past studies using the not normalized version of PofBs.

5.4 RQ4: Does the Ranking of Classifiers Change Across Normalized PofBs?

The main result of RQ4 is that no couple of NPofBs is perfectly correlated. Moreover, in 88% of the cases, the best classifier varies across the ten considered NPofBs. Thus, the main implication of RQ4 is that a single EAM does not exhaustively capture classifiers' ability to rank defective candidate classes. Thus, researchers shall use multiple EAMs to support a comprehensive understanding of classifiers accuracy. Past studies that validated classifiers via a single EAM shall be replicated to increase their results generalizability.

6 The ACUME Tool

In this paper we provide a new tool called ACUME (ACcUracY MEtrics) which can compute PofB, the new proposed NPofB, Popt, IFA, and the performance metrics reported in Section 2.

In order to run, ACUME requires to:

- Download project files from GitHub;
- Place the csv files of the whole dataset in the data folder, and if needed in the test folder place your test files;
- Update configs.py file to your needs by following the instructions of ReadMe.md file.

In order to run the code, you need python installed; more details are presented in the Readme file.

ACUME has been developed with clean code principles, kiss principle, and functional/OOP4 programming. There is linear complexity and minimal repetition of calculation to achieve a fast script with minimal memory usage by sacrificing readability for efficiency. Within approximately 1000 lines of code, there are two data classes: class DataEntity (no function) and ProcessedDataEntity (10 functions/methods), six stand-alone models—not class-dependent as well as many helper functions in each class.

Figure 6 reports the steps of the process on which ACUME works:

1. Step 1: The research teams create or reuse datasets. A name identifies the different datasets, e.g., KeymindA.csv, ant.csv.
2. Step 2: The research team provides the datasets to a ML engine, such as Weka. The ML engine applies one or more defect prediction models that vary in classifiers, feature selection, balancing, etc.
3. Step 3: The ML engine outputs the predicted file. The predicted file is designed to be as simple as possible; for each predicted entity (e.g., a class), the predicted file reports the ID, the size, the predicted probability to be defective, and the actual defectiveness. The different predicted files are identified simply with a name, e.g., KeymindA.RandomForest.withFeatureSelection.csv, ant.RandomForest.withFeatureSelection.csv, etc.
4. Step 4: The research team provides the predicted files to the ACUME tool.
5. Step 5: The ACUME tool outputs a single CSV file reporting, for each row, the performances of a predicted file in terms of accuracy metrics and EAMs.

We provide online material about ACUME.¹

ACUME has been used and validated by several researchers and students at the University of Rome Tor Vergata. We tested ACUME using a set of unit tests hence comparing the accuracy metrics computed by ACUME with expected values. To compute the expected values, we used a mixed approach according to metrics under test. Specifically, for metrics available in WEKA, such as AUC, we computed the expected values via WEKA on the project breast-cancer, as natively provided in WEKA. For metrics not available in WEKA, such as EAMs, we computed the expected values via Excels formulas on the ten projects used to address the research questions in this paper. The validation was led by the first author and double-checked by the last author. During the validation process we fixed a few bugs related to the measurement of the AUC metric. The validation folder in the replication package reports the validation artifacts.

Finally, since its code is open, we welcome bug reports and feature requests.

7 Threats to Validity

In this section, we report the threats to validity of our study. The section is organized by threat type, i.e., Conclusion, Internal, Construct, and External.

¹<https://github.com/jonidacarka/ACUME.git>

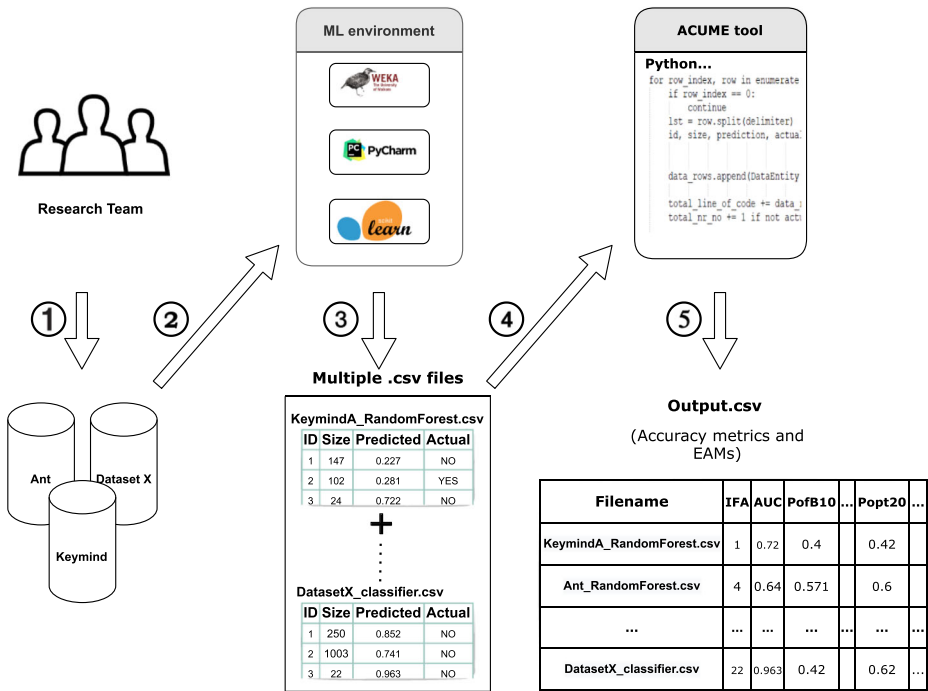


Fig. 6 Distribution of the correlation among each couple of EAMs

7.1 Conclusion

Conclusion validity concerns issues that affect the ability to draw accurate conclusions regarding the observed relationships between the independent and dependent variables (Wohlin et al. 2012).

We tested all hypotheses with a non-parametric test (e.g., Wilcoxon Signed Rank) (Wilcoxon 1945) which is prone to type-2 error, i.e., not rejecting a false hypothesis. We have rejected the hypotheses in all cases; thus, the likelihood of a type-2 error is null. Moreover, the alternative would have been using parametric tests (e.g., ANOVA) that are prone to type-1 error, i.e., rejecting a true hypothesis, which is less desirable than type-2 error in our context.

7.2 Internal

Internal validity concerns the influences that can affect the independent variables concerning causality (Wohlin et al. 2012). A threat to internal validity is the lack of ground truth for class defectiveness, which could have been underestimated in our measurements. To avoid this threat, we used a set of projects already successfully used in our recent study (Falessi et al. 2020). Such datasets have been derived from a set of publicly available datasets which have been originated far before many issues were known, including mislabeling (Bird et al. 2009; Herzig et al. 2013; Kim et al. 2011; Rahman et al. 2013; Tantithamthavorn et al. 2015), snoring (Falessi et al. 2022), and wrong origin (Rodríguez-Pérez et al. 2018b). Thus, despite being publicly available and largely used, our datasets might be inaccurate.

As many other studies on defect prediction (Falessi et al. 2020, 2022; Fukushima et al. 2014; Tantithamthavorn et al. 2016c, 2019; Turhan et al. 2009; Vandehei et al. 2021); we do not differentiate among severity levels of the (predicted) defects.

7.3 Construct

Construct validity is concerned with the degree to which our measurements indeed reflect what we claim to measure (Wohlin et al. 2012).

In order to make our empirical investigation reliable, we used the walk-forward technique as suggested in our recent study (Falessi et al. 2020). It could be that our results are impacted by our specific design choices, including classifiers, features, and accuracy metrics. In order to face this threat, we based our choice on past studies.

Despite many studies suggest the tuning of hyperparameters (Fu et al. 2016; Tantithamthavorn et al. 2019), we used default hyperparameters due to resource constraints and to the static time-ordering design of our evaluation. Moreover, tuning might be relevant in studies aiming at improving the accuracy of classifiers whereas in this study tuning might be irrelevant as we aim at measuring the accuracy of classifiers, regardless of their tuning status. Finally, as our paper suggests, we plan to validate hyperparameters tuning via multiple and normalized PofBs.

Finally, the labeling of entities as defective or not has been subject to significant effort, and we still do not know how to perfectly label entities (Vandehei et al. 2021). To avoid this type of noise in the data, we used data coming from the literature and largely used in the past (Falessi et al. 2020).

7.4 External

External validity is concerned with the extent to which the research elements (subjects, artifacts, etc.) are representative of actual elements (Wohlin et al. 2012).

This study used a large set of datasets and hence could be deemed of high generalization compared to similar studies. Moreover, in this study, we used both open-source and industry-type of projects.

Finally, to promote reproducible research, all datasets, results and scripts for this paper are available online¹.

8 Conclusion

Despite the importance of EAMs, there is no study investigating EAMs usage trends and validity. Therefore, in this paper, we analyze trends in EAMs usage in the software engineering literature. Our systematic mapping study found 152 primary studies (referenced in the appendix, Section A.1) in major software 630 engineering journals, and it shows that most studies using EAMs use only a single EAM 631 (e.g., Popt or PofB20) and that some studies mismatched EAMs names.

To improve the internal validity of results provided by PofBs, we proposed normalization of PofBs. The normalization is based on ranking the defective candidates by the probability of the candidate to be defective divided by its size. We validated the normalization of PofBs by analyzing 10 PofBs, 10 classifiers, two industrial projects and 12 open-source projects. Our results show that the normalization increases statistically and of orders of magnitude the PofBs. Thus, past studies reporting PofBs underestimate the benefits of using classifiers

for ranking defective classes and might have hindered the practical adoption of prediction models in the industry. The proposed normalization increases the realism of PofBs values as it relates to better use of classifiers and promotes the practical adoption of prediction models in the industry.

We showed that when considering the same dataset, in most of the cases, the best classifier for a PofB changes when considering the normalized version of that PofB. Thus, past studies that used the non-normalized version of PofBs likely identified the wrong best classifier, i.e., past studies likely identified as best a classifier not providing the highest benefit to the user in ranking defective classes.

In this paper, we also showed that multiple PofBs are needed to support a comprehensive understanding of classifiers accuracy. Thus, we provide a tool to compute EAMs automatically; this aims at supporting researchers in: 1) avoiding extra effort in EAMs computation as there is no available tool to compute EAMs, 2) increasing results reproducibility as the way to compute EAMs is shared across researchers, 3) increasing results validity by avoiding the observed EAMs misnaming and, 4) increase results generalizability by avoiding single EAMs usage.

Researchers and practitioners involved in validating defect prediction models shall always consider using several EAMs. Researchers shall try to propose and evaluate EAMs, or new normalization, that more realistically measure the benefits provided by classifiers when ranking candidate defective entities.

In the future, we plan to replicate the validation of past defect prediction studies that considered a single EAM by including multiple EAMs. Hence, we want to check if the observed best classifier varies when considering multiple EAMs. We also plan to validate past studies by using the proposed normalization of PofBs rather than the original PofBs. Thus, we want to check if the observed best classifier varies when considering the proposed normalization. Finally, we plan to investigate the opinions of developers on EAMs and specifically on the validated new NPofB metric.

Appendix

Table 12 Correlations among specific couples of normalized PofBs

Compared variables	Rho	Pvalue	Intepretation
NAverage_PofB vs. NPofB10	0.783856	<0.0001	Very strong
NAverage_PofB vs. NPofB20	0.85454	<0.0001	Very strong
NAverage_PofB vs. NPofB30	0.946433	<0.0001	Very strong
NAverage_PofB vs. NPofB40	0.964284	<0.0001	Very strong
NAverage_PofB vs. NPofB50	0.958544	<0.0001	Very strong
NAverage_PofB vs. NPofB60	0.922743	<0.0001	Very strong
NAverage_PofB vs. NPofB70	0.855365	<0.0001	Very strong
NAverage_PofB vs. NPofB80	0.809668	<0.0001	Very strong
NAverage_PofB vs. NPofB90	0.537388	<0.0001	Moderate
NPofB20 vs. NPofB10	0.932106	<0.0001	Very strong
NPofB30 vs. NPofB10	0.779776	<0.0001	Very strong
NPofB30 vs. NPofB20	0.846695	<0.0001	Very strong
NPofB40 vs. NPofB10	0.731393	<0.0001	Very strong

Table 12 (continued)

Compared variables	Rho	Pvalue	Intepretation
NPofB40 vs. NPofB20	0.800647	<0.0001	Very strong
NPofB40 vs. NPofB30	0.966495	<0.0001	Very strong
NPofB50 vs. NPofB10	0.651501	<0.0001	Moderate
NPofB50 vs. NPofB20	0.727009	<0.0001	Very strong
NPofB50 vs. NPofB30	0.909338	<0.0001	Very strong
NPofB50 vs. NPofB40	0.95835	<0.0001	Very strong
NPofB60 vs. NPofB10	0.578208	<0.0001	Moderate
NPofB60 vs. NPofB20	0.650734	<0.0001	Moderate
NPofB60 vs. NPofB30	0.823908	<0.0001	Very strong
NPofB60 vs. NPofB40	0.880341	<0.0001	Very strong
NPofB60 vs. NPofB50	0.939176	<0.0001	Very strong
NPofB70 vs. NPofB10	0.486744	<0.0001	Fair
NPofB70 vs. NPofB20	0.562888	<0.0001	Moderate
NPofB70 vs. NPofB30	0.717087	<0.0001	Very strong
NPofB70 vs. NPofB40	0.781574	<0.0001	Very strong
NPofB70 vs. NPofB50	0.850415	<0.0001	Very strong
NPofB70 vs. NPofB60	0.930698	<0.0001	Very strong
NPofB80 vs. NPofB10	0.474829	<0.0001	Fair
NPofB80 vs. NPofB20	0.561707	<0.0001	Moderate
NPofB80 vs. NPofB30	0.684472	<0.0001	Moderate
NPofB80 vs. NPofB40	0.714814	<0.0001	Very strong
NPofB80 vs. NPofB50	0.777562	<0.0001	Very strong
NPofB80 vs. NPofB60	0.844337	<0.0001	Very strong
NPofB80 vs. NPofB70	0.916603	<0.0001	Very strong
NPofB90 vs. NPofB10	0.185601	<0.0001	Fair
NPofB90 vs. NPofB20	0.313019	<0.0001	Fair
NPofB90 vs. NPofB30	0.3962	<0.0001	Fair
NPofB90 vs. NPofB40	0.458927	<0.0001	Fair
NPofB90 vs. NPofB50	0.497088	<0.0001	Fair
NPofB90 vs. NPofB60	0.5854	<0.0001	Moderate
NPofB90 vs. NPofB70	0.71	<0.0001	Very strong
NPofB90 vs. NPofB80	0.739416	<0.0001	Very strong

A.1 Primary Studies

Sistematic Litterature Review Studies

Afric, P., Sikic, L., Kurdija, A. S., & Silic, M. (2020). REPD: source code defect prediction as anomaly detection. *J. Syst. Softw.*, 168, 110641. doi:10.1016/j.jss.2020.110641

Ali, A., Khan, N., Abu-Tair, M. I., Noppen, J., McClean, S. I., & McChesney, I. R. (2021). Discriminating features-based cost-sensitive approach for software defect prediction. *Autom. Softw. Eng.*, 28(2), 11. doi:10.1007/s10515-021-00289-8

Almhana, R., & Kessentini, M. (2021). Considering dependencies between bug reports to improve bugs triage. *Autom. Softw. Eng.*, 28(1), 1. doi:10.1007/s10515-020-00279-2

Amasaki, S. (2020). Cross-version defect prediction: Use historical data, crossproject data, or both? *Empir. Softw. Eng.*, 25(2), 1573–1595. doi:10.1007/s10664-019-09777-8

Andreou, A. S., & Chatzis, S. P. (2016). Software defect prediction using doubly stochastic poisson processes driven by stochastic belief networks. *J. Syst. Softw.*, 122, 72–82. doi:10.1016/j.jss.2016.09.001

Balaram, A., & Vasundra, S. (2022). Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm. *Autom. Softw. Eng.*, 29(1), 6. doi:10.1007/s10515-021-00311-z

Bangash, A. A., Sahar, H., Hindle, A., & Ali, K. (2020). On the time-based conclusion stability of cross-project defect prediction models. *Empir. Softw. Eng.*, 25(6), 5047–5083. doi:10.1007/s10664-020-09878-9

Bell, R. M., Ostrand, T. J., & Weyuker, E. J. (2013). The limited impact of individual developer data on software defect prediction. *Empir. Softw. Eng.*, 18(3), 478–505. doi:10.1007/s10664-011-9178-4

Bennin, K. E., Keung, J. W., & Monden, A. (2019). On the relative value of data resampling approaches for software defect prediction. *Empir. Softw. Eng.*, 24(2), 602–636. doi:10.1007/s10664-018-9633-6

Bennin, K. E., Keung, J., Phannachitta, P., Monden, A., & Mensah, S. (2018). MAHAKIL: diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Trans. Software Eng.*, 44(6), 534–550. doi:10.1109/TSE.2017.2731766

Bowes, D., Hall, T., & Gray, D. (2014). Dconfusion: A technique to allow cross study performance evaluation of fault prediction studies. *Autom. Softw. Eng.*, 21(2), 287–313. doi:10.1007/s10515-013-0129-8

Chang, C., Chu, C., & Yeh, Y. (2009). Integrating in-process software defect prediction with association mining to discover defect pattern. *Inf. Softw. Technol.*, 51(2), 375–384. doi:10.1016/j.infsof.2008.04.008

Chen, H. [Haowen], Jing, X., Li, Z., Wu, D., Peng, Y., & Huang, Z. (2021). An empirical study on heterogeneous defect prediction approaches. *IEEE Trans. Software Eng.*, 47(12), 2803–2822. doi:10.1109/TSE.2020.2968520

Chen, X., Mu, Y., Liu, K., Cui, Z., & Ni, C. (2021). Revisiting heterogeneous defect prediction methods: How far are we? *Inf. Softw. Technol.*, 130, 106441. doi:10.1016/j.infsof.2020.106441

Chen, X., Zhang, D., Zhao, Y., Cui, Z., & Ni, C. (2019). Software defect number prediction: Unsupervised vs supervised methods. *Inf. Softw. Technol.*, 106, 161–181. doi:10.1016/j.infsof.2018.10.003

Chen, X., Zhao, Y., Wang, Q., & Yuan, Z. (2018). MULTI: multi-objective effort-aware just-in-time software defect prediction. *Inf. Softw. Technol.*, 93, 1–13. doi:10.1016/j.infsof.2017.08.004

D'Ambros, M., Lanza, M., & Robbes, R. (2012). Evaluating defect prediction approaches: A benchmark and an extensive comparison. *Empir. Softw. Eng.*, 17(4–5), 531–577. doi:10.1007/s10664-011-9173-9

Dalla Palma, S., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2021). Within-project defect prediction of infrastructure-as-code using product and process metrics. *IEEE Transactions on Software Engineering*, 1–1. doi:10.1109/TSE.2021.3051492

- Devine, T. R., Goseva-Popstojanova, K., Krishnan, S., & Lutz, R. R. (2016). Assessment and cross-product prediction of software product line quality: Accounting for reuse across products, over multiple releases. *Autom. Softw. Eng.*, 23(2), 253–302. doi:10.1007/s10515-014-0160-4
- dos Santos, G. E., Figueiredo, E., Veloso, A., Viggiano, M., & Ziviani, N. (2020). Understanding machine learning software defect predictions. *Autom. Softw. Eng.*, 27(3), 369–392. doi:10.1007/s10515-020-00277-4
- Ekanayake, J., Tappolet, J., Gall, H. C., & Bernstein, A. (2012). Time variance and defect prediction in software projects - towards an exploitation of periods of stability and change as well as a notion of concept drift in software projects. *Empir. Softw. Eng.*, 17(4–5), 348–389. doi:10.1007/s10664-011-9180-x
- Eken, B., & Tosun, A. (2021). Investigating the performance of personalized models for software defect prediction. *J. Syst. Softw.*, 181, 111038. doi:10.1016/j.jss.2021.111038
- Fan, Y., Xia, X., da Costa, D. A., Lo, D., Hassan, A. E., & Li, S. (2021). The impact of mislabeled changes by SZZ on just-in-time defect prediction. *IEEE Trans. Software Eng.*, 47(8), 1559–1586. doi:10.1109/TSE.2019.2929761
- Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K. E., Kabir, M. A., & Zhang, M. (2021). COSTE: complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction. *Inf. Softw. Technol.*, 129, 106432. doi:10.1016/j.infsof.2020.106432
- Feng, S., Keung, J., Yu, X., Xiao, Y., & Zhang, M. (2021). Investigation on the stability of smote-based oversampling techniques in software defect prediction. *Inf. Softw. Technol.*, 139, 106662. doi:10.1016/j.infsof.2021.106662
- Feng, S., Keung, J., Zhang, P., Xiao, Y., & Zhang, M. (2022). The impact of the distance metric and measure on smote-based techniques in software defect prediction. *Inf. Softw. Technol.*, 142, 106742. doi:10.1016/j.infsof.2021.106742
- Fenton, N. E., Neil, M., Marsh, W., Hearty, P. S., Radlinski, L., & Krause, P. (2008). On the effectiveness of early life cycle defect prediction with bayesian nets. *Empir. Softw. Eng.*, 13(5), 499–537. doi:10.1007/s10664-008-9072-x
- Ferenc, R., Gyimesi, P., Gyimesi, G., Tóth, Z., & Gyimóthy, T. (2020). An automatically created novel bug dataset and its validation in bug prediction. *J. Syst. Softw.*, 169, 110691. doi:10.1016/j.jss.2020.110691
- Gadelha, G., Ramalho, F., & Massoni, T. (2021). Traceability recovery between bug reports and test cases—a mozilla firefox case study. *Automated Software Engineering*, 28. doi:10.1007/s10515-021-00287-w
- Gomes, L. A. F., da Silva Torres, R., & Cortes, M. L. (2019). Bug report severity level prediction in open source software: A survey and research opportunities. *Inf. Softw. Technol.*, 115, 58–78. doi:10.1016/j.infsof.2019.07.009
- Goswami, D., Chakrabarti, S., & Bilgaiyan, S. (2020). Effort estimation of web based applications using erd, use case point method and machine learning. (pp. 19–37). doi:10.1007/978-3-030-38006-9_2
- Haider, S. W., Cangussu, J. W., Cooper, K. M. L., Dantu, R., & Haider, S. (2008). Estimation of defects based on defect decay model: $Ed^{\{3\}m}$. *IEEE Trans. Software Eng.*, 34(3), 336–356. doi:10.1109/TSE.2008.23
- Hassouna, A., & Tahvildari, L. (2010). An effort prediction framework for software defect correction. *Inf. Softw. Technol.*, 52(2), 197–209. doi:10.1016/j.infsof.2009.10.003

He, P., Li, B., Liu, X., Chen, J., & Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. *Inf. Softw. Technol.*, 59, 170–190. doi:10.1016/j.infsof.2014.11.006

He, Z., Shu, F., Yang, Y., Li, M., & Wang, Q. (2012). An investigation on the feasibility of cross-project defect prediction. *Autom. Softw. Eng.*, 19(2), 167–199. doi:10.1007/s10515-011-0090-3

Herbold, S. (2019). On the costs and profit of software defect prediction. *CoRR*, abs/1911.04309. arXiv: 1911.04309. Retrieved from <http://arxiv.org/abs/1911.04309>

Herbold, S., Trautsch, A., & Grabowski, J. (2017). Global vs. local models for cross-project defect prediction - A replication study. *Empir. Softw. Eng.*, 22(4), 1866–1902. doi:10.1007/s10664-016-9468-y

Herbold, S., Trautsch, A., & Grabowski, J. (2018). A comparative study to benchmark cross-project defect prediction approaches. *IEEE Trans. Software Eng.*, 44(9), 811–833. doi:10.1109/TSE.2017.2724538

Herbold, S., Trautsch, A., & Trautsch, F. (2020). On the feasibility of automated prediction of bug and non-bug issues. *Empir. Softw. Eng.*, 25(6), 5333–5369. doi:10.1007/s10664-020-09885-w

Herbold, S., Trautsch, A., & Trautsch, F. (2021). On the feasibility of automated prediction of bug and non-bug issues. In A. Koziol, I. Schaefer, & C. Seidl (Eds.), *Software engineering 2021, fachtagung des gifachbereichs softwaretechnik, 22.–26. februar 2021, braunschweig/virtuell* (Vol. P-310, pp. 55–56). doi:10.18420/SE2021_16

Honsel, D., Herbold, V., Waack, S., & Grabowski, J. (2021). Investigation and prediction of open source software evolution using automated parameter mining for agent-based simulation. *Autom. Softw. Eng.*, 28 (1), 3. doi:10.1007/s10515-021-00280-3

Hosseini, S., Turhan, B., & Mantyla, M. (2018). A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. *Inf. Softw. Technol.*, 95, 296–312. doi:10.1016/j.infsof.2017.06.004

Huang, L., Ng, V., Persing, I., Chen, M., Li, Z., Geng, R., & Tian, J. (2015). Autooodc: Automated generation of orthogonal defect classifications. *Autom. Softw. Eng.*, 22(1), 3–46. doi:10.1007/s10515-014-0155-1

Huang, Q., Xia, X., & Lo, D. (2019). Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empir. Softw. Eng.*, 24(5), 2823–2862. doi:10.1007/s10664-018-9661-2

Jiarpakdee, J., Tantithamthavorn, C. K., Dam, H. K., & Grundy, J. (2022). An empirical study of model-agnostic techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 48(1), 166–185. doi:10.1109/TSE.2020.2982385

Jing, X., Wu, F., Dong, X., & Xu, B. (2017). An improved SDA based defect prediction framework for both within-project and cross-project class-imbalance problems. *IEEE Trans. Software Eng.*, 43(4), 321–339. doi:10.1109/TSE.2016.2597849

Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., & Hassan, A. E. [Ahmed E.]. (2016). Studying just-in-time defect prediction using cross-project models. *Empir. Softw. Eng.*, 21(5), 2072–2106. doi:10.1007/s10664-015-9400-x

Keung, J., Kocaguneli, E., & Menzies, T. (2013). Finding conclusion stability for selecting the best effort predictor in software effort estimation. *Autom. Softw. Eng.*, 20(4), 543–567. doi:10.1007/s10515-012-0108-5

Kondo, M., Bezemer, C., Kamei, Y., Hassan, A. E., & Mizuno, O. (2019). The impact of feature reduction techniques on defect prediction models. *Empir. Softw. Eng.*, 24(4), 1925–1963. doi:10.1007/s10664-018-9679-5

- Kondo, M., Germán, D. M., Mizuno, O., & Choi, E. (2020). The impact of context metrics on just-in-time defect prediction. *Empir. Softw. Eng.*, 25(1), 890–939. doi:10.1007/s10664-019-09736-3
- Kpodjedjo, S., Ricca, F., Galinier, P., Gueheneuc, Y.-G., & Antoniol, G. (2011). Design evolution metrics for defect prediction in object oriented systems. *Empir. Softw. Eng.*, 16(1), 141–175. doi:10.1007/s10664-010-9151-7
- Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Inf. Softw. Technol.*, 58, 388–402. doi:10.1016/j.infsof.2014.07.005
- Larson, E. (2010). SUDS: an infrastructure for creating dynamic software defect detection tools. *Autom. Softw. Eng.*, 17(3), 301–346. doi:10.1007/s10515-010-0067-7
- Lee, T., Nam, J., Han, D., Kim, S., & In, H. P. (2016). Developer micro interaction metrics for software defect prediction. *IEEE Trans. Software Eng.*, 42(11), 1015–1035. doi:10.1109/TSE.2016.2550458
- Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Trans. Software Eng.*, 34(4), 485–496. doi:10.1109/TSE.2008.35
- Li, M., Zhang, H., Wu, R., & Zhou, Z. (2012). Sample-based software defect prediction with active and semi-supervised learning. *Autom. Softw. Eng.*, 19(2), 201–230. doi:10.1007/s10515-011-0092-1
- Li, N., Shepperd, M. J., & Guo, Y. (2020). A systematic review of unsupervised learning techniques for software defect prediction. *Inf. Softw. Technol.*, 122, 106287. doi:10.1016/j.infsof.2020.106287
- Li, W., Zhang, W., Jia, X., & Huang, Z. (2020). Effort-aware semi-supervised just-in-time defect prediction. *Inf. Softw. Technol.*, 126, 106364. doi:10.1016/j.infsof.2020.106364
- Li, Z., Jing, X., Wu, F., Zhu, X., Xu, B., & Ying, S. (2018). Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction. *Autom. Softw. Eng.*, 25(2), 201–245. doi:10.1007/s10515-017-0220-7
- Li, Z., Jing, X., Zhu, X., Zhang, H., Xu, B., & Ying, S. (2019). Heterogeneous defect prediction with two-stage ensemble learning. *Autom. Softw. Eng.*, 26(3), 599–651. doi:10.1007/s10515-019-00259-1
- Limsettho, N., Bennin, K. E., Keung, J. W., Hata, H., & Matsumoto, K. (2018). Cross project defect prediction using class distribution estimation and oversampling. *Inf. Softw. Technol.*, 100, 87–102. doi:10.1016/j.infsof.2018.04.001
- Liparas, D., Angelis, L., & Feldt, R. (2012). Applying the mahalanobis-taguchi strategy for software defect diagnosis. *Autom. Softw. Eng.*, 19(2), 141–165. doi:10.1007/s10515-011-0091-2
- Liu, C., Yang, D., Xia, X., Yan, M., & Zhang, X. (2019). A two-phase transfer learning model for cross-project defect prediction. *Inf. Softw. Technol.*, 107, 125–136. doi:10.1016/j.infsof.2018.11.005
- Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. *Inf. Softw. Technol.*, 54(3), 248–256. doi:10.1016/j.infsof.2011.09.007
- Mahmood, Z., Bowes, D., Hall, T., Lane, P. C. R., & Petric, J. (2018). Reproducibility and replicability of software defect prediction studies. *Inf. Softw. Technol.*, 99, 148–163. doi:10.1016/j.infsof.2018.02.003
- Malhotra, R., & Khanna, M. (2017). An exploratory study for software change prediction in object-oriented systems using hybridized techniques. *Autom. Softw. Eng.*, 24(3), 673–717. doi:10.1007/s10515-016-0203-0

Maruf, M., & Ozturk. (2017). Which type of metrics are useful to deal with class imbalance in software defect prediction? *Inf. Softw. Technol.*, 92, 17–29. doi:10.1016/j.infsof.2017.07.004

McIntosh, S., & Kamei, Y. (2018). Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. *IEEE Trans. Software Eng.*, 44(5), 412–428. doi:10.1109/TSE.2017.2693980

Menzies, T., Butcher, A., Cok, D. R., Marcus, A., Layman, L., Shull, F., ... Zimmermann, T. (2013). Local versus global lessons for defect prediction and effort estimation. *IEEE Trans. Software Eng.*, 39(6), 822–834. doi:10.1109/TSE.2012.83

Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., & Bener, A. B. (2010). Defect prediction from static code features: Current results, limitations, new approaches. *Autom. Softw. Eng.*, 17(4), 375–407. doi:10.1007/s10515-010-0069-5

Minku, L. L., & Yao, X. (2017). Which models of the past are relevant to the present? A software effort estimation approach to exploiting useful past models. *Autom. Softw. Eng.*, 24(3), 499–542. doi:10.1007/s10515-016-0209-7

Mo, R., Wei, S., Feng, Q., & Li, Z. (2022). An exploratory study of bug prediction at the method level. *Information and Software Technology*, 144, 106794. doi:https://doi.org/10.1016/j.infsof.2021.106794

Morasca, S., & Lavazza, L. (2020). On the assessment of software defect prediction models via ROC curves. *Empir. Softw. Eng.*, 25(5), 3977–4019. doi:10.1007/s10664-020-09861-4

Mori, T., & Uchihira, N. (2019). Balancing the trade-off between accuracy and interpretability in software defect prediction. *Empir. Softw. Eng.*, 24(2), 779–825. doi:10.1007/s10664-018-9638-1

Nair, T. R. G., & Selvarani, R. (2012). Defect proneness estimation and feedback approach for software design quality improvement. *Inf. Softw. Technol.*, 54(3), 274–285. doi:10.1016/j.infsof.2011.10.001

Nam, J., Fu, W., Kim, S., Menzies, T., & Tan, L. (2017). Heterogeneous defect prediction. *IEEE Trans. Software Eng.*, 44(9), 874–896. doi:10.1109/TSE.2017.2720603

Ni, C., Chen, X., Wu, F., Shen, Y., & Gu, Q. (2019). An empirical study on pareto based multi-objective feature selection for software defect prediction. *J. Syst. Softw.*, 152, 215–238. doi:10.1016/j.jss.2019.03.012

Ni, C., Xia, X., Lo, D., Chen, X., & Gu, Q. (2020). Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. *IEEE Transactions on Software Engineering*, 1–1. doi:10.1109/TSE.2020.3001739

Nizamani, Z. A., Liu, H., Chen, D. M., & Niu, Z. (2018). Automatic approval prediction for software enhancement requests. *Autom. Softw. Eng.*, 25(2), 347–381. doi:10.1007/s10515-017-0229-y

Nucci, D. D., Palomba, F., Rosa, G. D., Bavota, G., Oliveto, R., & Lucia, A. D. (2018). A developer centered bug prediction model. *IEEE Trans. Software Eng.*, 44(1), 5–24. doi:10.1109/TSE.2017.2659747

Okutan, A., & Yildiz, O. T. (2014). Software defect prediction using bayesian networks. *Empir. Softw. Eng.*, 19(1), 154–181. doi:10.1007/s10664-012-9218-8

Ozakinci, R., & Tarhan, A. (2018). Early software defect prediction: A systematic map and review. *J. Syst. Softw.*, 144, 216–239. doi:10.1016/j.jss.2018.06.025

Palomba, F., Zanoni, M., Fontana, F. A., Lucia, A. D., & Oliveto, R. (2019). Toward a smell-aware bug prediction model. *IEEE Trans. Software Eng.*, 45(2), 194–218. doi:10.1109/TSE.2017.2770122

Pascarella, L., Palomba, F., & Bacchelli, A. (2019). Fine-grained just-in-time defect prediction. *J. Syst. Softw.*, *150*, 22–36. doi:10.1016/j.jss.2018.12.001

Pascarella, L., Palomba, F., & Bacchelli, A. (2020). On the performance of method-level bug prediction: A negative result. *J. Syst. Softw.*, *161*. doi:10.1016/j.jss.2019.110493

Peters, F., Menzies, T., Gong, L., & Zhang, H. (2013). Balancing privacy and utility in cross-company defect prediction. *IEEE Trans. Software Eng.*, *39*(8), 1054–1068. doi:10.1109/TSE.2013.6

Peters, F., Tun, T. T., Yu, Y., & Nuseibeh, B. (2019). Text filtering and ranking for security bug report prediction. *IEEE Trans. Software Eng.*, *45*(6), 615–631. doi:10.1109/TSE.2017.2787653

Qu, Y., Chi, J. [Jianlei], & Yin, H. (2021). Leveraging developer information for efficient effort-aware bug prediction. *Inf. Softw. Technol.*, *137*, 106605. doi:10.1016/j.infsof.2021.106605

Qu, Y., Zheng, Q., Chi, J., Jin, Y., He, A., Cui, D., ... Liu, T. (2021). Using k-core decomposition on class dependency networks to improve bug prediction model's practical performance. *IEEE Trans. Software Eng.*, *47*(2), 348–366. doi:10.1109/TSE.2019.2892959

Rana, R., Staron, M., Berger, C., Hansson, J., Nilsson, M., & Meding, W. (2016). Analyzing defect inflow distribution and applying bayesian inference method for software defect prediction in large software projects. *J. Syst. Softw.*, *117*, 229–244. doi:10.1016/j.jss.2016.02.015

Rodríguez-Pérez, G., Nagappan, M., & Robles, G. (2021). Watch out for extrinsic bugs! A case study of their impact in just-in-time bug prediction models on the openstack project. *CoRR*, *abs/2103.15180*. arXiv: 2103.15180. Retrieved from <https://arxiv.org/abs/2103.15180>

Rodríguez, D., Ruiz, R., Riquelme, J. C., & Harrison, R. (2013). A study of subgroup discovery approaches for defect prediction. *Inf. Softw. Technol.*, *55*(10), 1810–1822. doi:10.1016/j.infsof.2013.05.002

Rodríguez, R. J. (2017). A petri net tool for software performance estimation based on upper throughput bounds. *Autom. Softw. Eng.*, *24*(1), 73–99. doi:10.1007/s10515-015-0186-2

Ryu, D., Choi, O., & Baik, J. (2016). Value-cognitive boosting with a support vector machine for cross-project defect prediction. *Empir. Softw. Eng.*, *21*(1), 43–71. doi:10.1007/s10664-014-9346-4

Shippey, T., Bowes, D., & Hall, T. (2019). Automatically identifying code features for software defect prediction: Using AST n-grams. *Inf. Softw. Technol.*, *106*, 142–160. doi:10.1016/j.infsof.2018.10.001

Song, Q., Guo, Y., & Shepperd, M. J. (2019). A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Trans. Software Eng.*, *45*(12), 1253–1269. doi:10.1109/TSE.2018.2836442

Song, Q., Jia, Z., Shepperd, M. J., Ying, S., & Liu, J. (2011). A general software defect-proneness prediction framework. *IEEE Trans. Software Eng.*, *37*(3), 356–370. doi:10.1109/TSE.2010.90

Tan, Y., Xu, S., Wang, Z., Zhang, T., Xu, Z., & Luo, X. (2020). Bug severity prediction using question-and-answer pairs from stack overflow. *J. Syst. Softw.*, *165*, 110567. doi:10.1016/j.jss.2020.110567

Tantithamthavorn, C., Hassan, A. E., & Matsumoto, K. (2020). The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Trans. Software Eng.*, *46*(11), 1200–1219. doi:10.1109/TSE.2018.2876537

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2017). An empirical comparison of model validation techniques for defect prediction models. *IEEE Trans. Software Eng.*, *43*(1), 1–18. doi:10.1109/TSE.2016.2584050

Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2019). The impact of automated parameter optimization on defect prediction models. *IEEE Trans. Software Eng.*, *45*(7), 683–711. doi:10.1109/TSE.2018.2794977

Tarvo, A., & Reiss, S. P. (2018). Automatic performance prediction of multithreaded programs: A simulation approach. *Autom. Softw. Eng.*, *25*(1), 101–155. doi:10.1007/s10515-017-0214-5

Thummalapenta, S., & Xie, T. (2011). Alattin: Mining alternative patterns for defect detection. *Autom. Softw. Eng.*, *18*(3–4), 293–323. doi:10.1007/s10515-011-0086-z

Thung, F., Lucia, Lo, D., Jiang, L., Rahman, F., & Devanbu, P. T. (2015). To what extent could we detect field defects? an extended empirical study of false negatives in static bug-finding tools. *Autom. Softw. Eng.*, *22*(4), 561–602. doi:10.1007/s10515-014-0169-8

Tian, Y., Lo, D., Xia, X., & Sun, C. (2015). Automated prediction of bug report priority using multi-factor analysis. *Empir. Softw. Eng.*, *20*(5), 1354–1383. doi:10.1007/s10664-014-9331-y

Tong, H., Liu, B., & Wang, S. (2018). Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. *Inf. Softw. Technol.*, *96*, 94–111. doi:10.1016/j.infsof.2017.11.008

Tong, H., Liu, B., & Wang, S. (2021). Kernel spectral embedding transfer ensemble for heterogeneous defect prediction. *IEEE Trans. Software Eng.*, *47*(9), 1886–1906. doi:10.1109/TSE.2019.2939303

Tong, H., Lu, W., Xing, W., Liu, B., & Wang, S. (2022). SHSE: A subspace hybrid sampling ensemble method for software defect number prediction. *Inf. Softw. Technol.*, *142*, 106747. doi:10.1016/j.infsof.2021.106747

Tosun, A., Bener, A. B., Turhan, B., & Menzies, T. (2010). Practical considerations in deploying statistical methods for defect prediction: A case study within the turkish telecommunications industry. *Inf. Softw. Technol.*, *52*(11), 1242–1257. doi:10.1016/j.infsof.2010.06.006

Tu, H., Yu, Z., & Menzies, T. (2022). Better data labelling with emblem (and how that impacts defect prediction). *IEEE Transactions on Software Engineering*, *48*(1), 278–294. doi:10.1109/TSE.2020.2986415

Turhan, B., Menzies, T., Bener, A. B., & Stefano, J. S. D. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empir. Softw. Eng.*, *14*(5), 540–578. doi:10.1007/s10664-008-9103-7

Wang, S., Liu, T., Nam, J., & Tan, L. (2020). Deep semantic feature learning for software defect prediction. *IEEE Trans. Software Eng.*, *46*(12), 1267–1293. doi:10.1109/TSE.2018.2877612

Wang, T., Zhang, Z., Jing, X., & Zhang, L. (2016). Multiple kernel ensemble learning for software defect prediction. *Autom. Softw. Eng.*, *23*(4), 569–590. doi:10.1007/s10515-015-0179-1

Weyuker, E. J., Ostrand, T. J., & Bell, R. M. (2008). Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models. *Empir. Softw. Eng.*, *13*(5), 539–559. doi:10.1007/s10664-008-9082-8

Wu, X., Zheng, W., Chen, X., Zhao, Y., Yu, T., & Mu, D. (2021). Improving high-impact bug report prediction with combination of interactive machine learning and active learning. *Inf. Softw. Technol.*, *133*, 106530. doi:10.1016/j.infsof.2021.106530

- Xia, X., & Lo, D. (2017). An effective change recommendation approach for supplementary bug fixes. *Autom. Softw. Eng.*, 24(2), 455–498. doi:10.1007/s10515-016-0204-z
- Xia, X., Lo, D., Pan, S. J., Nagappan, N., & Wang, X. (2016). HYDRA: massively compositional model for cross-project defect prediction. *IEEE Trans. Software Eng.*, 42(10), 977–998. doi:10.1109/TSE.2016.2543218
- Xia, X., Lo, D., Shihab, E., Wang, X., & Zhou, B. (2015). Automatic, high accuracy prediction of reopened bugs. *Autom. Softw. Eng.*, 22(1), 75–109. doi:10.1007/s10515-014-0162-2
- Xiao, P., Liu, B., & Wang, S. (2018). Feedback-based integrated prediction: Defect prediction based on feedback from software testing process. *J. Syst. Softw.*, 143, 159–171. doi:10.1016/j.jss.2018.05.029
- Xu, Z., Li, L., Yan, M., Liu, J., Luo, X., Grundy, J., ... Zhang, X. (2021). A comprehensive comparative study of clustering-based unsupervised defect prediction models. *J. Syst. Softw.*, 172, 110862. doi:10.1016/j.jss.2020.110862
- Xu, Z., Li, S., Luo, X., Liu, J., Zhang, T., Tang, Y., ... Keung, J. (2019). TSTSS: A two-stage training subset selection framework for cross version defect prediction. *J. Syst. Softw.*, 154, 59–78. doi:10.1016/j.jss.2019.03.027
- Xu, Z., Li, S., Xu, J., Liu, J., Luo, X., Zhang, Y., ... Tang, Y. (2019). LDFR: learning deep feature representation for software defect prediction. *J. Syst. Softw.*, 158. doi:10.1016/j.jss.2019.110402
- Xu, Z., Liu, J. [Jin], Luo, X., Yang, Z., Zhang, Y., Yuan, P., ... Zhang, T. (2019). Software defect prediction based on kernel PCA and weighted extreme learning machine. *Inf. Softw. Technol.*, 106, 182–200. doi:10.1016/j.infsof.2018.10.004
- Yang, X., Lo, D., Xia, X., & Sun, J. (2017). TLEL: A two-layer ensemble learning approach for just-in-time defect prediction. *Inf. Softw. Technol.*, 87, 206–220. doi:10.1016/j.infsof.2017.03.007
- Yao, J., & Shepperd, M. J. (2021). The impact of using biased performance metrics on software defect prediction research. *Inf. Softw. Technol.*, 139, 106664. doi:10.1016/j.infsof.2021.106664
- Yu, Q., Jiang, S., & Zhang, Y. (2017). A feature matching and transfer approach for cross-company defect prediction. *J. Syst. Softw.*, 132, 366–378. doi:10.1016/j.jss.2017.06.070
- Yu, T., Wen, W., Han, X., & Hayes, J. H. (2019). Conpredictor: Concurrency defect prediction in real-world applications. *IEEE Trans. Software Eng.*, 45(6), 558–575. doi:10.1109/TSE.2018.2791521
- Zhang, F., Hassan, A. E., McIntosh, S., & Zou, Y. (2017). The use of summation to aggregate software metrics hinders the performance of defect prediction models. *IEEE Trans. Software Eng.*, 43(5), 476–491. doi:10.1109/TSE.2016.2599161
- Zhang, F., Mockus, A., Keivanloo, I., & Zou, Y. (2016). Towards building a universal defect prediction model with rank learned predictors. *Empir. Softw. Eng.*, 21(5), 2107–2145. doi:10.1007/s10664-015-9396-2
- Zhang, Z., Jing, X., & Wang, T. (2017). Label propagation based semi-supervised learning for software defect prediction. *Autom. Softw. Eng.*, 24(1), 47–69. doi:10.1007/s10515-016-0194-x
- Zhao, Y., Yang, Y., Lu, H., Liu, J., Leung, H., Wu, Y., ... Xu, B. (2017). Understanding the value of considering client usage context in package cohesion for fault-proneness prediction. *Autom. Softw. Eng.*, 24(2), 393–453. doi:10.1007/s10515-016-0198-6
- Zhou, T., Sun, X., Xia, X., Li, B., & Chen, X. (2019). Improving defect prediction with deep forest. *Inf. Softw. Technol.*, 114, 204–216. doi:10.1016/j.infsof.2019.07.003

Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., ... Xu, B. (2018). How far we have progressed in the journey? an examination of cross-project defect prediction. *ACM Trans. Softw. Eng. Methodol.*, 27(1), 1:1–1:51. doi:10.1145/3183339

Zhu, K., Ying, S., Zhang, N., & Zhu, D. (2021). Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. *J. Syst. Softw.*, 180, 111026. doi:10.1016/j.jss.2021.111026

Zou, Q., Lu, L., Yang, Z., Gu, X., & Qiu, S. (2021). Joint feature representation learning and progressive distribution matching for cross-project defect prediction. *Inf. Softw. Technol.*, 137, 106588. doi:10.1016/j.infsof.2021.106588

Acknowledgements We thank Bailey Renee Vandehei for proofreading this article.

Funding Open access funding provided by Università degli Studi di Roma Tor Vergata within the CRUI-CARE Agreement.

Declarations

Conflict of Interest The authors have no relevant financial or non-financial interests to disclose. The authors have no conflicts of interest to declare that are relevant to the content of this article.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Agrawal A, Menzies T (2018) Is “better data” better than “better data miners”? On the benefits of tuning SMOTE for defect prediction. In: Proceedings of the 40th international conference on software engineering, ICSE 2018, Gothenburg, Sweden, May 27–June 03, 2018, pp 1050–1061
- Aha D, Kibler D (1991) Instance-based learning algorithms. *Mach Learn* 6:37–66
- Ahluwalia A, Falessi D, Penta MD (2019) Snoring: a noise in defect prediction datasets. In: Storey MD, Adams B, Haiduc S (eds) Proceedings of the 16th international conference on mining software repositories, MSR 2019, 26–27 May 2019, Montreal, Canada, pp 63–67. <https://doi.org/10.1109/MSR.2019.00019>
- Akoglu H (2018) User's guide to correlation coefficients. *Turk J Emerg Med* 18(3):91–93. <https://doi.org/10.1016/j.tjem.2018.08.001>
- Altman NS (1992) An introduction to kernel and nearest-neighbor nonparametric regression. *Am Stat* 46(3):175–185. Retrieved from <http://www.jstor.org/stable/2685209>
- Amasaki S (2020) Cross-version defect prediction: use historical data, crossproject data, or both? *Empir Softw Eng* 25(2):1573–1595
- Arisholm E, Briand LC, Fuglerud M (2007) Data mining techniques for building fault-proneness models in telecom java software. In: ISSRE 2007, the 18th IEEE international symposium on software reliability, Trollhättan, Sweden, 5–9 November 2007, pp 215–224. <https://doi.org/10.1109/ISSRE.2007>
- Bangash AA, Sahar H, Hindle A, Ali K (2020) On the time-based conclusion stability of cross-project defect prediction models. *Empir Softw Eng* 25(6):5047–5083
- Basili VR, Briand LC, Melo WL (1996) A validation of objectoriented design metrics as quality indicators. *IEEE Trans Softw Eng* 22(10):751–761

- Ben-Gal I (2008) Bayesian networks. <https://doi.org/10.1002/9780470061572.eqr089>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470061572.eqr089>
- Bennin KE, Keung J, Phannachitta P, Monden A, Mensah S (2018) MAHAKIL: diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. *IEEE Trans Softw Eng* 44(6):534–550. <https://doi.org/10.1109/TSE.2017.2731766>
- Bennin KE, Keung JW, Monden A (2019) On the relative value of data resampling approaches for software defect prediction. *Empir Softw Eng* 24(2):602–636
- Bird C, Bachmann A, Aune E, Duffy J, Bernstein A, Filkov V, Devanbu PT (2009) Fair and balanced?: Bias in bug-fix datasets. In: van Vlie H, Issarny V (eds) Proceedings of the 7th joint meeting of the european software engineering conference and the ACM SIGSOFT international symposium on foundations of software engineering, 2009, Amsterdam, The Netherlands, August 24–28, 2009, pp 121–130. <https://doi.org/10.1145/1595696.1595716>
- Breiman L (2001) Random forests. *Mach Learn* 45(1):5–32
- Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP (2002) Smote: synthetic minority over-sampling technique. *J Artif Intell Res* 16:321–357
- Chen T-H, Nagappan M, Shihab E, Hassan AE (2014) An empirical study of dormant bugs. In: Proceedings of the 11th working conference on mining software repositories - MSR, p 2014. <https://doi.org/10.1145/2597073.2597108>
- Chen H, Liu W, Gao D, Peng X, Zhao W (2017) Personalized defect prediction for individual source files. *Comput Sci* 44(4):90–95. <https://doi.org/10.11896/j.issn.1002-137X.2017.04.020>
- Chen H, Jing X, Li Z, Wu D, Peng Y, Huang Z (2020) An empirical study on heterogeneous defect prediction approaches. *IEEE Trans Softw Eng* (01):1–1. <https://doi.org/10.1109/TSE.2020.2968520>
- Chen X, Mu Y, Liu K, Cui Z, Ni C (2021) Revisiting heterogeneous defect prediction methods: how far are we? *Inf Softw Technol* 130:106441. <https://doi.org/10.1016/j.infsof.2020.106441>
- Chi J, Honda K, Washizaki H, Fukazawa Y, Munakata K, Morita S, Yamamoto R (2017) Defect analysis and prediction by applying the multistage software reliability growth model. In: IWSESE. IEEE Computer Society, pp 7–11
- Cleary JG, Trigg LE (1995) K*: an instance-based learner using an entropic distance measure. In: 12th International conference on machine learning, pp 108–114
- Dalla Palma S, Di Nucci D, Palomba F, Tamburri DA (2021) Withinproject defect prediction of infrastructure-as-code using product and process metrics. *IEEE Trans Softw Eng* 1–1. <https://doi.org/10.1109/TSE.2021.3051492>
- D’Ambros M, Lanza M, Robbes R (2012) Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empir Softw Eng* 17(4–5):531–577. <https://doi.org/10.1007/s10664-011-9173-9>
- Falessi D, Huang J, Narayana L, Thai JF, Turhan B (2020) On the need of preserving order of data when validating within-project defect classifiers. *Empir Softw Eng* 25(6):4805–4830. <https://doi.org/10.1007/s10664-020-09868-x>
- Falessi D, Ahluwalia A, Penta MD (2022) The impact of dormant defects on defect prediction: a study of 19 apache projects. *ACM Trans. Softw Eng Methodol* 31(1):4:1–4:26. <https://doi.org/10.1145/3467895>
- Fan Y, Xia X, da Costa DA, Lo D, Hassan AE, Li S (2021) The impact of mislab eled changes by SZZ on just-in-time defect prediction. *IEEE Trans Software Eng* 47(8):1559–1586. <https://doi.org/10.1109/TSE.2019.2929761>
- Feng S, Keung J, Yu X, Xiao Y, Bennin KE, Kabir MA, Zhang M (2021) COSTE: complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction. *Inf Softw Technol* 129:106432. <https://doi.org/10.1016/j.infsof.2020.106432>
- Flint SW, Chauhan J, Dyer R (2021) Escaping the time pit: Pitfalls and guidelines for using time-based git data. In: 18th IEEE/ACM international conference on mining software repositories, MSR 2021, Madrid, Spain, May 17–19, 2021, pp 85–96. <https://doi.org/10.1109/MSR52588.2021.00022>
- Fu W, Menzies T, Shen X (2016) Tuning for software analytics: is it really necessary? *Softw Technol* 76:135–146. <https://doi.org/10.1016/j.infsof.2016.04.017>
- Fukushima T, Kamei Y, McIntosh S, Yamashita K, Ubayashi N (2014) An empirical study of just-in-time defect prediction using crossproject models. In: Proceedings of the 11th working conference on mining software repositories, pp 172–181
- Ghotra B, McIntosh S, Hassan AE (2017) A large-scale study of the impact of feature selection techniques on defect classification models. In: 2017 IEEE/ACM 14th international conference on mining software repositories (msr). IEEE, pp 146–157
- Giger E, D’Ambros M, Pinzger M, Gall H (2012) Method-level bug prediction, pp 171–180. <https://doi.org/10.1145/2372251.2372285>
- Grissom RJ, Kim JJ (2005) Effect sizes for research: a broad practical approach, 2nd edn. Lawrence Earlbaum Associates

- Gyimóthy T, Ferenc R, Siket I (2005) Empirical validation of objectoriented metrics on open source software for fault prediction. *IEEE Trans Softw Eng* 31(10):897–910
- Hall MA (1998) Correlation-based feature subset selection for machine learning (Doctoral dissertation University of Waikato, Hamilton, New Zealand)
- Hassan AE (2009) Predicting faults using the complexity of code changes. In: 31st International conference on software engineering, ICSE 2009, May 16–24, 2009, Vancouver, Canada, proceedings, pp 78–88. <https://doi.org/10.1109/ICSE.2009.5070510>
- Herbold S (2017) Comments on scottknotted in response to “an empirical comparison of model validation techniques for defect prediction models”. *IEEE Trans Softw Eng* 43(11):1091–1094. <https://doi.org/10.1109/TSE.2017.2748129>
- Herbold S (2019) On the costs and profit of software defect prediction. *CoRR*. arXiv:1911.04309
- Herbold S, Trautsch A, Grabowski J (2017) Global vs. local models for cross-project defect prediction—a replication study. *Empir Softw Eng* 22(4):1866–1902
- Herbold S, Trautsch A, Grabowski J (2018) A comparative study to benchmark cross-project defect prediction approaches. *IEEE Trans Softw Eng* 44(9):811–833. <https://doi.org/10.1109/TSE.2017.2724538>
- Herbold S, Trautsch A, Grabowski J (2019) Correction of “a comparative study to benchmark cross-project defect prediction approaches”. *IEEE Trans Softw Eng* 45(6):632–636
- Herbold S, Trautsch A, Trautsch F (2020) On the feasibility of automated prediction of bug and non-bug issues. *Empir Softw Eng* 25(6):5333–5369
- Herzig K, Just S, Zeller A (2013) It’s not a bug, it’s a feature: how misclassification impacts bug prediction. In: Notkin D, Cheng BHC, Pohl K (eds) 35th International conference on software engineering, ICSE ’13, San Francisco, CA, USA, May 18–26, 2013, pp 392–401. <https://doi.org/10.1109/ICSE.2013.6606585>
- Hosseini S, Turhan B, Gunarathna D (2019) A systematic literature review and meta-analysis on cross project defect prediction. *IEEE Trans Softw Eng* 45(2):111–147
- Huang Q, Xia X, Lo D (2019) Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. *Empir Softw Eng* 24(5):2823–2862
- Jiang T, Tan L, Kim S (2013) Personalized defect prediction. <https://doi.org/10.1109/ASE.2013.6693087>
- Jiang Y, Cukic B, Menzies T (2008) Can data transformation help in the detection of fault-prone modules? In: Devanbu PT, Murphy B, Nagappan N, Zimmermann T (eds) Proceedings of the 2008 workshop on defects in large software systems, held in conjunction with the ACM SIGSOFT international symposium on software testing and analysis (ISSTA 2008), DEFECTS 2008, Seattle, Washington, USA, July 20, 2008, pp 16–20. <https://doi.org/10.1145/1390817.1390822>
- Jiarpakdee J, Tantithamthavorn C, Dam HK, Grundy J (2020) An empirical study of model-agnostic techniques for defect prediction models. *IEEE Trans Softw Eng* 1–1. <https://doi.org/10.1109/TSE.2020.2982385>
- Jing X, Wu F, Dong X, Xu B (2017) An improved SDA based defect prediction framework for both within-project and cross-project classimbalance problems. *IEEE Trans Softw Eng* 43(4):321–339
- John GH, Langley P (1995) Estimating continuous distributions in bayesian classifiers. In: Eleventh conference on uncertainty in artificial intelligence. Morgan Kaufmann, San Mateo, pp 338–345
- Kamei Y, Shihab E, Adams B, Hassan AE, Mockus A, Sinha A, Ubayashi N (2012) A large-scale empirical study of just-in-time quality assurance. *IEEE Trans Softw Eng* 39(6):757–773
- Kamei Y, Fukushima T, McIntosh S, Yamashita K, Ubayashi N, Hassan AE (2016) Studying just-in-time defect prediction using cross-project models. *Empir Softw Eng* 21(5):2072–2106
- Khoshgoftaar TM, Allen EB, Goel N, Nandi A, McMullan J (1996) Detection of software modules with high debug code churn in a very large legacy system. In: Seventh international symposium on software reliability engineering, ISSRE 1996, white plains, NY, USA, October 30, 1996–Nov. 2, 1996, pp 364–371. <https://doi.org/10.1109/ISSRE.1996.558896>
- Kim S, Zimmermann T Jr, Whitehead EJ, Zeller A (2007) Predicting faults from cached history. In: 29th International conference on software engineering (ICSE 2007), Minneapolis, MN, USA, May 20–26, 2007, pp 489–498. <https://doi.org/10.1109/ICSE.2007.66>
- Kim S Jr, Whitehead EJ, Zhang Y (2008) Classifying software changes: clean or buggy. *IEEE Trans Softw Eng* 34(2):181–196
- Kim S, Zhang H, Wu R, Gong L (2011) Dealing with noise in defect prediction. In: Taylor RN, Gall HC, Medvidovic N (eds) Proceedings of the 33rd international conference on software engineering, ICSE 2011, Waikiki, Honolulu, HI, USA, May 21–28, 2011, pp 481–490. <https://doi.org/10.1145/1985793.1985859>
- Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering, EBSE 2007-001. Keele University and Durham University Joint Report, (Jul 9 2007)
- Kochhar PS, Xia X, Lo D, Li S (2016) Practitioners’ expectations on automated fault localization. In: Proceedings of the 25th international symposium on software testing and analysis, pp 165–176. <https://doi.org/10.1145/2931037.2931051>

- Kohavi R (1995) The power of decision tables. In: 8th European conference on machine learning. Springer, pp 174–189
- Kondo M, Bezemer C-P, Kamei Y, Hassan AE, Mizuno O (2019) The impact of feature reduction techniques on defect prediction models. *Empir Softw Eng* 24(4):1925–1963
- Kondo M, German DM, Mizuno O, Choi E (2020) The impact of context metrics on just-in-time defect prediction. *Empir Softw Eng* 25(1):890–939
- Kotsiantis S, Tsekouras G, Pintelas P (2005) Bagging model trees for classification problems
- Le Cessie JC, Van Houwelingen S (1992) Ridge estimators in logistic regression. *Applied statistics*
- Lee T, Nam J, Han D, Kim S, In HP (2016) Developer micro interaction metrics for software defect prediction. *IEEE Trans Softw Eng* 42(11):1015–1035
- Liu J, Zhou Y, Yang Y, Lu H, Xu B (2017) Code churn: a neglected metric in effort-aware just-in-time defect prediction. In: ESEM. IEEE Computer Society, pp 11–19
- Matthews BW (1975) Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochim Biophys Acta (BBA)* 2(405):442–451
- McCallum A, Nigam K (1998) A comparison of event models for naive Bayes text classification. In: Learning for text categorization: papers from the 1998 AAAI workshop, pp 41–48. Retrieved from <http://www.kamalnigam.com/papers/multinomial-aaaiws98.pdf>
- McIntosh S, Kamei Y (2018) Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. *IEEE Trans Softw Eng* 44(5):412–428
- Mende T, Koschke R (2009) Revisiting the evaluation of defect prediction models. In: Ostrand TJ (ed) Proceedings of the 5th international workshop on predictive models in software engineering, PROMISE 2009, Vancouver, BC, Canada, May 18–19, 2009, p 7. <https://doi.org/10.1145/1540438.1540448>
- Menzies T, Dekhtyar A, Stefano JSD, Greenwald J (2007a) Problems with precision: a response to “comments on ‘data mining static code attributes to learn defect predictors’”. *IEEE Trans Softw Eng* 33(9):637–640. <https://doi.org/10.1109/TSE.2007.70721>
- Menzies T, Greenwald J, Frank A (2007b) Data mining static code attributes to learn defect predictors. *IEEE Trans Softw Eng* 33(1):2–13. <https://doi.org/10.1109/TSE.2007.256941>
- Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Basar Bener A (2010) Defect prediction from static code features: current results, limitations, new approaches. *Autom Softw Eng* 17(4):375–407
- Morasca S, Lavazza L (2020) On the assessment of software defect prediction models via ROC curves. *Empir Softw Eng* 25(5):3977–4019
- Mori T, Uchihira N (2019) Balancing the trade-off between accuracy and interpretability in software defect prediction. *Empir Softw Eng* 24(2):779–825
- Moser R, Pedrycz W, Succì G (2008) A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: 30th International conference on software engineering (ICSE 2008), Leipzig, Germany, May 10–18, 2008, pp 181–190
- Nagappan N, Ball T (2005) Use of relative code churn measures to predict system defect density. In: Roman G, Griswold WG, Nuseibeh B (eds) 27th International conference on software engineering (ICSE 2005), 15–21 May 2005, St. Louis, Missouri, pp 284–292. <https://doi.org/10.1145/1062455.1062514>
- Nucci DD, Palomba F, Rosa GD, Bavota G, Oliveto R, Lucia AD (2018) A developer centered bug prediction model. *IEEE Trans Softw Eng* 44(1):5–24
- Ohlsson N, Alberg H (1996) Predicting fault-prone software modules in telephone switches. *IEEE Trans Softw Eng* 22(12):886–894. <https://doi.org/10.1109/32.553637>
- Ostrand TJ, Weyuker EJ (2004) A tool for mining defect-tracking systems to predict fault-prone files. In: Hassan AE, Holt RC, Mockus A (eds) Proceedings of the 1st international workshop on mining software repositories, msr@icse 2004, Edinburgh, Scotland, UK, 25th May 2004, pp 85–89
- Ostrand TJ, Weyuker EJ, Bell RM (2005) Predicting the location and number of faults in large software systems. *IEEE Trans Softw Eng* 31(4):340–355
- Palomba F, Zanzi M, Fontana FA, Lucia AD, Oliveto R (2019) Toward a smell-aware bug prediction model. *IEEE Trans Softw Eng* 45(2):194–218
- Pascarella L, Palomba F, Bacchelli A (2019) Fine-grained just-in-time defect prediction. *J Syst Softw* 150:22–36
- Pascarella L, Palomba F, Bacchelli A (2020) On the performance of method-level bug prediction: a negative result. *J Syst Softw* 161
- Peters F, Tun TT, Yu Y, Nuseibeh B (2019) Text filtering and ranking for security bug report prediction. *IEEE Trans Softw Eng* 45(6):615–631
- Platt J (1998) Fast training of support vector machines using sequential minimal optimization. In: Schoelkopf B, Burges C, Smola A (eds) Advances in kernel methods—support vector learning. MIT Press. Retrieved from <http://research.microsoft.com/%5C~jplatt/smo.html>
- Powers DMW (2007) Evaluation: from precision, recall and F-measure to ROC, informedness, markedness & correlation. *J Mach Learn Technol* 2(1):37–63

- Qu Y, Zheng Q, Chi J, Jin Y, He A, Cui D (2021a) Using k-core decomposition on class dependency networks to improve bug prediction model's practical performance. *IEEE Trans Softw Eng* 47(2):348–366
- Qu Y, Chi J, Yin H (2021b) Leveraging developer information for efficient effort-aware bug prediction. *Inf Softw Technol* 137:106605. <https://doi.org/10.1016/j.infsof.2021.106605>
- Quinlan R (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers, San Mateo
- Rahman F, Posnett D, Devanbu PT (2012) Recalling the “imprecision” of cross-project defect prediction. In: Tracz W, Robillard MP, Bultan T (eds) 20th ACM SIGSOFT symposium on the foundations of software engineering (fse-20), sigsoft/fse'12, Cary, NC, USA—November 11–16, 2012, p 61. <https://doi.org/10.1145/2393596.2393669>
- Rahman F, Posnett D, Herraiz I, Devanbu PT (2013) Sample size vs. bias in defect prediction. In: Meyer B, Baresi L, Mezini M (eds) Joint meeting of the european software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, esec/fse'13, Saint Petersburg, Russian Federation, August 18–26, 2013, pp 147–157. <https://doi.org/10.1145/2491411.2491418>
- Rodríguez-Pérez G, Zaidman A, Serebrenik A, Robles G, González-Barahona JM (2018b) What if a bug has a different origin?: Making sense of bugs without an explicit bug introducing change. In: Oivo M, Fernández DM, Mockus A (eds) Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement, ESEM 2018, Oulu, Finland, October 11–12, 2018, pp 52:1–52:4. <https://doi.org/10.1145/3239235.3267436>
- Rodríguez-Pérez G, Nagappan M, Robles G (2020) Watch out for extrinsic bugs! A case study of their impact in just-in-time bug prediction models on the openstack project. *IEEE Trans Softw Eng* 1–1. <https://doi.org/10.1109/TSE.2020.3021380>
- Shepperd M, Song Q, Sun Z, Mair C (2013) Data quality: some comments on the nasa software defect datasets. *IEEE Trans Softw Eng* 39(9):1208–1215
- Shepperd M, Bowes D, Hall T (2014) Researcher bias: the use of machine learning in software defect prediction. *IEEE Trans Softw Eng* 40(6):603–616. <https://doi.org/10.1109/TSE.2014.2322358>
- Shepperd MJ, Hall T, Bowes D (2018) Authors' reply to “comments on ‘researcher bias: the use of machine learning in software defect prediction’”. *IEEE Trans Softw Eng* 44(11):1129–1131
- Song Q, Guo Y, Shepperd MJ (2019) A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Trans Softw Eng* 45(12):1253–1269
- Spearman C (1904) The proof and measurement of association between two things. *Am J Psychol* 15(1):72–101
- Tantithamthavorn C, McIntosh S, Hassan AE, Ihara A, Matsumoto K (2015) The impact of mislabelling on the performance and interpretation of defect prediction models. In: Bertolino A, Canfora G, Elbaum SG (eds) 37th IEEE/ACM international conference on software engineering, ICSE 2015, Florence, Italy, May 16–24, 2015, vol 1, pp 812–823. <https://doi.org/10.1109/ICSE.2015.93>
- Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016b) Comments on “researcher bias: the use of machine learning in software defect prediction”. *IEEE Trans Softw Eng* 42(11):1092–1094. <https://doi.org/10.1109/TSE.2016.2553030>
- Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2016c) An empirical comparison of model validation techniques for defect prediction models. *IEEE Trans Softw Eng* 43(1):1–18
- Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2019) The impact of automated parameter optimization on defect prediction models. *IEEE Trans Softw Eng* 45(7):683–711. <https://doi.org/10.1109/TSE.2018.2794977>
- Tantithamthavorn C, Hassan AE, Matsumoto K (2020) The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Trans Softw Eng* 46(11):1200–1219. <https://doi.org/10.1109/TSE.2018.2876537>
- Tian Y, Lo D, Xia X, Sun C (2015) Automated prediction of bug report priority using multi-factor analysis. *Empir Softw Eng* 20(5):1354–1383
- Tu H, Yu Z, Menzies T (2020) Better data labelling with emblem (and how that impacts defect prediction). *IEEE Trans Softw Eng* 1–1. <https://doi.org/10.1109/TSE.2020.2986415>
- Turhan B, Menzies T, Bener AB, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. *Empir Softw Eng* 14(5):540–578. <https://doi.org/10.1007/s10664-008-9103-7>
- Vandehi B, da Costa DA, Falessi D (2021) Leveraging the defects life cycle to label affected versions and defective classes. *ACM Trans Softw Eng Methodol* 30(2):24:1–24:35
- Vargha A, Delaney HD (2000) A critique and improvement of the cl common language effect size statistics of McGraw and Wong. *J Educ Behav Stat* 25(2):101–132
- Wang S, Liu T, Tan L (2016) Automatically learning semantic features for defect prediction. In: Proceedings of the 38th international conference on software engineering, ICSE 2016, Austin, TX, USA, May 14–22, 2016, pp 297–308

- Wang S, Liu T, Nam J, Tan L (2020) Deep semantic feature learning for software defect prediction. *IEEE Trans Software Eng* 46(12):1267–1293. <https://doi.org/10.1109/TSE.2018.2877612>
- Weyuker EJ, Ostrand TJ, Bell RM (2010) Comparing the effectiveness of several modeling methods for fault prediction. *Empir Softw Eng* 15(3):277–295. <https://doi.org/10.1007/s10664-009-9111-2>
- Wilcoxon F (1945) Individual comparisons by ranking methods. *Biometrics* 1(6):80. <https://doi.org/10.2307/3001968>
- Wohlin C, Runeson P, Hst M, Ohlsson MC, Regnell B, Wessln A (2012) Experimentation in software engineering. Springer Publishing Company Incorporated
- Xia X, Lo D, Pan SJ, Nagappan N, Wang X (2016) HYDRA: massively compositional model for cross-project defect prediction. *IEEE Trans Softw Eng* 42(10):977–998. <https://doi.org/10.1109/TSE.2016.2543218>
- Yan M, Fang Y, Lo D, Xia X, Zhang X (2017) File-level defect prediction: unsupervised vs. supervised models. In: ESEM. IEEE Computer Society, pp 344–353
- Yu T, Wen W, Han X, Hayes JH (2019) Conpredictor: concurrency defect prediction in real-world applications. *IEEE Trans Softw Eng* 45(6):558–575
- Zhang H, Zhang X (2007) Comments on “data mining static code attributes to learn defect predictors”. *IEEE Trans Softw Eng* 33(9):635–637. <https://doi.org/10.1109/TSE.2007.70706>
- Zhang F, Mockus A, Keivanloo I, Zou Y (2016) Towards building a universal defect prediction model with rank transformed predictors. *Empir Softw Eng* 21(5):2107–2145
- Zhang F, Hassan AE, McIntosh S, Zou Y (2017) The use of summation to aggregate software metrics hinders the performance of defect prediction models. *IEEE Trans Softw Eng* 43(5):476–491

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Jonida Çarka is a Ph.D. student at the University of Rome Tor Vergata, Italy. She is also an academic staff member of the University of Tirana, Faculty of Economics, Department of Statistics and Applied Informatics. Her main research interest is in measuring the impact of defect classifiers on software development. She received her MSc in Information Systems in Economy and BSc degrees in Economic Informatics.



Matteo Esposito is a Ph.D. student at the University of Rome Tor Vergata, Italy. He is also a Secure Software Engineer for Multitel SRL. He is the Chairman of the ACM Rome Tor Vergata Student Chapter. His main research interest is in using machine learning to support software security. He received his MSc and BSc degrees in Computer Engineering from the University of Rome Tor Vergata, Italy.



Davide Falessi is an Associate Professor of Software Engineering at the University of Rome Tor Vergata, Italy. He is the Associate Editor in Software Economics of IEEE Software and a senior member of IEEE. He is a reviewer board member of the IEEE Transactions on Software Engineering. He has been the Guest Editor of special issues in several journals, including the Empirical Software Engineering Journal, the Journal of Systems and Software and IEEE Software. His main research interest is in devising and empirically assessing scalable solutions for the development of software-intensive systems. He received his Ph.D., MSc, and BSc degrees in Computer Engineering from the University of Rome Tor Vergata, Italy.