An Empirical Comparison of Model Validation Techniques for Defect Prediction Models

Chakkrit Tantithamthavorn, *Student Member, IEEE,* Shane McIntosh, *Member, IEEE,* Ahmed E. Hassan, *Member, IEEE,* and Kenichi Matsumoto, *Senior Member, IEEE*

Abstract—Defect prediction models help software quality assurance teams to allocate their limited resources to the most defect-prone modules. Model validation techniques, such as k-fold cross-validation, use historical data to estimate how well a model will perform in the future. However, little is known about how accurate the estimates of model validation techniques tend to be. In this paper, we investigate the bias and variance of model validation techniques in the domain of defect prediction. Analysis of 101 public defect datasets suggests that 77% of them are highly susceptible to producing unstable results—selecting an appropriate model validation technique is a critical experimental design choice. Based on an analysis of 256 studies in the defect prediction literature, we select the 12 most commonly adopted model validation techniques for evaluation. Through a case study of 18 systems, we find that single-repetition holdout validation tends to produce estimates with 46%-229% more bias and 53%-863% more variance than the top-ranked model validation techniques. On the other hand, out-of-sample bootstrap validation yields the best balance between the bias and variance of estimates in the context of our study. Therefore, we recommend that future defect prediction studies avoid single-repetition holdout validation, and instead, use out-of-sample bootstrap validation.

Index Terms—Defect Prediction Models, Model Validation Techniques, Bootstrap Validation, Cross Validation, Holdout Validation.

1 Introduction

Defect prediction models help software quality assurance teams to effectively focus their limited resources on the most defect-prone software modules. Broadly speaking, a defect prediction model is a statistical regression model or a machine learning classifier that is trained to identify defect-prone software modules. These defect prediction models are typically trained using software metrics that are mined from historical development data that is recorded in software repositories.

Prediction models may provide an unrealistically optimistic estimate of model performance when (re)applied to the same sample with which they were trained. To address this problem, *model validation techniques* (e.g., k-fold cross-validation) are used to estimate the model performance. Model performance is used to (1) indicate how well a model will perform on unseen data [22, 25, 65, 83, 113, 121]; (2) select the top-performing prediction model [36, 54, 62, 73, 77, 107, 115]; and (3) combine several prediction models [7, 91, 111, 117].

The conclusions that are derived from defect prediction models may not be sound if the estimated performance is unrealistic or unstable. Such unrealistic and unstable performance estimates could lead to incorrect model selection in practice and inaccurate conclusions for defect prediction studies [70, 105].

Recent research has raised concerns about the *bias* (i.e., how much do the performance estimates differ from the model performance on unseen data?) and *variance* (i.e., how much do performance estimates vary when the experiment is repeated on the same data?) of model validation techniques when they are applied to defect prediction models [34, 68, 73, 77, 92, 114]. An optimal model validation technique would not overestimate or underestimate the model performance on unseen data. Moreover, the performance estimates should not vary greatly when the experiment is repeated. Mittas *et al.* [73] and Turhan *et al.* [114] point out that the random nature of sampling used by model validation techniques may introduce bias. Myrtveit *et al.* [77] point out that a high variance in the performance estimates that are derived from model validation techniques is a critical problem in comparative studies of prediction models.

To assess the risk that defect prediction datasets pose with respect to producing unstable results, we analyze the number of *Events Per Variable* (EPV) in publicly-available defect datasets. Models that are trained using datasets where the EPV is low (i.e., below 10) are especially susceptible to unstable results. We find that 77% of defect prediction datasets have EPV values below 10, and thus are highly susceptible to producing unstable results. Hence, selecting an appropriate model validation technique is a critical experimental design choice.

Therefore, in this paper, we explore the bias and variance of model validation techniques in both high-risk (i.e., EPV=3) and low-risk (i.e., EPV=10) contexts. Based on an analysis of 256 defect prediction studies that

- *C. Tantithamthavorn and K. Matsumoto are with the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. E-mail: {chakkrit-t,matumoto}@is.naist.jp.*
- *S. McIntosh is with the Department of Electrical and Computer Engineering, McGill University, Canada. E-mail: shane.mcintosh@mcgill.ca.*
- *A. E. Hassan is with the School of Computing, Queen's University, Canada. E-mail: ahmed@cs.queensu.ca.*

Manuscript received August 9, 2015; revised May 24, 2016.

were published from 2000-2011, we select the 12 most popular model validation techniques for our study. The selected techniques include holdout, cross-validation, and bootstrap family techniques. We evaluate 3 types of classifiers, i.e., probability-based (i.e., naïve bayes), regression-based (i.e., logistic regression) and machine learning (i.e., random forest) classifiers. Through a case study of 18 systems spanning both proprietary and open source domains, we address the following two research questions:

(RQ1) Which model validation techniques are the least biased for defect prediction models?
Irrespective of the type of classifier, the out-of-sample bootstrap tends to provide the least biased performance estimates in terms of both threshold-dependent (e.g., precision) and threshold-independent performance measures (e.g., AUC).

(RQ2) Which model validation techniques are the most stable for defect prediction models?
Irrespective of the type of classifier, the ordinary bootstrap is the most stable model validation technique in terms of threshold-dependent and threshold-independent performance measures.

Furthermore, we derive the following practical guidelines for future defect prediction studies:

1) **The single holdout family of model validation techniques should be avoided**, since we find that single holdout validation tends to produce performance estimates with 46%-229% more bias and 53%-863% more variance than the top-ranked model validation techniques.
2) **Researchers should use the out-of-sample bootstrap model validation techniques**, since we find that out-of-sample bootstrap validation is less prone to bias and variance than holdout or cross-validation counterparts in the sparse data contexts that are present in many publicly-available defect datasets.

To the best of our knowledge, this paper is the first work to examine: (1) a large collection of model validation techniques, especially bootstrap validation, which has only rarely been explored in the software engineering literature; (2) the bias and variance of model validation techniques for defect prediction models; and (3) the distribution of Events Per Variable (EPV) of publicly-available defect datasets (Section 2). Furthermore, we introduce (4) the Scott-Knott Effect Size Difference (ESD) test—an enhancement of the standard Scott-Knott test (which cluster distributions into statistically distinct ranks [90]), which makes no assumptions about the underlying distribution and takes the effect size into consideration (Section 5.8.1).

***Paper organization*.** Section 2 introduces the Events Per Variable (EPV) in a dataset, i.e., a metric that quantifies the risk of producing unstable results, and presents realistic examples to illustrate its potential impact. Section 3 introduces the studied model validation techniques. Section 4 situates this paper with respect to the related work. Section 5 discusses the design of our case study, while Section 6 presents the results with respect to our two research questions. Section 7 provides a broader discussion of the implications of our results, while Section 8 derives practical guidelines for future research. Section 9 discloses the threats to the validity of our study. Finally, Section 10 draws conclusions.

2 Motivating Examples

Mende [67] and Jiang *et al.* [51] point out that model validation techniques may not perform well when using a small dataset. An influential characteristic in the performance of a model validation technique is the number of *Events Per Variable* (EPV) [2, 6, 82, 99], i.e., the ratio of the number of occurrences of the least frequently occurring class of the dependent variable (i.e., the events) to the number of independent variables used to train the model (i.e., the variables). Models that are trained using datasets where the EPV is low (i.e., too few events are available relative to the number of independent variables) are especially susceptible to overfitting (i.e., being fit too closely to the training data) and produce unstable results [18, 82].

In order to assess whether low EPV values are affecting defect prediction studies, we analyze 101 publicly-available defect datasets (see Section 5.1 for details on our selection process for the studied datasets). 76 datasets are downloaded from the Tera-PROMISE repository [69], 12 clean NASA datasets are provided by Shepperd *et al.* [93], 5 datasets are provided by Kim *et al.* [56, 118], 5 datasets are provided by D'Ambros *et al.* [21, 22], and 3 datasets are provided by Zimmermann *et al.* [122].

As is often done in defect prediction studies [21, 22, 68, 86, 106, 122], we create our dependent variable by classifying the modules in these datasets as defective (i.e., #defects $>$ 0) or clean (i.e., #defects $=$ 0). We then calculate the EPV using the number of independent variables that are offered by the studied datasets.

Figure 1 shows the distribution of EPV values in the studied datasets using a beanplot [53]. Beanplots are boxplots in which the horizontal curves summarize the distribution of a dataset. The long vertical black line indicates the median value. Peduzzi *et al.* [82] argue that, in order to avoid unstable results, the EPV of a dataset should be at least 10. Thus, we shade the dataset that fall into the high-risk area in red.

78 out of 101 publicly-available defect datasets (77%) are highly susceptible to producing inaccurate and unstable results. Hence, selecting an appropriate model validation technique is a critical experimental design choice.

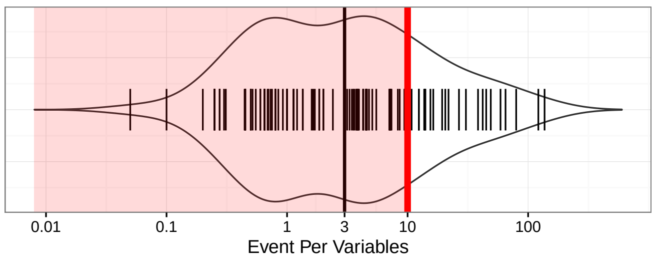


Fig. 1: The distribution of Events Per Variable (EPV) values in publicly-available defect prediction datasets. The black line indicates the median value. The vertical red line indicates the rule-of-thumb EPV value of 10 that is recommended by Peduzzi *et al.* [82]. The red-shaded area indicates the high-risk datasets that do not meet this recommendation (EPV ≤ 10).

Figure 1 shows that 77% of the studied datasets have an EPV value below 10. Furthermore, the median EPV value that we observe is 3, indicating that half of the studied datasets are at a high risk of producing inaccurate and unstable results. Indeed, only 23% of the studied datasets have EPV values that satisfy the recommendation of Peduzzi *et al.* [82].

Below, we present realistic examples to illustrate impact that model validation techniques and EPV can have on the performance of defect prediction models.

2.1 The Risk of Producing Inaccurate Performance Estimate

To assess the risk that model validation techniques pose with respect to producing inaccurate performance estimates, we analyze the performance of a defect prediction model when it is produced by model validation techniques. We select the `JM1` NASA dataset as the subject of our analysis, since it is widely used in different defect prediction studies [37, 47, 48, 51, 62, 66, 98, 116]. We focus on the high-risk EPV context (i.e., EPV$=3$) because Figure 1 shows that 50% of the 101 studied defect datasets have an EPV value below 3. Thus, we select a sample from the `JM1` dataset such that the EPV is 3. This sampling yields a dataset with 300 observations (i.e., 63 defective and 237 clean modules). We train a defect prediction model using logistic regression [20] with the implementation that is provided by the `glm` R function [85]. We measure the performance of the defect prediction models using the Area Under the receiver operator characteristic Curve (AUC) [39]. Finally, we apply 12 different model validation techniques in order to produce performance estimates (see Section 3 for details on our selection process for the studied model validation techniques).

The performance estimates that are produced by model validation techniques vary by up to 15 percentage points. We find that the AUC performance estimates that are produced by model validation techniques vary from 0.58 to 0.73. Indeed, the ordinary bootstrap

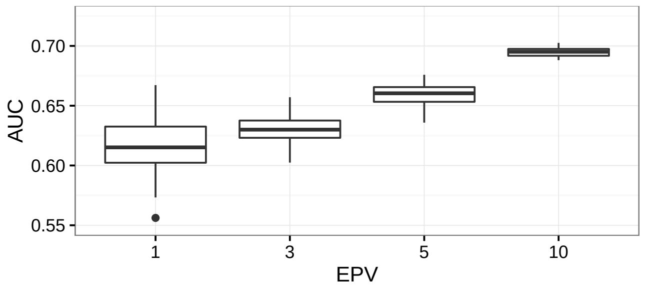


Fig. 2: The distribution of performance estimates that are produced by the repeated 10-fold cross-validation at different EPV contexts when the experiment is repeated 100 times.

produces an estimated AUC of 0.73, while the holdout 50% validation produces an estimated AUC of 0.58. This suggests that defect prediction models may produce inaccurate performance estimates if the wrong different model validation technique is selected. However, it is not clear which model validation techniques provide the most accurate performance estimates in low EPV contexts. Hence, we evaluate each of our research questions across a variety of model validation techniques.

2.2 The Risk of Producing Unstable Results in Defect Datasets

To assess the risk that defect datasets pose with respect to producing unstable performance estimates, we analyze the variation in performance estimates that are produced by defect prediction models when the experiments are repeated at different EPV contexts. Similar to Section 2.1, we use the `JM1` NASA dataset as the subject of our analysis. We train defect prediction models using logistic regression [20] and measure the AUC performance. In order to assess whether datasets at different EPV values are affecting performance estimates, we draw sample defect datasets of different EPV values (i.e., EPV$= 1, 3, 5, 10$). Since we preserve the original rate of defective modules in these samples (i.e., 21%), our samples are of 100, 300, 500, and 1,000 modules for an EPV value of 1, 3, 5, and 10, respectively (see Section 5.2 for details on sample size generation). We apply the repeated 10-fold cross-validation, since it is one of the most commonly-used model validation techniques in defect prediction studies (see Section 3 for details on our literature analysis). In order to investigate the variation of performance estimates, we repeat the experiment 100 times. Figure 2 shows the distribution of the 100 AUC performance estimates for each EPV context.

Performance estimates that are produced by a model that is trained in a low-risk EPV context (i.e., EPV$= 10$) are more stable than that of a model that is trained in a high-risk EPV context (i.e., EPV$= 3$). Figure 2 shows that the performance estimates produced by the model that is trained in a low-risk EPV context (i.e., EPV$= 10$) vary from 0.69 to 0.70, while the performance estimates

TABLE 1: Summary of model validation techniques.

Family	Technique	Training sample	Testing sample	Estimated performance	Iteration(s)
Holdout	Holdout 0.5	50% of original	Independent: 50% of original	A single estimate	1
	Holdout 0.7	70% of original	Independent: 30% of original	A single estimate	1
	Repeated Holdout 0.5	50% of original	Independent: 50% of original	Average of performance of several samples	100
	Repeated Holdout 0.7	70% of original	Independent: 30% of original	Average of performance of several samples	100
Cross-validation	Leave-one-out	N-1 of original	Independent: Subject that is not included in the training sample	Average of performance of several samples	N
	2-Fold	50% of original	Independent: 50% of original	Average of performance of several samples	2
	10-Fold	90% of original	Independent: 10% of original	Average of performance of several samples	10
	10 $\times$ 10-Fold	90% of original	Independent: 10% of original	Average of performance of several samples	100
Bootstrapping	Ordinary	Bootstrap	Original	Average of performance of several samples	100
	Optimism-reduced	Bootstrap	Original	Apparent[†] - optimism	100
	Out-of-sample	Bootstrap	Independent: the training subjects that are not sampled in bootstrap	Average of performance of several samples	100
	.632 Bootstrap	Bootstrap	Independent: the training subjects that are not sampled in bootstrap	0.368 $\times$ Apparent[†] + 0.632 $\times$ average(out-of-sample)	100

[†]*Apparent performance* is computed from a model that is trained and tested on the original sample.

produced by the model that is trained in a high-risk EPV context (i.e., EPV= 3) vary from 0.56 to 0.67. Thus, model validation techniques may help to counter the risk of low EPV datasets through built-in repetition (e.g., M bootstrap iterations or k folds of cross-validation). However, it is not clear which model validation techniques provide the most stable performance estimates in such low EPV contexts. Hence, we evaluate each of our research questions in both high-risk (i.e., EPV=3) and low-risk (i.e., EPV=10) contexts.

3 Model Validation Techniques in Defect Prediction Literature

There are a plethora of model validation techniques available. Since it is impractical to study all of these techniques, we would like to select a manageable, yet representative set of model validation techniques for our study. To do so, we analyze the defect prediction literature in order to identify the commonly used model validation techniques.

We begin by selecting 310 papers that were published between 2000-2011 from two literature surveys of defect prediction—208 papers from the survey of Hall *et al.* [38] and 102 papers from the survey of Shihab [94]. After eliminating duplicate papers, we are left with 256 unique defect prediction studies for analysis.

We read the 256 papers in order to identify the most commonly-used model validation techniques in defect prediction research. We find that 38 of the studied papers needed to be excluded from our analysis because they did not train defect prediction models. For example, many of the excluded papers performed correlation analyses, which does not require a model validation technique. Furthermore, another 35 papers also needed to be excluded because they did not use any model validation technique. Finally, our analysis below describes our findings with respect to the 183 papers that used model validation techniques.

89 studies (49%) use k-fold cross-validation, 83 studies (45%) use holdout validation, 10 studies (5%) use leave-one-out cross-validation, and 1 study (0.5%) uses bootstrap validation.

Below, we describe each studied model validation technique. Table 1 provides an overview of the three families of model validation techniques that we select based on our study of the defect prediction literature.

3.1 Holdout Validation

Holdout validation randomly splits a dataset into training and testing corpora according to a given proportion (e.g., 30% holdout for testing). The training corpus is only used to train the model, while the testing corpus is only used to estimate the performance of the model. Prior work has shown that holdout validation is statistically inefficient because much of the data is not used to train the prediction model [31, 33, 40, 75, 100, 102]. Moreover, an unfortunate split of the training and testing corpora may cause the performance estimate of holdout validation to be misleading. To reduce the bias and variance of holdout validation results, prior studies suggest that it be applied in a repeated fashion [9, 80, 109, 119, 120]. In this paper, we study the commonly used variant of repeated holdout validation, where the entire process is repeated 100 times.

3.2 Cross-Validation

Cross-validation extends the idea of holdout validation by repeating the splitting process several times. In this paper, we study the *k-fold cross-validation* technique, which randomly partitions the data into k folds of roughly equal size where each fold contains roughly the same proportions of the defective ratio [35, 102]. A single fold is used for the testing corpus, and the remaining $k-1$ folds are used for the training corpus. The process of splitting is repeated k times, using each of the k folds as the testing corpus once. The average of the k results is used to estimate the true model performance.

The advantage of k-fold cross-validation is that all of the data is at one point used for both training and testing. However, selecting an appropriate value for k presents a challenge. In this paper, we explore two popular k values (i.e., $k = 2$ and $k = 10$).

While the cross-validation technique is known to be nearly unbiased, prior studies have shown that it can produce unstable results for small samples [11, 45, 51]. To improve the variance of cross-validation results, the entire cross-validation process can be repeated several times. In this paper, we study the commonly used variant of the 10-fold cross-validation where the entire process is repeated 10 times (i.e., 10 $\times$ 10-fold cross-validation).

Leave-One-Out Cross Validation (LOOCV) is the extreme case of k-fold cross-validation, where k is equal to the total number of observations (n). A classifier is trained n times using $n-1$ observations, and the one remaining observation is used for testing. In a simulation experiment [13], and an experiment using effort estimation datasets [57], prior work has shown that LOOCV is the most unbiased model validation technique.

We considered two approaches to estimate model performance when using LOOCV in the defect prediction context (i.e., classifying if a module is defective or clean):

(1) Computing performance metrics once for each iteration. However, threshold-dependent performance measures achieve unrealistic results. Take, for example, the precision metric—when the one testing observation is a defective module, the precision value will either 0% (meaning the model suggests that the module is clean) or 100% (the model suggests that the module is defective). Alternatively, when the one testing observation is a clean module, the precision value is undefined because the denominator (i.e., #true positives + #false positives) is zero.

(2) Computing performance metrics using the N predicted values all at once. While this approach avoids the pitfalls of approach 1, this approach will produce a single performance value. Thus, using this approach, we cannot measure the variance of performance estimates. Furthermore, this approach yields bias values that are not comparable to the other studied model validation techniques, since they are not based on a distribution.

By considering the trade-offs of the considered approaches, we opt to apply LOOCV using approach 1 to only the Brier score (see Section 5.6.2) because it can be computed using a single observation for testing [40, 43, 84].

3.3 Bootstrap Validation

The bootstrap is a powerful model validation technique that leverages aspects of statistical inference [33, 40, 43, 100]. In this paper, we study four variants of the bootstrap. We describe each variant below.

The *ordinary bootstrap* was proposed by Efron *et al.* [31]. The bootstrap process is made up of two steps:

(Step 1) A bootstrap sample of size N is randomly drawn with replacement from an original dataset that is also of size N.

(Step 2) A model is trained using the bootstrap sample and tested using the original sample.

These two steps are repeated M times to produce a distribution of model performance measurements from which the average is reported as the performance estimate. The key intuition is that the relationship between the studied dataset and the theoretical population from which it is derived is asymptotically equivalent to the relationship between the bootstrap samples and the studied dataset.

The *optimism-reduced bootstrap* is an enhancement to the ordinary bootstrap that is used to correct for upward bias [30]. The enhancement alters Step 2 of the ordinary bootstrap procedure. A model is still trained using the i^{th} bootstrap sample, but the model is tested twice—once using the original sample and again using the bootstrap sample from which the model was trained. The optimism of the model is estimated by subtracting the performance (P) of the model when it is applied to the i^{th} bootstrap sample from the performance of the model when it is applied to the original sample (see Equation 1). Optimism calculations are repeated M times to produce a distribution from which the average optimism is derived.

$$\text{Optimism} = \frac{1}{M}\sum_{i=1}^{M}(P^{\text{boot}}_{\text{boot}(i)} - P^{\text{orig}}_{\text{boot}(i)}) \tag{1}$$

Finally, a model is trained using the original sample and tested on the original sample, which yields an unrealistically inflated performance value. The average optimism is subtracted from that performance value to obtain the performance estimate (see Equation 2).

$$\text{Optimism-reduced} = P^{\text{orig}}_{\text{orig}} - \text{Optimism} \tag{2}$$

The *out-of-sample bootstrap*, another enhancement to the ordinary bootstrap, is used to leverage the unused rows from the bootstrap samples. Similar to the optimism-reduced bootstrap, Step 2 of the ordinary bootstrap procedure is altered. A model is still trained using the drawn bootstrap sample, but rather than testing the model on the original sample, the model is instead tested using the rows that do not appear in the bootstrap sample [29]. On average, approximately 36.8% of the rows will not appear in the bootstrap sample, since the bootstrap sample is drawn with replacement. Again, the entire bootstrap process is repeated M times, and the average out-of-sample performance is reported as the performance estimate.

The *.632 bootstrap*, an enhancement to the out-of-sample bootstrap, is designed to correct for the downward bias in performance estimates [29]. Similar to the optimism-reduced bootstrap, a model is first trained using the original sample and tested on the same original sample to obtain an "apparent" performance value. This yields an unrealistically inflated performance value, which is combined with the upwardly-biased estimate from the out-of-sample bootstrap as follows:

$$\text{.632 Bootstrap} = \frac{1}{M}\sum_{i=1}^{M}(0.368 \times P^{\text{orig}}_{\text{orig}} + 0.632 \times P^{-\text{boot}(i)}_{\text{boot}(i)}) \tag{3}$$

The two constants in Equation 3 (i.e., 0.632 and 0.368) are carefully selected. The constants are selected because the training corpus of the out-of-sample bootstrap will have approximately 63.2% of the unique observations from the original dataset and the testing corpus will have 36.8% (i.e., $100\% - 63.2\%$) of the observations. Furthermore, since the out-of-sample bootstrap tends to underestimate and the apparent performance tends to overestimate, 0.632 is used to correct the downward bias of the out-of-sample bootstrap and 0.368 is used to correct the upward bias of the apparent performance.

4 Related Work & Research Questions

Defect prediction models may produce an unrealistic estimation of model performance when inaccurate and unreliable model validation techniques are applied. Such inaccurate and unreliable model validation techniques could lead to incorrect model selection in practice and unstable conclusions of defect prediction studies.

Recent research has raised concerns about the bias of model validation techniques when applied to defect prediction models [34, 68, 70, 73, 77, 114]. The *bias* of a model validation technique is often measured in terms of the difference between a performance estimate that is derived from a model validation technique and the model performance on unseen data. A perfectly unbiased model validation technique will produce a performance estimate that is equivalent to the model performance on unseen data. Mittas *et al.* [73], Turhan *et al.* [114], and Myrtveit *et al.* [77] point out that the random nature of sampling that is employed by model validation techniques is a potential source of bias. Gao *et al.* [34] point out that an unfortunate division of training and testing corpora may also introduce bias.

Plenty of prior studies have explored the bias of model validation techniques in other research domains. However, these studies have arrived at contradictory conclusions. Indeed, some studies conclude that the bootstrap achieves the least biased estimate of model performance [8, 11], while others conclude that 10-fold cross-validation achieves the least biased estimate of model performance [58, 75]. Other work concludes that LOOCV should be used to achieve the most accurate estimate of model performance [13, 57].

We suspect that the contradictory conclusions of prior work are largely related to changes in experimental context. For example, some studies conduct simulation experiments [11, 24, 28, 32, 45, 55], while others use empirical data [11, 28, 57, 58, 75].

The lack of consistency in the conclusions of prior work makes it hard to derive practical guidelines about the most appropriate model validation technique to use in defect prediction research. To address this, we formulate the following research question:

(RQ1) Which model validation techniques are the least biased for defect prediction models?

In addition to the bias of a model validation technique, it is also important to evaluate its variance [51, 67, 70, 77, 78, 92]—the performance estimates should not vary broadly when the experiment is repeated. The *variance* of a model validation technique is typically measured in terms of the variability of the estimated performance, i.e., how much do performance estimates vary when the experiment is repeated on the same data. Myrtveit *et al.* [77] point out that high variance in performance estimates from model validation techniques is a critical challenge in comparative studies of prediction models. Shepperd and Kadoda [92] show that the performance of defect prediction models that are trained using different subsamples that are drawn from the same underlying dataset vary widely. To structure our analysis of the variance of model validation techniques, we formulate the following research question:

(RQ2) Which model validation techniques are the most stable for defect prediction models?

5 Case Study Design

In this section, we discuss our selection criteria for the studied systems and then describe the design of our case study experiment that we perform in order to address our research questions.

5.1 Studied Datasets

In selecting the studied datasets, we identified three important criteria that needed to be satisfied:

- **Criterion 1—Different corpora**: Recent research points out that the performance of defect prediction models may be limited to the dataset from which they are trained [98, 108]. To extend the generality of our conclusions, we choose to train our defect prediction models using systems from different corpora and domains.
- **Criterion 2—Sufficient EPV**: Since we would like to study cases where EPV is low-risk (i.e, = 10) and high-risk (i.e, = 3), the systems that we select for analysis should begin with a low-risk EPV. Our rationale is that we prefer to control for dataset-specific model performance by generating low and high-risk EPV settings using the same dataset. This ensures that a comparison of EPV is derived from the same context. Essentially, we disregard datasets with low initial EPV because we prefer to under-sample to generate low and high-risk EPV datasets from an initially high EPV dataset. For example, if we were to select datasets with an initial EPV of 5, we would need to over-sample the defective class in order to raise the EPV to 10. However, the defective class of a system with an initial EPV of 15 can be under-sampled in order to lower the EPV to 10.
- **Criterion 3—Sane defect data**: Since it is unlikely that more software modules have defects than are free of defects, we choose to study datasets that have a rate of defective modules below 50%.

To satisfy criterion 1, we begin our study using the 101 publicly-available defect datasets described in Section 2.2. To satisfy criterion 2, we exclude the 78 datasets that we found to have an EPV value lower than 10 in Section 2.2. To satisfy criterion 3, we exclude an additional 5 datasets because they have a defective ratio above 50%.

Table 2 provides an overview of the 18 datasets that satisfy our criteria for analysis. To combat potential bias in our conclusions, the studied datasets include proprietary and open source systems with varying size, domain, and defective ratio.

Figure 3 provides an overview of the approach that we apply to each studied system. The crux of our approach is that we calculate model performance on unseen data such that the performance estimates that are derived from model validation techniques can be compared to the model performance on unseen data. The approach is repeated 1,000 times to ensure that the results are robust and that they converge.

Project	System	Defective Ratio	#Files	#Metrics	EPV
NASA	JM1[1]	21%	7,782	21	80
	PC5[1]	28%	1,711	38	12
Proprietary	Prop-1[2]	15%	18,471	20	137
	Prop-2[2]	11%	23,014	20	122
	Prop-3[2]	11%	10,274	20	59
	Prop-4[2]	10%	8,718	20	42
	Prop-5[2]	15%	8,516	20	65
Apache	Camel 1.2[2]	36%	608	20	11
	Xalan 2.5[2]	48%	803	20	19
	Xalan 2.6[2]	46%	885	20	21
Eclipse	Platform 2.0[3]	14%	6,729	32	30
	Platform 2.1[3]	11%	7,888	32	27
	Platform 3.0[3]	15%	10,593	32	49
	Debug 3.4[4]	25%	1,065	17	15
	SWT 3.4[4]	44%	1,485	17	38
	JDT[5]	21%	997	15	14
	Mylyn[5]	13%	1,862	15	16
	PDE[5]	14%	1,497	15	14

[1]Provided by Shepperd *et al.* [93].
[2]Provided by Jureczko *et al.* [52].
[3]Provided by Zimmermann *et al.* [120].
[4]Provided by Kim *et al.* [56].
[5]Provided by D'Ambros *et al.* [21, 22].

TABLE 2: An overview of the studied systems.

5.2 Generate Sample Data

In order to compare the studied model validation techniques, we begin by creating sample and unseen datasets using historical data from a studied dataset. The sample dataset is created in order to train our model for performance on unseen data, while the unseen dataset is used to test it. The sample dataset is also used to train and test models using the studied model validation techniques. The performance on the unseen dataset is compared to the performance estimates that are derived from the studied model validation techniques.

5.2.1 *Sample Dataset*

In order to produce our sample datasets, we select observations with replacement from the input dataset, while controlling for two confounding factors:

(C1) **The defective ratio**: While generating sample datasets, we preserve the defective ratio of the original dataset to ensure that the sample and unseen datasets are representative of the original dataset. Thus, the defective ratio of the unseen datasets are the same as the defective ratio of the original datasets.

(C2) **The EPV**: As mentioned above, we explore high-risk (EPV = 3) and low-risk (EPV = 10) contexts.

Note that by controlling for C1 and C2, we have specified the number of defective modules (and indirectly, the number of clean modules) in the sample dataset. For example, the size of the original JM1 dataset is 7,782 modules. To generate a sample of the JM1 dataset with EPV values of 3 and 10, we need to preserve (1) the defective ratio of 21%; and (2) the 21 variables. Thus, a sample size of the JM1 dataset with an EPV of 3 is 300 modules (i.e., 63 defective and 237 clean modules). Similarly, a sample size of the JM1 dataset with an EPV of 10 is 1000 (i.e., 210 defective and 790 clean modules).

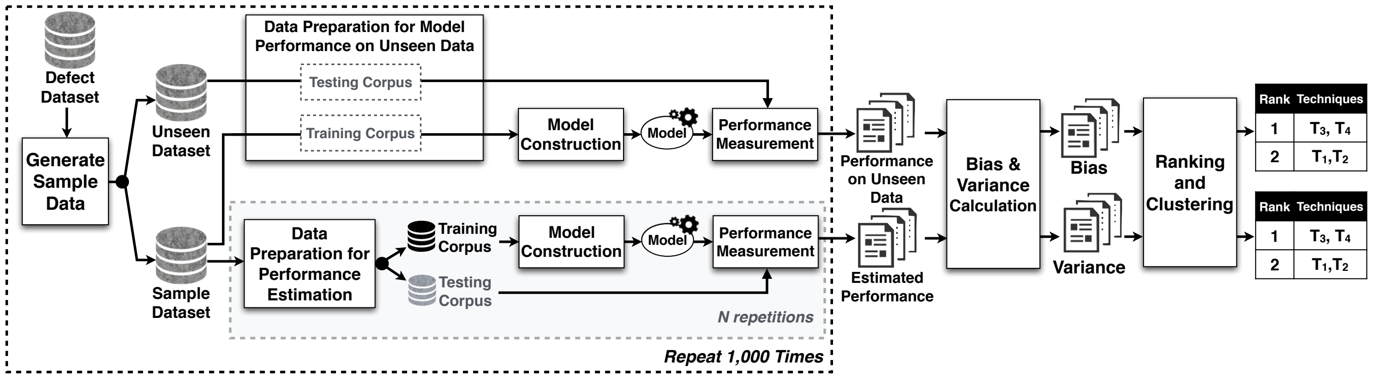


Fig. 3: An overview of the design of our case study experiment.

5.3 Data Preparation for Model Performance on Unseen Data

Unfortunately, the population of the input dataset is unknown, so the actual model performance on unseen data cannot be directly measured. Inspired by previous studies [11, 45, 55, 58, 59], we estimate the model performance using *unseen data*, i.e., the observations from the input dataset that do not appear in the sample dataset. To do so, we use the sample dataset as the training corpus for training a defect prediction model and we use the unseen dataset as the testing corpus to measure the performance of that defect prediction model.

5.4 Data Preparation for Performance Estimation

In order to compare the estimates of the studied model validation techniques to the model performance on unseen data, we apply the model validation techniques to the sample dataset. To split the sample dataset into training and testing corpora, we use: (1) the `createDataPartition` function of the `caret` R package for the holdout family of model validation techniques [60, 61], (2) the `createFolds` function of the `caret` R package for the cross-validation family of model validation techniques [60, 61], and (3) the `boot` function of the `boot` R package for the bootstrap family of model validation techniques [15].

5.5 Model Construction

We select three types of classifiers that are often used in defect prediction literature. These types of classifiers include probability-based (i.e., naïve bayes [27]), regression-based (i.e., logistic regression [20]) and machine learning-based (i.e., random forest [12]) classifiers.

Naïve bayes is a probability-based technique that assumes that all of the predictors are independent of each other [27]. We use the implementation of naïve bayes that is provided by the `naiveBayes` R function [71].

Logistic regression measures the relationship between a categorical dependent variable and one or more independent variables [20]. We use the implementation of logistic regression that is provided by the `glm` R function [85].

Random forest is a machine learning classifier that constructs multiple decision trees from bootstrap samples [12]. Since each tree in the forest may return a different outcome, the final class of a software module is computed by aggregating the votes from all trees. We use the implementation of random forest that is provided by the `randomForest` R function [63].

To ensure that the training and testing corpora have similar characteristics, we do not re-balance or re-sample the training data, as suggested by Turhan [112].

5.5.1 Normality Adjustment

Analysis of the distributions of our independent variables reveals that they are right-skewed. As suggested by previous research [49, 68], we mitigate this skew by log-transforming each independent variable ($ln(x+1)$) prior to using them to train our models.

5.6 Performance Measurement

We apply the defect prediction models that we train using the training corpus to the testing corpus in order to measure their performance. We use both threshold-dependent and threshold-independent performance measures to quantify the performance of our models. We describe each performance metric below.

5.6.1 Threshold-Dependent Performance Measures

When applied to a module from the testing corpus, a defect prediction model will report the probability of that module being defective. In order to calculate the threshold-dependent performance measures, these probabilities are transformed into a binary classification (defective or clean) using a threshold value of 0.5, i.e., if a module has a predicted probability above 0.5, it is considered defective; otherwise, the module is considered clean.

Using the threshold of 0.5, we compute the precision and recall performance measures. *Precision* measures the proportion of modules that are classified as defective, which are actually defective ($\frac{TP}{TP+FP}$). *Recall* measures the proportion of actually defective modules that were classified as such ($\frac{TP}{TP+FN}$). We use the `confusionMatrix` function of the `caret` R package [60, 61] to compute the precision and recall of our models.

5.6.2 Threshold-Independent Performance Measures

Prior research has argued that the precision and recall are unsuitable for measuring the performance of defect prediction models because they: (1) depend on an arbitrarily-selected threshold (typically 0.5) [1, 5, 62, 86, 95], and (2) are sensitive to imbalanced data [23, 44, 64, 67, 103]. Thus, we also use three threshold-independent performance measures to quantify the performance of our defect prediction models.

First, we use the *Area Under the receiver operator characteristic Curve* (AUC) [39] to measure the discrimination power of our models. The AUC measures a classifier's ability to discriminate between defective and clean modules (i.e., do the defective modules tend to have higher predicted probabilities than clean modules?). AUC is computed by measuring the area under the curve that plots true positive rate against the false positive rate while varying the threshold that is used to determine whether a module is classified as defective or not. Values of AUC range between 0 (worst classifier performance) and 1 (best classifier performance).

In addition to the discrimination power, practitioners often use the predicted probabilities to rank the defect-prone files [74, 79, 96, 122]. Shihab *et al.* [96] point out that practitioners often use the predicted probability to make decisions. Mockus *et al.* [74] point out that the appropriate range of probability values is important to make an appropriate decision (e.g., high-reliability systems may require a lower cutoff value than 0.5). However, the AUC does not capture all of the dimensions of a prediction model [26, 40, 100, 101]. To measure the accuracy of the predicted probabilities, we use the Brier score and the calibration slope.

We use the *Brier score* [14, 89] to measure the distance between the predicted probabilities and the outcome. The Brier score is calculated as:

$$\mathrm{B} = \frac{1}{N}\sum_{i=1}^{N}(f_t - o_t)^2 \tag{4}$$

where f_t is the predicted probability, o_t is the outcome for module t encoded as 0 if module t is clean and 1 if it is defective, and N is the total number of modules. The Brier score ranges from 0 (best classifier performance) to 1 (worst classifier performance).

Finally, we use the *calibration slope* to measure the direction and spread of the predicted probabilities [20, 26, 40, 42, 72, 99, 101]. The calibration slope is the slope of a logistic regression model that is trained using the predicted probabilities of our original defect prediction model to predict whether a module will be defective or not [20]. A calibration slope of 1 indicates the best classifier performance and a calibration slope of 0 (or less) indicates the worst classifier performance. We use the `val.prob` function of the `rms` R package [41] to calculate the Brier score, AUC, and calibration slope.

5.7 Bias and Variance Calculation

We calculate each performance measure using the model performance on unseen data and the model validation techniques. In order to address RQ1, we calculate the *bias*, i.e., the absolute difference between the performance that we derive from the model validation techniques and the model performance on unseen data. In order to address RQ2, we calculate the *variance* of the performance measures that we derive from the model validation techniques (in terms of standard deviation).

5.8 Ranking and Clustering

Finally, we group the model validation techniques into statistically distinct ranks according to the their bias and variance using the Scott-Knott Effect Size Difference (ESD) test.

5.8.1 The Scott-Knott Effect Size Difference (ESD) test

The Scott-Knott test [90] uses hierarchical cluster analysis to partition the set of treatment means into statistically distinct groups ($\alpha = 0.05$). Two major limitations of the Scott-Knott test are that (1) it assumes that the data should be normally distributed; and (2) it may create groups that are trivially different from one another. To strengthen the Scott-Knott test, we propose the Scott-Knott Effect Size Difference (ESD) test—a variant of the Scott-Knott test that is normality and effect size aware. The Scott-Knott ESD test will (1) correct the non-normal distribution of an input dataset; and (2) merge any two statistically distinct groups that have a negligible effect size into one group.

Normality correction. The Scott-Knott test assumes that the data under analysis are normally distributed. Thus, we mitigate the skewness by log-transforming each treatment ($ln(x+1)$), since it is a commonly-used transformation technique in software engineering research [49, 68].

Effect size correction. To quantify the effect size, we use Cohen's delta (d) [16], which is the difference between two means divided by the standard deviation of the data:

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s.d.} \tag{5}$$

where the magnitude is assessed using the thresholds provided in Cohen [17], i.e. $|d| < 0.2$ "negligible", $|d| < 0.5$ "small", $|d| < 0.8$ "medium", otherwise "large".

We implement the Scott-Knott ESD test[1] [104] based on the implementation of the Scott-Knott test that is provided by the `ScottKnott` R package [46] and the implementation of Cohen's delta provided by the `effsize` R package [110].

5.8.2 Ranking and Clustering Approach

Figure 4 provides an overview of our approach. First, to identify the least biased model validation techniques, similar to our prior work [36], we perform a double

1. https://github.com/klainfo/ScottKnottESD

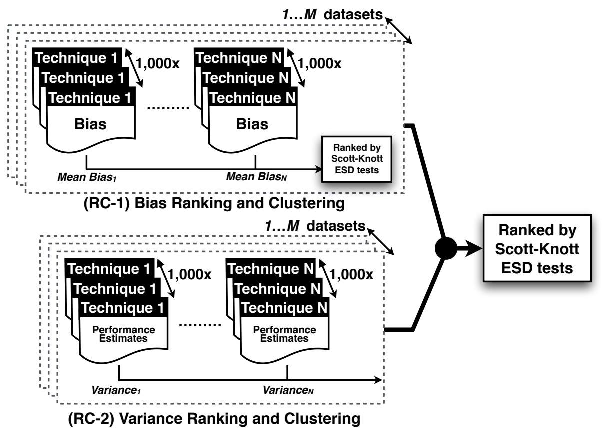


Fig. 4: An overview of our ranking and clustering approach.

Scott-Knott test. We apply a Scott-Knott ESD test on the bias results from the 1,000 iterations that are performed for each studied dataset individually. After performing this first set of Scott-Knott ESD tests, we have a list of ranks for each model validation technique (i.e., one rank from each studied dataset). We provide these lists of ranks to a second run of the Scott-Knott ESD test, which produces a ranking of model validation techniques across all of the studied datasets. We perform the first set of Scott-Knott ESD tests in order to control for dataset-specific model performance, since some datasets may be more or less susceptible to bias than others. Since only one variance value is computed for the 1,000 runs of one technique, only a single Scott-Knott ESD test needs to be applied to the variance values.

6 Case Study Results

In this section, we present the results of our case study with respect to our two research questions.

(RQ1) Which model validation techniques are the least biased for defect prediction models?

In order to address RQ1, we compute the bias in terms of precision, recall, Brier score, AUC, and calibration slope. We then present the results with respect to probability-based (RQ1-a), regression-based (RQ1-b), and machine learning-based (RQ1-c) classifiers. Figure 5 shows the statistically distinct Scott-Knott ESD ranks of the bias of the studied model validation techniques.

RQ1-a: Probability-Based Classifiers

The .632, out-of-sample, and optimism-reduced bootstraps are the least biased model validation techniques for naïve bayes classifiers. Figure 5a shows that the .632, out-of-sample, and optimism-reduced bootstraps are the only model validation techniques that consistently appear in the top Scott-Knott ESD rank in terms of precision, recall, Brier score, AUC, and calibration slope bias in both EPV contexts.

Other techniques appear consistently in the top rank for some metrics, but not across all metrics. For example, the repeated 10-fold and 10-fold cross-validation techniques appear in the top rank in terms of precision, recall, AUC, and Brier score, but not in calibration slope for the high-risk EPV context. We suspect that the calibration slope that we derived from the 10-fold cross-validation is upwardly-biased because of the scarcity of defective modules in the small testing corpus of 10-fold cross-validation (i.e., 10% of the input dataset).

While advanced model validation techniques with built-in repetitions (i.e., repeated holdout validation, cross-validation and bootstrap validation) tend to produce the least biased performance estimates, single-repetition holdout validation tends to produce the most biased performance estimates. Indeed, single-repetition holdout validation tends to produce performance estimates with 46%-229% more bias than the top-ranked model validation techniques, suggesting that researchers should avoid using single holdout validation, since it tends to produce overly optimistic or pessimistic performance estimates. However, repeated holdout validation produces up to 43% less biased performance estimates than single holdout validation does. Indeed, by repeating the holdout validation technique, the amount of bias of the high-risk EPV context is substantially reduced by 30%, 20%, 32%, and 43% for precision, recall, AUC, and calibration slope, respectively. Indeed, single-repetition holdout validation should be avoided.

RQ1-b: Regression-Based Classifier

The out-of-sample bootstrap is the least biased model validation technique for logistic regression classifiers. Figure 5b shows that the out-of-sample bootstrap is the only model validation technique that consistently appears in the top Scott-Knott ESD rank in terms of precision, recall, Brier score, AUC, and calibration slope bias in both EPV contexts.

Other techniques appear consistently in the top rank with respect to the low-risk EPV context, but not the high-risk EPV context. For example, the .632 and optimism-reduced bootstraps, and the 10-fold and repeated 10-fold cross-validation techniques consistently appear in the top rank in the low-risk EPV context, but do not consistently appear in the top ranks of the high-risk EPV context. Indeed, Figure 5b indicates that the .632 bootstrap tends to produce recall estimates with 21% more bias than the top-ranked technique in the high-risk EPV context, while the optimism-reduced bootstrap, 10-fold, and repeated 10-fold cross-validation techniques tend to produce precision estimates that are 29% more biased than the top ranked technique in the high-risk EPV context. Furthermore, the bias in performance estimates in the high-risk EPV context is larger than those of the low-risk EPV context. This indicates that the performance estimates that are derived from the least biased model validation techniques in the high-risk EPV context tend to produce 46% more bias than the least biased model validation techniques in the low-risk EPV context. *A lack of generality across EPV contexts of some model validation techniques and an increase of performance bias in the high-risk EPV context, suggests that selecting a robust model validation technique is an especially important experimental design choice in the high-risk EPV contexts that are commonplace in defect datasets (cf. Section 2).*

RQ1-c: Machine Learning Classifier

The out-of-sample bootstrap and 2-fold cross-validation are the least biased model validation techniques for random forest classifiers. Figure 5c shows that the out-of-sample bootstrap and 2-fold cross-validation techniques consistently appear in the top Scott-Knott ESD rank in terms of precision, recall, Brier score, AUC, and calibration slope bias in both EPV contexts.

While the .632 and optimism-reduced bootstraps are less biased for the naïve bayes and logistic regression classifiers, the .632 and optimism-reduced bootstraps are quite biased for random forest classifiers. We suspect that the upward-bias in the .632 and optimism-reduced bootstraps have to do with the low training error rate of random forest. Since the low training error rate often produces a high apparent performance, the calculation of .632 and optimism-reduced bootstraps, which are partly computed using the apparent performance, are biased. This suggests that the .632 and optimism-reduced bootstraps are not appropriate for classifiers that have low training error rates, such as random forest. This finding is also complementary to Kohavi *et al.* [58], who suggests that repeated 10-fold cross-validation should be used. On the other hand, Kohavi *et al.* [58] did not evaluate the out-of-sample bootstrap. In our analysis, we find that the out-of-sample bootstrap is less-prone to bias than repeated 10-fold cross-validation is.

Irrespective of the type of classifier, the out-of-sample bootstrap tends to provide the least biased performance estimates in terms of both threshold-dependent and threshold-independent performance metrics.

(RQ2) Which model validation techniques are the most stable for defect prediction models?

In order to address RQ2, we compute the variance in terms of precision, recall, Brier score, AUC, and calibration slope. We then present the results with respect to probability-based (RQ2-a), regression-based (RQ2-b), and machine learning-based (RQ2-c) classifiers. Figure 6 shows the statistically distinct Scott-Knott ESD ranks of the variance of the studied model validation techniques.

RQ2-a: Probability-Based Classifier

All variants of the bootstrap, repeated 10-fold cross validation, and repeated holdout validation techniques are the most stable model validation techniques for naïve bayes classifiers. Figure 6a shows that, in addition to the repeated 10-fold cross-validation and the repeated holdout techniques, the .632, optimism-reduced, out-of-sample, and ordinary bootstrap techniques are the only model validation techniques that consistently appear in the top Scott-Knott ESD rank in terms of both precision and recall variance in both EPV contexts.

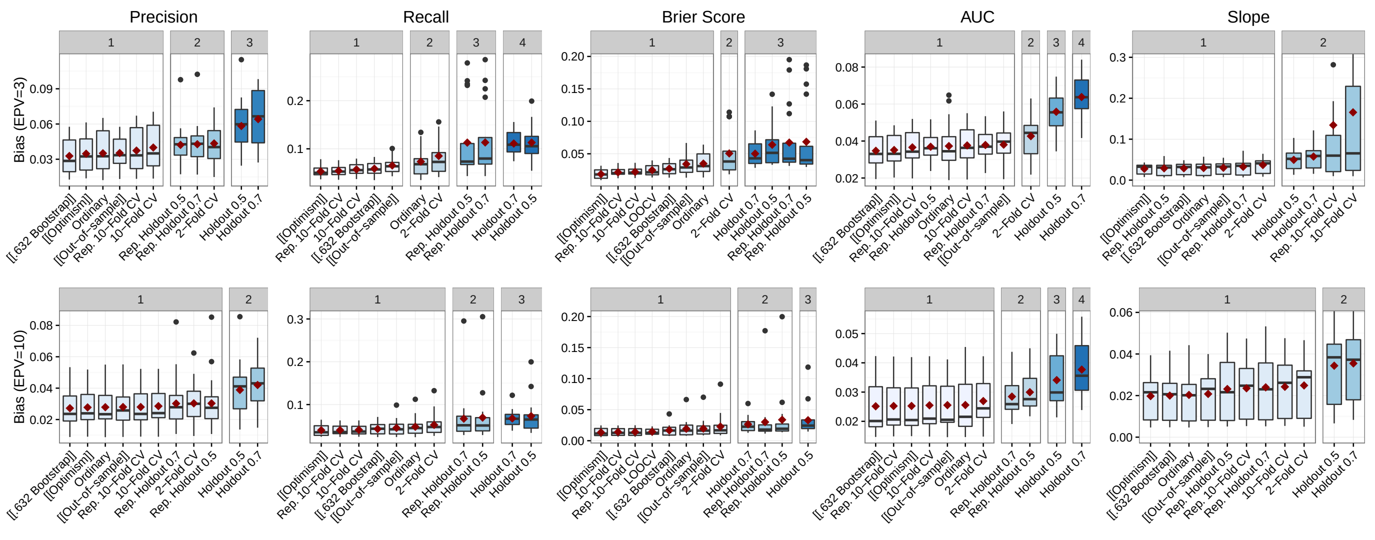
While advanced model validation techniques with built-in repetitions (i.e., repeated holdout validation, cross-validation and bootstrap validation) tend to yield the most stable performance estimates, single holdout validation, which only uses a single iteration, tends to yield the least stable performance estimates. Indeed, single holdout validation tends to produce performance estimates with 53%-863% more variance than the most stable model validation techniques. Moreover, the repeated holdout validation produces 40%-68% more stable performance estimates than single holdout validation does. Indeed, by repeating the holdout validation technique, the amount of variance in the high-risk EPV context is substantially reduced by 45%, 45%, 48%, 40%, 68% for precision, recall, Brier score, AUC, and calibration slope, respectively. Indeed, holdout validation should be avoid unless it is repeated.

RQ2-b: Regression-Based Classifier

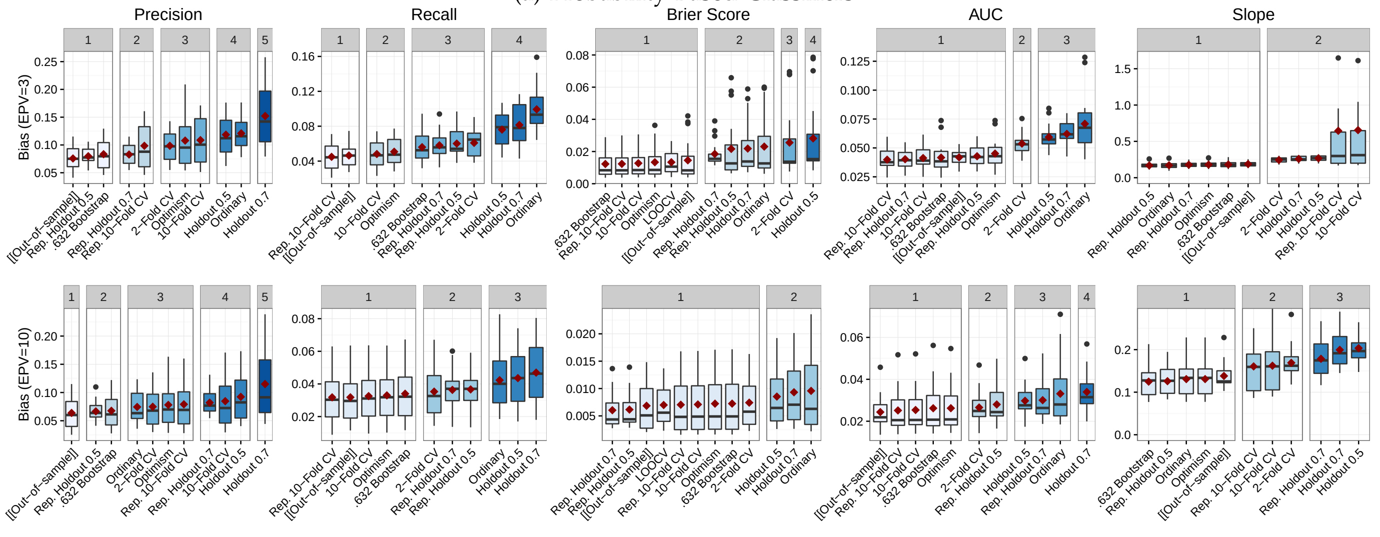
The .632 and ordinary bootstraps are the most stable model validation techniques for logistic regression classifiers. Figure 6b shows that the .632 and ordinary bootstraps are the only model validation techniques that consistently appear in the top Scott-Knott ESD rank in terms of precision, recall, Brier score, AUC, and calibration slope variance in both EPV contexts.

Other techniques appear consistently in the top rank for the low-risk EPV context, but not for high-risk EPV context. For example, the optimism-reduced bootstrap and the 10-fold cross-validation techniques appear consistently in the top rank in terms of threshold-dependent metrics in the low-risk EPV context, but not in the high-risk EPV context. Indeed, Figure 6b shows that there is (on average) a 66% increase in terms of precision variance and 91% increase in terms of recall variance when those techniques are used in the high-risk EPV context. We suspect that the precision and recall that are derived from the cross-validation family are less stable in the high-risk EPV context because an unfortunate split of the training and testing corpora may result in too few defective modules appearing in the testing corpus. Hence, very few correct (or incorrect) predictions can have a very large impact on cross-validation performance estimates.

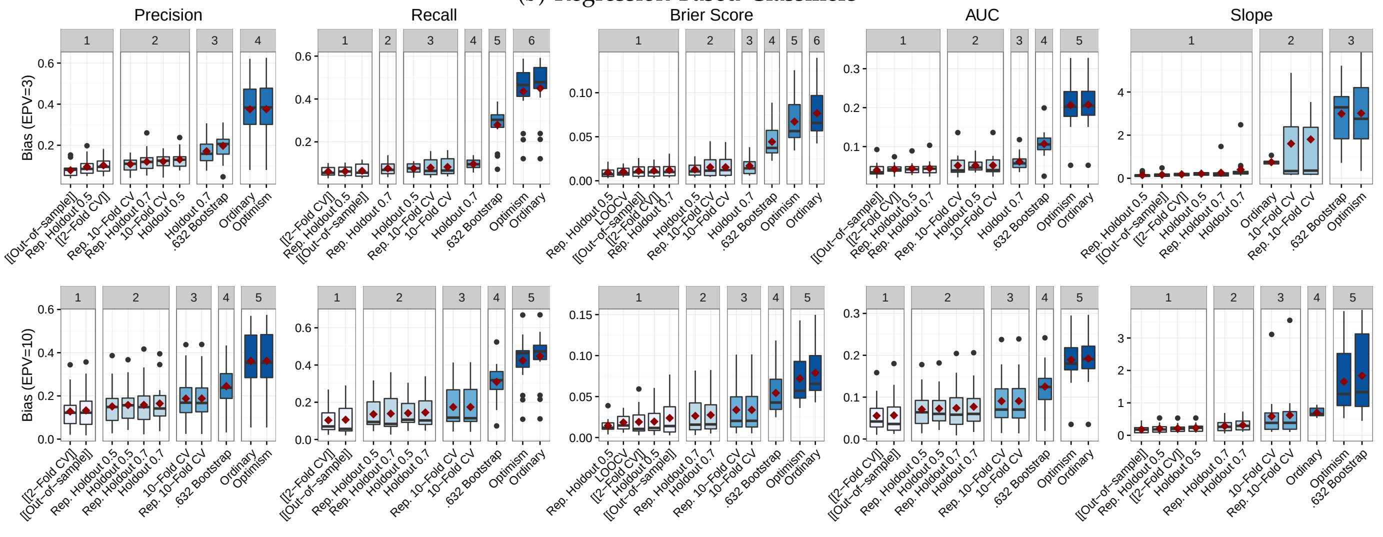
Furthermore, the optimism-reduced and out-of-sample bootstraps, as well as the repeated 10-fold cross-validation techniques appear consistently in the top rank in terms of threshold-independent metrics in the low-risk EPV context, but not in the high-risk EPV context. Indeed, Figure 6b shows that there is (on



(a) Probability-Based Classifiers

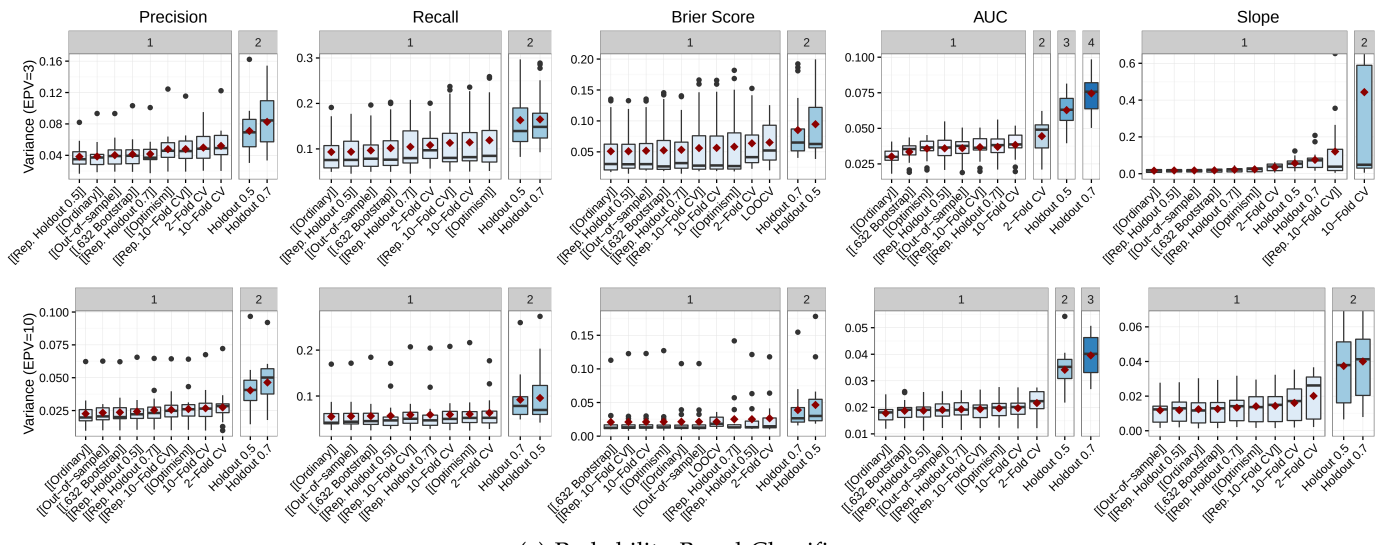


(b) Regression-Based Classifiers

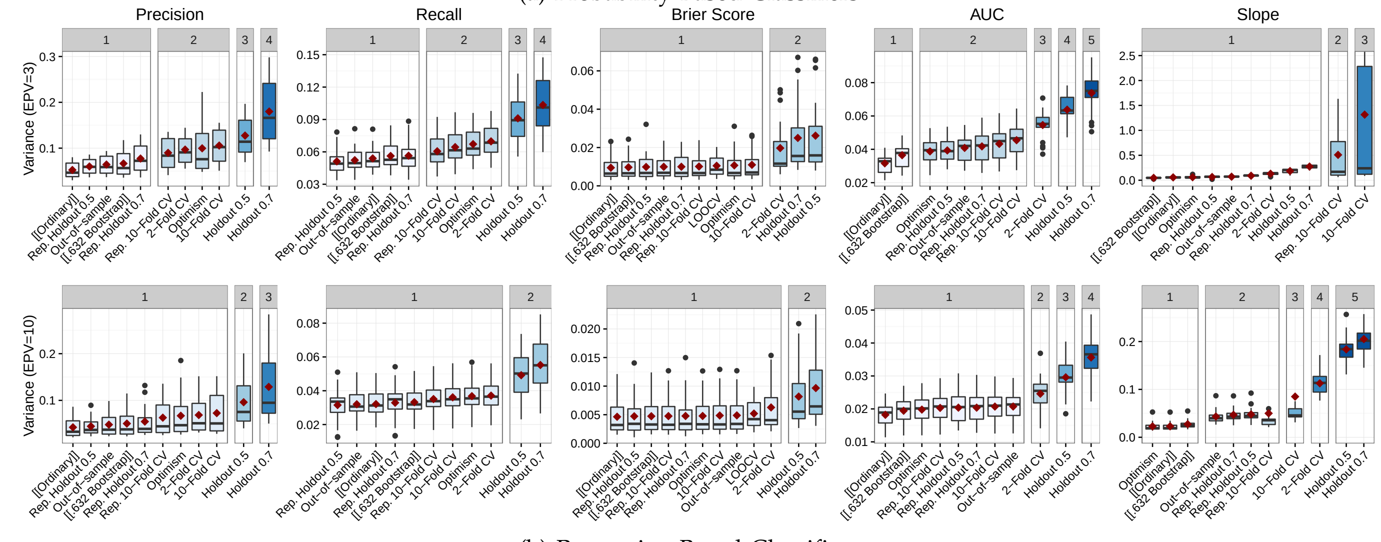


(c) Machine Learning-Based Classifiers

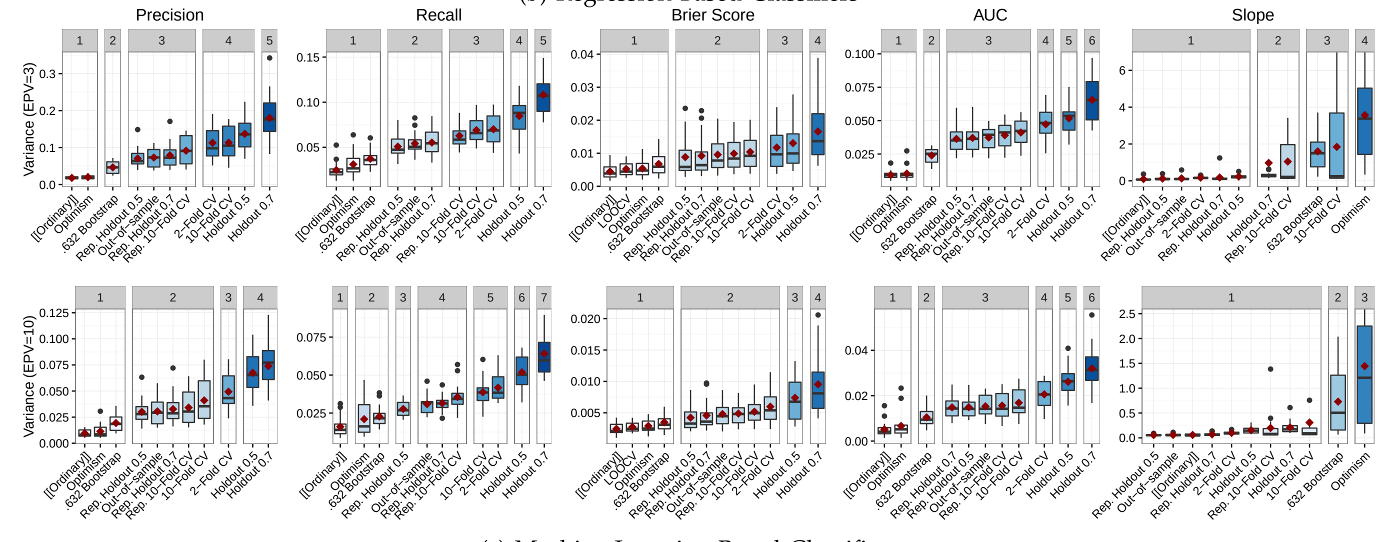
Fig. 5: The Scott-Knott ESD ranking of the *bias* of model validation techniques. The technique in a bracket indicates the top-performing technique for each classifier type. The red diamond indicates the average amount of bias across our studied datasets.



(a) Probability-Based Classifiers



(b) Regression-Based Classifiers



(c) Machine Learning-Based Classifiers

Fig. 6: The Scott-Knott ESD ranking of the *variance* of model validation techniques. The technique in a bracket indicates the top-performing technique for each classifier type. The red diamond indicates the average amount of variance across our studied datasets.

average) a 110% increase in terms of AUC variance when those techniques are used in the high-risk EPV context. We suspect that the AUC is less stable because the AUC derived from these techniques are calculated from a small amount of data in the testing corpus. For example, 10-fold cross-validation is validated on 10% of the original data and the optimism-reduced and out-of-sample bootstraps are validated on 36.8% of the original data. Conversely, the performance estimates when using the ordinary bootstrap that are computed using a sample that has the same size as the original data is likely to produce the most stable performance estimates. On the other hand, the .632 bootstrap, which is an enhancement of the out-of-sample bootstrap (*cf.* Section 4), is also generally robust to the AUC variance. A lack of generality across EPV contexts of some model validation techniques and an increase of performance variance in the high-risk EPV context, indeed, suggest that the EPV plays a major role in the stability of performance estimates.

RQ2-c: Machine Learning-Based Classifier

The ordinary bootstrap is the most stable model validation technique for random forest classifiers. Figure 6c shows that the ordinary bootstrap is the only model validation technique that consistently appears in the top Scott-Knott ESD rank in terms of precision, recall, Brier score, AUC, and calibration slope variance in both EPV contexts.

We suspect that the most stable performance estimates that are being produced by the ordinary bootstrap because of the statistical inference properties of the bootstrap itself. Indeed, the key intuition is that the relationship between the performance that is derived from a studied dataset and the true performance that would be derived from the population of defect datasets is asymptotically equivalent to the relationship between the performance that is derived from a bootstrap sample and the performance that is derived from the studied dataset.

In addition to yielding highly biased results for random forest classifiers (*cf.* RQ1-c), the .632 and optimism-reduced bootstraps also tend to produce the least stable performance estimates in terms of calibration slope. Indeed, this further supports our suspicion that the .632 and optimism-reduced bootstraps are not appropriate for classifiers that have low training error rates, such as random forest.

Irrespective of the type of classifier, the ordinary bootstrap is the most stable model validation technique in terms of threshold-dependent and threshold-independent metrics.

7 Discussion

7.1 Leave-One-Out Cross-Validation

In Section 3, we find that 5% of the analyzed defect prediction literature uses the LOOCV technique. However, since LOOCV only performs one prediction per fold, it is incompatible with the majority of our performance metrics (see Section 3.2). Thus, we exclude it from our core analysis of our research questions. On the other hand, the bias and variance of the LOOCV technique can be assessed using the Brier score.

Leave-one-out cross-validation is among the least biased and most stable model validation techniques in terms of Brier score. Indeed, Figures 5 and 6 show that LOOCV appears in the top Scott-Knott ESD ranks in terms of Brier score bias and variance. This finding is very much complementary to recent work on software effort estimation [57], which argues for the use of LOOCV for assessing software effort estimation models.

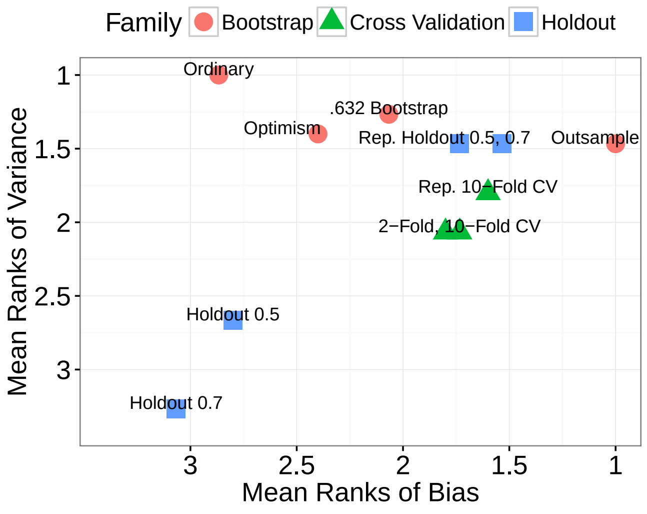
7.2 External Validation

We estimate the model performance using *unseen data*, which may not be realistic, since it is derived from a random sampling. While this random splitting approach is common in other research areas [11, 45, 55, 58, 59], recent studies in software engineering tend to estimate the performance of defect models using data from subsequent releases [50, 86–88]. We perform an additional analysis in order to investigate whether the performance of classifiers that is derived from model validation techniques is similar to the performance of classifiers that are trained using one release and tested on the next one. We then repeat all of our experiments to compute the bias and variance in terms of precision, recall, Brier score, AUC, and calibration slope of naïve bayes, logistic regression, and random forest classifiers.

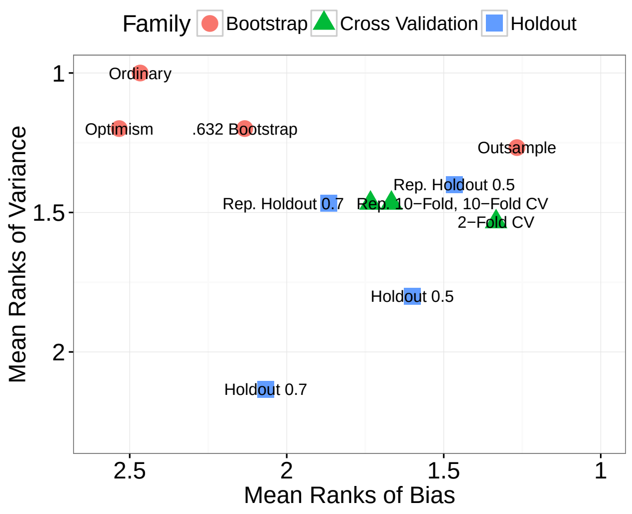
The out-of-sample bootstrap still yields the best balance between bias and variance of performance estimates. Based on our analysis of 5 releases of Proprietary systems, 2 releases of Apache Xalan, and 3 releases of the Eclipse Platform, Figure 7b shows that, similar to Section 6, the out-of-sample bootstrap tends to provide a good balance between bias and variance of performance estimates in the high-risk EPV context. We also find a consistent finding in the low-risk EPV context, indicating that internally validated estimates of model performance could accurately be obtained with the out-of-sample bootstrap.

7.3 The Computational Cost of Model Validation Techniques

Note that the computational cost (i.e., the number of iterations) of each model validation technique varies (see Table 1). For example, the performance that is derived from holdout validation requires a single iteration, while the performance derived from bootstrap validations requires several iterations. Thus, for complex classifiers (e.g., random forest), the advanced model validation



(a) Internal Validation (Section 6)



(b) External Validation (Section 7.2)

Fig. 7: A scatter plot of the mean Scott-Knott ESD ranks in terms of bias and variance among 5 performance metrics, 3 studied classifiers, and 18 studied systems for the high-risk EPV context (EPV= 3) when using two different types of unseen data, i.e., Figure 7a uses the observations from the input dataset that do not appear in the sample dataset; and Figure 7b uses the next software release. The techniques that appear in the upper-right corner are top-performers.

techniques (e.g., bootstrap validation and repeated 10-fold cross-validation) may require a large amount of total execution time. However, since each iteration is independent, they can be computed simultaneously. Therefore, researchers can speed up the process using multi-core or multi-thread processors.

In addition to the impact that the computational cost of model validation techniques has on the total execution time, it may affect the bias and variance of performance estimates.

The number of iterations impacts the variance of the performance estimates, but not the bias. Figure 6 and Figure 7b show that the bootstrap family and repeated 10-fold cross-validation, which requires several iterations, tend to yield the most stable performance estimates, indicating that increasing the number of iterations tends to produce more stable performance estimates. This suggests that the repeated 10-fold cross-validation is more preferable than the 10-fold cross-validation. Conversely, we find that the number of iterations tends to have less of an impact on the bias. For example, all variants of bootstrap and repeated 10-fold cross-validation that have the same amount of computational cost (i.e., 100 iterations), have different amounts of bias on performance estimates. Hence, we suspect that the impact that model validation techniques have on the bias of performance estimates may have more to do with their calculation techniques (e.g., type or size of training and testing data) than the number of iterations.

8 Practical Guidelines

Our experimental results indicate that the choice of model validation technique can influence model performance estimates, especially for complex classifiers. An inappropriate model validation technique could lead to misleading conclusions. In this section, we offer practical guidelines for future defect prediction studies:

(1) Single-repetition holdout validation should be avoided.
Section 3 shows that 45% of the surveyed defect prediction literature uses the holdout model validation technique. However, Section 6 shows that the single holdout family is consistently the most biased and least stable model validation technique. Indeed, our results show that the single holdout family tends to produce performance estimates with 46%-229% more bias and 53%-863% more variance than the top-ranked model validation techniques. Hence, researchers should avoid using the holdout family, since it may produce overly optimistic or pessimistic performance estimates and yield results that are difficult to reproduce. Section 6 shows that the repetitions of holdout validation technique substantially reduce the amount bias and variance, suggesting that the repetition must be applied for the holdout validation. Nonetheless, the repeated holdout validation techniques still produce performance estimates with more variance than the ordinary bootstrap.

(2) Researchers should use out-of-sample bootstrap validation instead of cross-validation or holdout.

Although other families of model validation techniques are comparable to the out-of-sample bootstrap techniques in the low-risk EPV context, Figure 7a shows that, in high-risk EPV contexts, the out-of-sample bootstrap technique is less biased (Figure 5) and more stable (Figure 6) than the other studied model validation techniques. Moreover, Figure 7b also confirms that internally validated estimates of model performance can accurately be obtained with the out-of-sample bootstrap. Furthermore, since Section 4 shows that many publicly-available defect datasets suffer from a high-risk EPV, we recommend that researchers use the out-of-sample bootstrap in future defect prediction studies.

9 Threats to Validity

Like any empirical study design, experimental design choices may impact the results of our study [105]. However, we perform a highly-controlled experiment to ensure that our results are robust. Below, we discuss threats that may impact the results of our study.

9.1 Construct Validity

The datasets that we analyze are part of several collections (e.g., NASA and PROMISE), which each provide different sets of software metrics. Since the metrics vary, this is another point of variation between the studied systems that could impact the results. On the other hand, the variation of metrics also strengthens the generalization of our results—our findings are not bound to one specific set of software metrics.

The conclusions of Section 2.2 are based on a rule-of-thumb EPV value that is suggested by Peduzzi *et al.* [82], who argue that, in order to avoid unstable results, the EPV of a dataset should be at least 10. However, their conclusions are not derived from defect prediction context. Thus, we plan to explore an optimal EPV value for defect prediction context in future work.

The design our experiments of Section 5 take the approaches that have been used in the other research areas into consideration [11, 45, 55, 58, 59]. Although our approach is built upon these successful previous studies, we made several improvements. For example, Kohavi *et al.* [58] adopt a simple holdout approach to generate sample and unseen corpora. However, our approach is based on the bootstrap concept, which leverages aspects of statistical inference. Moreover, we also maintain the same defective ratio as the original dataset to ensure that the sample and unseen datasets are representative of the dataset from which they were generated. We also repeat the experiment several times to ensure that the results are robust and that they converge.

Randomness may introduce bias. To combat this bias, we repeat the experiment 1,000 times to ensure that the results converge. While we have reported the results using 1,000 repetitions, we also repeated the experiment using 300 repetitions, and found consistent results. Thus, we believe the results have converged, and an increase in the number of repetitions would not alter the conclusions of our study.

In our experiment, parameter settings of classification techniques may impact the performance of defect prediction models [4, 19, 97, 105, 107]. However, only 2 out of the 3 studied classification techniques (i.e., naïve bayes and random forest) require at least one parameter setting. Our recent work [107] finds that both naïve bayes and random forest classifiers are relatively insensitive to the choice of parameter settings. Hence, we believe that the choice of parameter settings would not alter the conclusions of our study.

Although the Scott-Knott Effect Size Difference (ESD) test (Section 5.8.1) uses log-transformations [10] to mitigate the skewness of data distribution, we aware that other data transformation techniques (i.e., Box-Cox transformation [7, 10], Blom transformation [73]) may yield different results [81]. To ensure that the results are robust, we repeat the experiment 1,000 times, as suggested by Arcuri *et al.* [3]. Therefore, according to the Central Limit Theorem, the distribution of bias and variance results will be approximately normally distributed if the sample size is large enough [76]. We believe that other data transformation techniques would not alter the conclusions of our study.

Finally, all variants of bootstrap validation techniques are repeated 100 times (see Table 1). While our results show that the 100 repetitions of the out-of-sample bootstrap validation produce the least biased and most stable performance estimates, we also repeated the experiment using 200 repetitions, and found consistent results. Hence, we believe that 100 repetitions are sufficient.

9.2 Internal Validity

While we find that the out-of-sample bootstrap tends to provide the least bias and most stable performance estimates across measures in both EPV contexts, our findings are limited to 5 performance measures—2 threshold-dependent performance measures (i.e., precision and recall) and 3 threshold-independent performance measures (i.e., AUC, Brier score, calibration slope). Other performance measures may yield different results. However, 3 of the studied performance measures (i.e., precision, recall, and AUC) are commonly used measures in defect prediction studies. We also explore another 2 performance measures (i.e., Brier score and calibration slope) that capture other important dimensions of the performance of a classifier (*cf.* Section 5.6.2), which have not yet been explored in the software engineering literature. Nonetheless, other performance measures can be explored in future work. This paper provides a detailed methodology for others who would like to re-examine our findings using unexplored performance measures.

9.3 External Validity

We study a limited number of datasets in this paper. Thus, our results may not generalize to all software systems. To combat potential bias, we analyze 18 datasets from both proprietary and open source domains. Nonetheless, additional replication studies are needed.

We study only one technique for each classification family, i.e., probability-based, regression-based, and machine learning-based families. Thus, our results may not generalize to other classification techniques of each family. Hence, additional evaluation of families of classification techniques are needed.

10 Conclusions

Defect prediction models help software quality assurance teams to effectively allocate their limited resources to the most defect-prone software modules. Model validation techniques are used to estimate how well a model will perform on unseen data, i.e., data other than that which was used to train the model. However, the validity and the reliability of performance estimates rely heavily on the employed model validation techniques. Yet, little is known about which model validation techniques tend to produce the least biased and most stable performance estimates.

To that end, this paper investigates the bias and the variance of 12 model validation techniques in terms of 2 threshold-dependent performance measures (i.e., precision and recall) and 3 threshold-independent performance measures (i.e., Brier score, AUC, and calibration slope). Since many publicly available defect datasets are at a high risk of producing unstable results (see Section 2.2), we explore the bias and variance of model validation techniques in both high-risk (i.e., EPV=3) and low-risk (i.e., EPV=10) contexts. We evaluate 3 types of classifiers that include probability-based (i.e., naïve bayes), regression-based (i.e., logistic regression) and machine learning-based (i.e., random forest) classifiers. Through a case study of 18 datasets spanning both open source and proprietary domains, we make the following observations:

– The out-of-sample bootstrap is the least biased model validation technique in terms of both threshold-dependent and threshold-independent performance measures.
– The ordinary bootstrap is the most stable model validation technique in terms of both threshold-dependent and threshold-independent performance measures.
– In datasets with a high-risk of producing unstable results (i.e., the EPV is low), the out-of-sample bootstrap family yields a good balance between bias and variance of model performance.
– The single-repetition holdout validation consistently yields the most biased and least stable estimates of model performance.

To mitigate the risk of result instability that is present in many defect datasets, we recommend that future defect prediction studies avoid using the single-repetition holdout validation and instead opt to use the out-of-sample bootstrap model validation technique.

Acknowledgments

The authors would like to thank Frank E. Harrell Jr. for fruitful discussion on the state of the art and practice of model validation techniques and performance measures. This study would not have been possible without the data shared in the Tera-PROMISE repository (curated by Tim Menzies, Mitch Rees-Jones, Rahul Krishna and Carter Pape [69]), as well as the data shared by Shepperd *et al.* [93], Zimmermann *et al.* [122], Kim *et al.* [56, 118], and D'Ambros *et al.* [21, 22], as well as High Performance Computing (HPC) systems provided by the Compute Canada[2] and the Centre for Advanced Computing at Queen's University.[3] This work was supported by the JSPS Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers: Interdisciplinary Global Networks for Accelerating Theory and Practice in Software Ecosystem, the Grant-in-Aid for JSPS Fellows (No. 16J03360), and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] D. G. Altman, B. Lausen, W. Sauerbrei, and M. Schumacher, "Dangers of using "optimal" cutpoints in the evaluation of prognostic factors," *Journal of the National Cancer Institute*, vol. 86, no. 1994, pp. 829–835, 1994.

[2] D. G. Altman and P. Royston, "What do we mean by validating a prognostic model?" *Statistics in Medicine*, vol. 19, no. 4, pp. 453–473, 2000.

[3] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Prooceedings of the International Conference on Software Engineering (ICSE)*, 2011, pp. 1–10.

[4] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *Search Based Software Engineering*. Springer, 2011, pp. 33–47.

[5] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[6] P. C. Austin and E. W. Steyerberg, "Events per variable (EPV) and the relative performance of different strategies for estimating the out-of-sample validity of logistic regression models." *Statistical methods in medical research*, vol. 0, no. 0, pp. 1–13, 2014.

[7] M. Azzeh, A. B. Nassif, and L. L. Minku, "An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation," *Journal of Systems and Software*, vol. 103, pp. 36–52, 2015.

[8] C. Beleites, R. Baumgartner, C. Bowman, R. Somorjai, G. Steiner, R. Salzer, and M. G. Sowa, "Variance reduction in estimating classification error using sparse datasets," *Chemometrics and Intelligent Laboratory Systems*, vol. 79, no. 1-2, pp. 91–100, 2005.

2. http://www.computecanada.ca
3. http://cac.queensu.ca/

[9] C. Bird, B. Murphy, and H. Gall, "Don't Touch My Code! Examining the Effects of Ownership on Software Quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE'11)*, 2011, pp. 4–14.

[10] G. E. Box and D. R. Cox, "An analysis of transformations," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 211–252, 1964.

[11] U. M. Braga-Neto and E. R. Dougherty, "Is cross-validation valid for small-sample microarray classification?" *Bioinformatics*, vol. 20, no. 3, pp. 374–80, 2004.

[12] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[13] L. Breiman and P. Spector, "Submodel Selection and Evaluation in Regression. The X-Random Case," *International Statistical Institute*, vol. 60, no. 3, pp. 291–319, 1992.

[14] G. W. Brier, "Verification of Forecasets Expressed in Terms of Probability," *Monthly Weather Review*, vol. 78, no. 1, pp. 25–27, 1950.

[15] A. Canty and B. Ripley, "boot: Bootstrap r (s-plus) functions," http://CRAN.R-project.org/package=boot, 2014.

[16] J. Cohen, "Statistical power analysis for the behavioral sciences," 1988.

[17] ——, "A power primer." *Psychological bulletin*, vol. 112, no. 1, p. 155, 1992.

[18] J. Concato, A. R. Feinstein, and T. R. Holford, "The Risk of Determining Risk with Multivariable Models," *Annals of Internal Medicine*, vol. 118, no. 3, pp. 201–210, 1993.

[19] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "How effective is tabu search to configure support vector regression for effort estimation?" in *Proceedings of the International Conference on Predictive Models in Software Engineering (PROMISE)*, 2010, p. 4.

[20] D. R. Cox, "Two Further Applications of a Model for Binary Regression," *Biometrika*, vol. 45, no. 3, pp. 562–565, 1958.

[21] M. D'Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," in *Prooceedings of the Working Conference on Mining Software Repositories (MSR'10)*, 2010, pp. 31–41.

[22] ——, "Evaluating Defect Prediction Approaches : A Benchmark and an Extensive Comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[23] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the International Conference on Machine Learning (ICML'06)*, 2006, pp. 233–240.

[24] A. C. Davison and P. Hall, "On the bias and variability of bootstrap and cross-validation estimates of error rate in discrimination problems," *Biometrika*, vol. 79, no. 2, pp. 279–284, 1992.

[25] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens, "Data Mining Techniques for Software Effort Estimation: A Comparative Study," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 375–397, 2012.

[26] S. den Boer, N. F. de Keizer, and E. de Jonge, "Performance of prognostic models in critically ill cancer patients - a review." *Critical care*, vol. 9, no. 4, pp. R458–R463, 2005.

[27] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine learning*, vol. 29, no. 2-3, pp. 103–130, 1997.

[28] E. Dougherty, C. Sima, J. Hua, B. Hanczar, and U. Braga-Neto, "Performance of Error Estimators for Classification," *Bioinformatics*, vol. 5, no. 1, pp. 53–67, 2010.

[29] B. Efron, "Estimating the error rate of a prediction rule: some improvements on cross-validation," *Journal of the American Statistical Association*, vol. 78, no. 382, pp. 316–331, 1983.

[30] ——, "How Biased Is the Apparent Error Rate of a Prediction Rule ?" *Journal of the American Statistical Association*, vol. 81, no. 394, pp. 461–470, 1986.

[31] B. Efron and G. Gong, "A Leisurely Look at the Bootstrap , the Jackknife , and Cross-Validation," *The American Statistician*, vol. 37, no. 1, pp. 36–48, 1983.

[32] B. Efron and R. Tibshirani, "Improvements on Cross-Validation: The 632+ Bootstrap Method," *Journal of the American Statistical Association*, vol. 92, no. 438, pp. 548–560, 1997.

[33] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. Boston, MA: Springer US, 1993.

[34] K. Gao and T. M. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *IEEE Transactions on Reliability*, vol. 56, no. 2, pp. 223–236, 2007.

[35] S. Geisser, "Sample Reuse Method The Predictive with Applications," *Journal of the American Statistical Association*, vol. 70, no. 350, pp. 320–328, 1975.

[36] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings of The 37th International Conference on Software Engineering*, 2015, pp. 789–800.

[37] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pp. 417–428, 2004.

[38] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, Nov. 2012.

[39] J. a. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve." *Radiology*, vol. 143, no. 4, pp. 29–36, 1982.

[40] F. E. Harrell Jr., *Regression Modeling Strategies*, 1st ed. Springer, 2002.

[41] ——, "rms: Regression modeling strategies," http://CRAN.R-project.org/package=rms, 2015.

[42] F. E. Harrell Jr., K. L. Lee, and D. B. Mark, "Tutorial in Biostatistics Multivariable Prognostic Models : Issues in Developing Models, Evaluting Assumptions and Adequacy, and Measuring and Reducing Errors," *Statistics in Medicine*, vol. 15, pp. 361–387, 1996.

[43] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani, *The elements of statistical learning*. Springer, 2009.

[44] H. He and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[45] A. Isaksson, M. Wallman, H. Göransson, and M. Gustafsson, "Cross-validation and bootstrapping are unreliable in small sample classification," *Pattern Recognition Letters*, vol. 29, no. 14, pp. 1960–1965, 2008.

[46] E. G. Jelihovschi, J. C. Faria, and I. B. Allaman, *The ScottKnott Clustering Algorithm*, Universidade Estadual de Santa Cruz - UESC, Ilheus, Bahia, Brasil, 2014.

[47] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, no. August, pp. 561–595, 2008.

[48] Y. Jiang, B. Cukic, and T. Menzies, "Fault Prediction using Early Lifecycle Data," *The 18th IEEE International Symposium on Software Reliability (ISSRE '07)*, pp. 237–246, 2007.

[49] ——, "Can data transformation help in the detection of fault-prone modules?" in *Proceedings of the workshop on Defects in Large Software Systems (DEFECTS'08)*, 2008, pp. 16–20.

[50] Y. Jiang, B. Cukic, T. Menzies, and J. Lin, "Incremental development of fault prediction models," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 10, pp. 1399–1425, 2013.

[51] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance Analysis in Software Fault Prediction Models," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE'09)*, 2009, pp. 99–108.

[52] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the International Conference on Predictive Models in Software Engineering (PROMISE'10)*, 2010, pp. 9:1–9:10.

[53] P. Kampstra, "Beanplot: A Boxplot Alternative for Visual Comparison of Distributions," *Journal of Statistical Software*, vol. 28, no. 1, pp. 1–9, 2008.

[54] T. Khoshgoftaar and N. Seliya, "Comparative Assessment of Software Quality Classification Techniques : An Empirical Case Study," *Empirical Software Engineering*, vol. 9, pp. 229–257, 2004.

[55] J.-H. Kim, "Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap," *Computational Statistics & Data Analysis*, vol. 53, no. 11, pp. 3735–3745, 2009.

[56] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proceeding of the international conference on Software engineering (ICSE '11)*, 2011, pp. 481–490.

[57] E. Kocaguneli and T. Menzies, "Software effort models should be assessed via leave-one-out validation," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1879–1890, 2013.

[58] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995, pp. 1137–1143.

[59] R. Kohavi and D. H. Wolpert, "Bias plus variance decomposition for zero-one loss functions," in *Proceedings of the International Conference on Machine Learning (ICML'96)*, 1996.

[60] M. Kuhn, "Building Predictive Models in R Using caret Package," *Journal of Statistical Software*, vol. 28, no. 5, 2008.

[61] ——, "caret: Classification and regression training," http://CRAN.R-project.org/package=caret, 2015.

[62] S. Lessmann, S. Member, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction : A Proposed Framework and Novel Findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[63] A. Liaw and M. Wiener, "randomForest: Breiman and Cutler's random forests for classification and regression," http://CRAN.R-project.org/package=randomForest, 2014.

[64] Y. Ma and B. Cukic, "Adequate and Precise Evaluation of Quality Models in Software Engineering Studies," in *Proceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE'07)*, 2007, pp. 1–9.

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

[66] T. Mende and R. Koschke, "Effort-Aware Defect Prediction Models," *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR'10)*, pp. 107–116, Mar. 2010.

[67] T. Mende, "Replication of Defect Prediction Studies: Problems, Pitfalls and Recommendations," in *Proceedings of the International Conference on Predictive Models in Software Engineering (PROMISE'10)*, 2010, pp. 1–10.

[68] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[69] T. Menzies, C. Pape, R. Krishna, and M. Rees-Jones, "The Promise Repository of Empirical Software Engineering Data," http://openscience.us/repo, 2015.

[70] T. Menzies and M. Shepperd, "Special issue on repeatable results in software engineering prediction," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 1–17, 2012.

[71] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, F. Leisch, C.-C. Chang, and C.-C. Lin, "e1071: Misc Functions of the Department of Statistics (e1071), TU Wien," http://CRAN.R-project.org/package=e1071, 2014.

[72] M. E. Miller, S. L. Hui, and W. M. Tierney, "Validation techniques for logistic regression models." *Statistics in medicine*, vol. 10, no. 8, pp. 1213–1226, 1991.

[73] N. Mittas and L. Angelis, "Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 537–551, 2013.

[74] A. Mockus and D. M. Weiss, "Predicting Risk of Software Changes," *Bell Labs Technical Journal*, vol. 5, no. 6, pp. 169–180, 2000.

[75] A. M. Molinaro, R. Simon, and R. M. Pfeiffer, "Prediction error estimation: a comparison of resampling methods." *Bioinformatics*, vol. 21, no. 15, pp. 3301–3307, Aug. 2005.

[76] D. S. Moore, *The basic practice of statistics*. WH Freeman New York, 2007, vol. 2.

[77] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380–391, 2005.

[78] I. Myrtveit and E. Stensrud, "Validity and reliability of evaluation procedures in comparative studies of effort prediction models," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 23–33, 2011.

[79] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005.

[80] N. Nagappan, B. Murphy, and V. R. Basili, "The Influence of Organizational Structure on Software Quality : An Empirical Case Study," *Organization*, pp. 521–530, 2008.

[81] J. W. Osborne, "Improving your data transformations: Applying the box-cox transformation," *Practical Assessment, Research & Evaluation*, vol. 15, no. 12, pp. 1–9, 2010.

[82] P. Peduzzi, J. Concato, E. Kemper, T. R. Holford, and A. R. Feinstein, "A Simulation Study of the Number of Events per Variable in Logistic Regression Analysis," *Journal of Clinical Epidemiology*, vol. 49, no. 12, pp. 1373–1379, 1996.

[83] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing Privacy and Utility in Cross-Company Defect Prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1054–1068, 2013.

[84] K. L. Priddy and P. E. Keller, *Artificial Neural Networks: An Introduction*. SPIE Press, 2005, vol. 68.

[85] R Core Team, "R: A language and environment for statistical computing," http://www.R-project.org/, R Foundation for Statistical Computing, Vienna, Austria, 2013.

[86] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the International Conference on Software Engineering (ICSE'13)*, 2013, pp. 432–441.

[87] F. Rahman, I. Herraiz, D. Posnett, and P. Devanbu, "Sample Size vs . Bias in Defect Prediction," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (FSE'13)*, 2013, pp. 147–157.

[88] F. Rahman, D. Posnett, and P. T. Devanbu, "Recalling the "Imprecision" of Cross-Project Defect Prediction Categories and Subject Descriptors," in *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE'12)*, 2012, p. 61.

[89] K. Rufibach, "Use of Brier score to assess binary predictions," *Journal of Clinical Epidemiology*, vol. 63, no. 8, pp. 938–939, 2010.

[90] A. J. Scott and M. Knott, "A Cluster Analysis Method for Grouping Means in the Analysis of Variance," *Biometrics*, vol. 30, no. 3, pp. 507–512, 1974.

[91] G. Seni and J. F. Elder, *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*, Jan. 2010, vol. 2, no. 1.

[92] M. Shepperd and G. Kadoda, "Comparing software prediction techniques using simulation," *IEEE Transactions on Software Engineering*, vol. 27, no. 11, pp. 1014–1022, 2001.

[93] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, 2013.

[94] E. Shihab, "An Exploration of Challenges Limiting Pragmatic Software Defect Prediction," Ph.D. dissertation, Queen's University, 2012.

[95] E. Shihab, A. E. Hassan, B. Adams, and Z. M. Jiang, "An industrial study on the risk of software changes," in *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE'12)*, 2012, p. 62.

[96] E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan, "High-Impact Defects: A Study of Breakage and Surprise Defects Categories and Subject Descriptors," in *Proceedings of the European Conference on Foundations of Software Engineering (ESEC/FSE'11)*, 2011, pp. 300–310.

[97] L. Song, L. L. Minku, and X. Yao, "The impact of parameter tuning on software effort estimation using learning machines,"

in *Proceedings of the International Conference on Predictive Models in Software Engineering (PROMISE)*, 2013, p. 9.

[98] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A General Software Defect-Proneness Prediction Framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356–370, May 2011.

[99] E. W. Steyerberg, M. J. Eijkemans, F. E. Harrell Jr., and J. D. Habbema, "Prognostic modelling with logistic regression analysis: a comparison of selection and estimation methods in small data sets," *Statistics in Medicine*, vol. 19, pp. 1059–1079, 2000.

[100] E. W. Steyerberg, *Clinical prediction models: a practical approach to development, validation, and updating*. Springer Science & Business Media, 2008.

[101] E. W. Steyerberg, A. J. Vickers, N. R. Cook, T. Gerds, N. Obuchowski, M. J. Pencina, and M. W. Kattan, "Assessing the performance of prediction models: a framework for some traditional and novel measures," *Epidemiology*, vol. 21, no. 1, pp. 128–138, 2010.

[102] M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," *Journal of the Royal Statistical Society*, vol. 36, no. 2, pp. 111–147, 1974.

[103] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online Defect Prediction for Imbalanced Data," in *Prooceedings of the International Conference on Software Engineering (ICSE'15)*, 2015, pp. 99–108.

[104] C. Tantithamthavorn, "ScottKnottESD: An R package of The Scott-Knott Effect Size Difference (ESD) test," https://github.com/klainfo/ScottKnottESD, 2015.

[105] ——, "Towards a better understanding of the impact of experimental components on defect prediction modelling," in *Companion Proceedings of the International Conference on Software Engineering (ICSE)*, 2016, pp. 867–870.

[106] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models," in *Proceedings of the International Conference on Software Engineering*, 2015, pp. 812–823.

[107] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2016, pp. 321–332.

[108] ——, "Comments on "Researcher Bias: The Use of Machine Learning in Software Defect Prediction"," *IEEE Transactions on Software Engineering (TSE)*, 2016.

[109] a. Tarvo, "Using Statistical Models to Predict Software Regressions," *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, 2008.

[110] M. Torchiano, "effsize: Efficient effect size computation," http://CRAN.R-project.org/package=effsize, 2015.

[111] A. Tosun, "Ensemble of Software Defect Predictors : A Case Study," *Esem*, pp. 318–320, 2008.

[112] B. Turhan, "On the dataset shift problem in software engineering prediction models," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 62–74, 2011.

[113] B. Turhan, G. Kocak, and A. Bener, "Data mining source code for locating software bugs: A case study in telecommunication industry," *Expert Systems with Applications*, vol. 36, no. 6, pp. 9986–9990, 2009.

[114] B. Turhan, A. Tosun Misirli, and A. Bener, "Empirical evaluation of the effects of mixed project data on learning defect predictors," *Information and Software Technology*, vol. 55, no. 6, pp. 1101–1118, 2013.

[115] H. Wang, T. M. Khoshgoftaar, and A. Napolitano, "A Comparative Study of Ensemble Feature Selection Techniques for Software Defect Prediction," *2010 Ninth International Conference on Machine Learning and Applications*, pp. 135–140, 2010.

[116] T. Wang and W.-h. Li, "Naive Bayes Software Defect Prediction Model," *2010 International Conference on Computational Intelligence and Software Engineering*, no. 2006, pp. 1–4, 2010.

[117] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[118] R. Wu, H. Zhang, S. Kim, and S. C. Cheung, "ReLink : Recovering Links between Bugs and Changes," in *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE'11)*, 2011, pp. 15–25.

[119] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the International Conference on Software Engineering (ICSE'08)*, 2008, pp. 531–540.

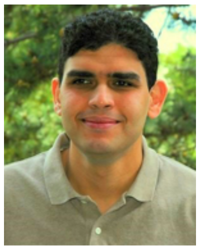
[120] ——, "Predicting subsystem failures using dependency graph complexities," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE'07)*, 2007, pp. 227–236.

[121] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction," in *Proceedings of the European Software Engineering Conference and the symposium on the Foundations of Software Engineering (FSE'09)*, 2009, pp. 91–100.

[122] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting Defects for Eclipse," in *Proooceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE'07)*, 2007, pp. 9–20.

Chakkrit Tantithamthavorn is a Ph.D. candidate and JSPS Research Fellow (DC2) at Nara Institute of Science and Technology (NAIST) under the supervision of Professor Kenichi Matsumoto. He was also a visiting researcher at Queen's University, Canada working with Professor Ahmed E. Hassan from September 2013 to April 2016. His work has been published at several top-tier software engineering venues, such as IEEE Transactions on Software Engineering (TSE) and the International Conference on Software Engineering (ICSE). His Ph.D. thesis aims to improve the fundamentals of predictive modelling for software engineering (e.g., defect prediction models) in order to produce more accurate predictions and reliable insights. His research interests also include empirical software engineering and mining software repositories (MSR). He received the B.E. degree (2012) in computer engineering from Kasetsart University, Thailand, and the M.E. degree (2014) in Information Science from Nara Institute of Science and Technology, Japan. More about him and his work is available online at http://chakkrit.com.

Shane McIntosh is an assistant professor in the Department of Electrical and Computer Engineering at McGill University. He received his Bachelor's degree in Applied Computing from the University of Guelph and his MSc and PhD in Computer Science from Queen's University. In his research, Shane uses empirical software engineering techniques to study software build systems, release engineering, and software quality. His research has been published at several top-tier software engineering venues, such as the International Conference on Software Engineering (ICSE), the International Symposium on the Foundations of Software Engineering (FSE), and the Springer Journal of Empirical Software Engineering (EMSE). More about Shane and his work is available online at http://shanemcintosh.org/.



Ahmed E. Hassan is the NSERC/RIM Industrial Research chair in Software Engineering for Ultra Large Scale Systems, School of Computing, Queen's University. He spearheaded the organization and creation of the Mining Software Repositories (MSR) conference and its research community. He co-edited special issues of the IEEE Transactions on Software Engineering and the Journal of Empirical Software Engineering on the MSR topic. Early tools and techniques developed by his team are already integrated into products used by millions of users worldwide. His industrial experience includes helping architect the Blackberry wireless platform at RIM, and working for IBM Research at the Almaden Research Lab and the Computer Research Lab at Nortel Networks. He is the named inventor of patents at several jurisdictions around the world including the United States, Europe, India, Canada, and Japan. He is a member of the IEEE.

Kenichi Matsumoto is a professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. He received the Ph.D. degree in information and computer sciences from Osaka University. His research interests include software measurement and software process. He is a fellow of the IEICE, a senior member of the IEEE, and a member of the ACM, and the IPSJ.