An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges

Chakkrit Tantithamthavorn
The University of Adelaide, Australia.
chakkrit.tantithamthavorn@adelaide.edu.au

Ahmed E. Hassan
Queen's University, Canada.
ahmed@cs.queensu.ca

ABSTRACT

Over the past decade with the rise of the Mining Software Repositories (MSR) field, the modelling of defects for large and long-lived systems has become one of the most common applications of MSR. The findings and approaches of such studies have attracted the attention of many of our industrial collaborators (and other practitioners worldwide). At the core of many of these studies is the development and use of analytical models for defects. In this paper, we discuss common pitfalls and challenges that we observed as practitioners attempt to develop such models or reason about the findings of such studies. The key goal of this paper is to document such pitfalls and challenges so practitioners can avoid them in future efforts. We also hope that other academics will be mindful of such pitfalls and challenges in their own work and industrial engagements.

CCS CONCEPTS

• **Software and its engineering** → *Software development process management*; • **Information systems** → *Data analytics*;

KEYWORDS

Empirical Software Engineering, Software Analytics, Mining Software Repositories, Experimental Design, Defect Modelling

ACM Reference Format:
Chakkrit Tantithamthavorn and Ahmed E. Hassan. 2018. An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges. In *Proceedings of 40th International Conference on Software Engineering: Software Engineering in Practice Track, Gothenburg, Sweden, May 27-June 3 2018 (ICSE-SEIP '18),* 10 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Over the past decades, analytical modelling of defects has become widespread within many software organizations, like Bell Labs [48], AT&T [52], Turkish Telecommunication [84], Microsoft Research [49–51, 83, 85], Google [42], Blackberry [62], Cisco [46, 56, 57, 69], IBM [8], Sony Mobile [64], and Amisoft [55]. Today many software organizations advocate the need for analytics specialist within software teams. For example, data scientists are now a major part of Microsoft's software teams [13, 31–33]. Hence, analytical modelling skills play a crucial role in software development organizations worldwide nowadays.

The successful deployment of defect analytical models in practice relies heavily on an in-depth understanding of many intricate details that are associated with the analytical modelling process. However, due to the ubiquitous access to statistical and machine learning toolkits (e.g., R, Weka, Scikit-learn) nowadays, many users of such modelling toolkits have limited knowledge about many important details (e.g., often missing to deal with correlated variables in defect models). Such limited knowledge often leads to major problems which in turn invalidate the results of studies and lead to the failure of defect analytics projects in practice [37].

While research efforts have paid attention to guidelines for conducting and reporting empirical studies [27, 35, 38], case study research [58, 59], systematic literature review [34, 36, 54], replication studies [9, 65], grounded-theory [67], controlled experiments with human participants [39], and statistical tests [1], the pitfalls and guidelines for analytical modelling of defects has not attracted enough attention from researchers.

Working closely with many industrial partners over the past decade (e.g., BlackBerry [30, 62], Sony Mobile [64], Avaya [62, 63], and a large number of small-to-medium size companies) on developing or deploying defect models in practice, or in explaining to practitioners the findings of published analytical models in their own settings, we have faced many challenges first hand. Hence, in this paper, we discuss common pitfalls and challenges that we faced. The key goal of our paper is to document such pitfalls and challenges so practitioners and researchers can avoid them in future efforts. We also hope that academics will be mindful of such pitfalls and challenges in their own work and industrial engagements.

Paper organization. Section 2 provides a working definition and an overview of the defect modelling process. Section 3 discusses the pitfalls that are associated with the development of defect models. Section 4 discusses challenges about defect modelling. Section 5 concludes the paper.

2 DEFECT MODELLING IN A NUTSHELL

2.1 Definition and Goal

Defect modelling refers to the development of a statistical or machine learning model that is trained using historical project data. The goal of such modelling is to discover meaningful patterns, confirm hypotheses, explore relationships, predict, and/or understand actionable insights in order to better inform decisions related to the various phases of software practice (e.g., software development, software maintenance, and software management). Defect modelling is a core instrument in software analytics.

ICSE-SEIP '18, May 27-June 3 2018, Gothenburg, Sweden

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

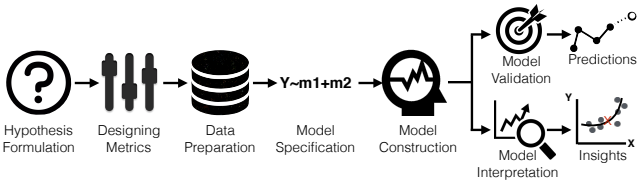


Figure 1: The key steps in the defect modelling process.

2.2 An Overview of the Defect Modelling Process

The defect modelling process involves several key steps (see Figure 1):

(Step-1) Hypothesis Formulation. One must formulate a set of hypotheses pertaining to a phenomena of interest (e.g., whether complex code increases project risk).

(Step-2) Designing Metrics. One must determine a set of metrics which operationalize the hypothesis of interest (e.g., McCabe's Cyclomatic Complexity (`CC`) to capture code complexity, and the number of field-reported bugs to capture the risk that is associated with a module (`bugs`)). One must also control for known confounding factors (e.g., large modules (`size`) are likely to have more bugs). We note that one needs to use control metrics to ensure that findings are not due to confounding factor. While academic studies often opt for complex transformations on metrics (e.g., using PCA to fuse correlated metrics), we observed that there is a higher preference for sensible, interpretable, and actionable metrics when applying defect modelling in an industrial setting [2, 5, 23, 32, 55, 62, 64, 81].

(Step-3) Data Preparation. One must prepare the data and compute metrics. Prior to model construction, one must analyze the distributions of the data [82] and the correlation of metrics [28] in order to understand the characteristics of the data.

(Step-4) Model Specification. One must define which metrics should be included or excluded in an analytical model. Also, one must decide the ordering of such metrics (e.g., `Model1: bugs~size +CC` vs `Model2: bugs~CC+size`).

(Step-5) Model Construction. One must construct an analytical model using, for example, statistical techniques (e.g., logistic or linear regression), or machine learning techniques (e.g., random forest) [15, 17, 74, 77].

(Step-6) Model Validation. Model validation techniques (e.g., out-of-sample bootstrap validation, repeated k-fold cross-validation) are commonly used to estimate the model performance on an unseen part of the dataset [76].

(Step-7) Model Interpretation. To test the hypotheses of interest and understand the relationship between a set of metrics and an outcome, one must examine the ranking of metrics using model interpretation techniques (e.g., many defect analytics studies use ANOVA Type-I [28], since it is the default built-in interpretation function for logistic regression (`glm`) models in `R`).

3 PITFALLS IN DEFECT MODELLING

In this section, we discuss several pitfalls that we faced over the years in the defect modelling process (see Figure 2). For each pitfall, we provide a detailed description of the pitfall along with a hands-on demonstration of the pitfall using a running example. We also discuss some pitfall avoidance strategies for each pitfall.

Table 1: Descriptions of the metrics of the running-example dataset [86].

Granularity	Metric	Description	File level
Method	`FOUT`	Number of method calls (fan out)	avg, max, total
	`MLOC`	Method lines of code	avg, max, total
	`NBD`	Nested block depth	avg, max, total
	`PAR`	Number of parameters	avg, max, total
	`CC`	McCabe's Cyclomatic Complexity	avg, max, total
Class	`NOF`	Number of fields	avg, max, total
	`NOM`	Number of methods	avg, max, total
	`NSF`	Number of static fields	avg, max, total
	`NSM`	Number of static methods	avg, max, total
File	`ACD`	Number of anonymous type declarations	value
	`NOI`	Number of interfaces	value
	`NOT`	Number of classes	value
	`TLOC`	Total lines of code	value

Running Example Dataset. To illustrate each pitfall, we use the `Eclipse 2.0` defect dataset as the subject of our analysis [86]. We select the `Eclipse 2.0` dataset since (1) Eclipse is a large-scale and active software project; and (2) it is widely used in a large number of prior studies [19]. The studied defect dataset has 6,740 modules at the file-level granularity with 31 software metrics and a defective ratio of 14.4%. Table 1 provides a description of the studied metrics.

3.1 Model Specification

> **Pitfall 1**—Testing hypotheses without including control metrics.

Description. Control metrics are confounding metrics that are not of interest even though they could affect the outcome of a model. However, studies are often not aware of the impact of not including control metrics when proposing a set of new metrics (or when studying the impact of a particular phenomenon). For example, to study if complexity metrics [10] are associated with project risk, one might use the number of reported bugs (`bugs`) to capture risk, and McCabe's Cyclomatic Complexity metrics to capture code complexity (`CC`), while controlling for code size (`TLOC`). We note that one needs to use control metrics to ensure that findings are not due to confounding factors (e.g., large modules are more likely to have more defects). Then, one must construct an analytical model with a model specification of `bugs~TLOC+CC`. One would then use an interpretation technique (e.g. ANOVA Type-I) to determine the ranking of metrics (i.e., whether the metrics of interest have a stronger relationship than the control metrics).

Demonstration. To illustrate the impact of not including control metrics, we examine the importance scores of the metric of interest (`CC`) with and without a control metric (`TLOC`). To simplify the demonstration, we select 3 metrics to be included in our model, i.e., `CC_max`, `PAR_max`, `FOUT_max`. We then include one control metric (`TLOC`) in the model. Then, we construct two different logistic regression models:

- (M1) one model without a control metric (i.e., `bugs~CC_max+PAR_max+FOUT_max`) and

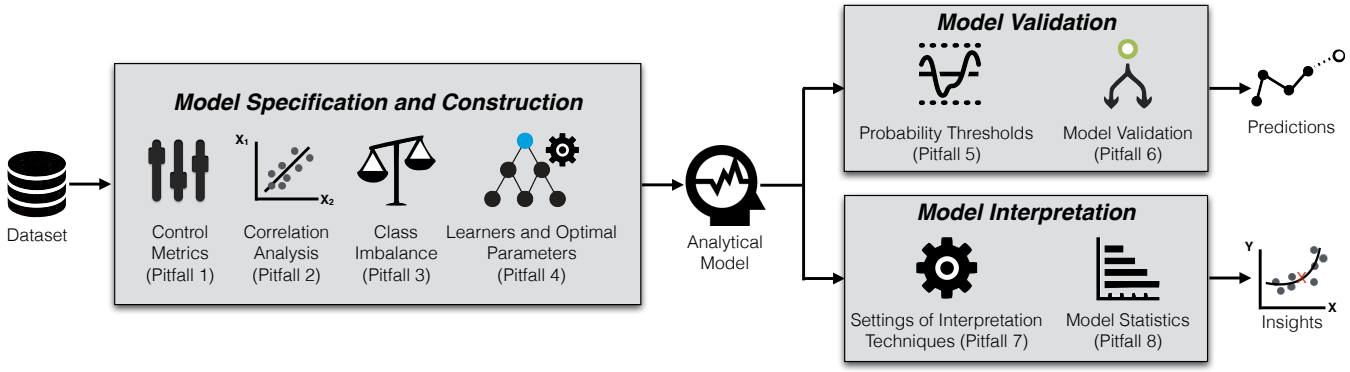


Figure 2: An overview of the eight pitfalls in analytical modelling process.

Table 2: The importance scores of a model with and a model without a control metric. The table shows that `CC_max` is the most important metric for the model without a control metric, yet `CC_max` is not the most important metric for the model with a control metric.

	M1	**M2**
Metrics	AUC=0.78	AUC=0.79
Control Metric		
TLOC	-	82%
Studied Metrics		
CC_max	76%	9%
PAR_max	17%	7%
FOUT_max	8%	2%

– (M2) another model with a control metric (i.e., bugs~TLOC+ CC_max+PAR_max+FOUT_max).

We then examine the importance scores of the two models using an ANOVA Type-I test.

Table 2 shows that CC_max is the most important metric for the model without a control metric, yet it is not the most important metric for the model with the control metric. Even though both models have a similar AUC performance, the importance score of CC_max substantially drops from 76% to 9% when including a control metric. This finding suggests that testing hypotheses without including control metrics could lead to different interpretation.

Avoidance Strategy. Control metrics must be included in defect models.

> **Pitfall 2**—Failure to deal with correlated metrics when interpreting models.

Description. Prior studies raise concerns that software metrics are often correlated [18, 26, 75, 82]. For example, Herraiz *et al.* [26], and Gil *et al.* [18] point out that code complexity (CC) is often correlated with code size (TLOC). Zhang *et al.* [82] point out that many metric aggregation schemes (e.g., averaging or summing of McCabe's Cyclomatic Complexity values at the function level to derive file-level metrics) often produce correlated metrics. Moreover,

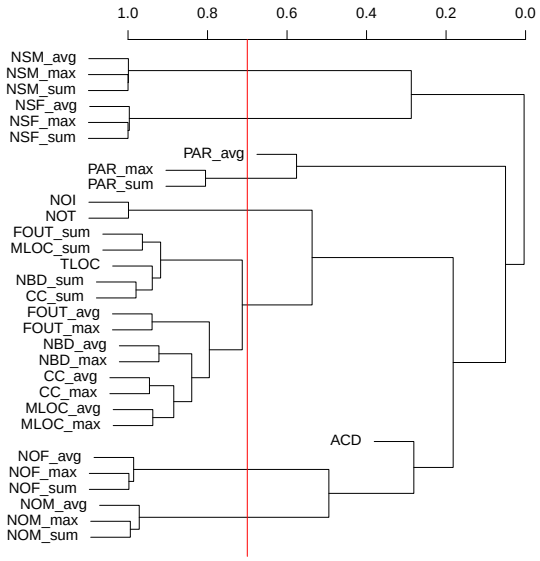


Figure 3: A hierarchical clustering view of the correlated metrics of the running-example dataset.

recent studies raise concerns that correlated metrics may impact the interpretation of defect models [28, 29, 44, 73, 75, 82]. Unfortunately, a literature survey of Shihab [61] shows that as much as 63% of published defect studies during 2000-2011 do not mitigate (e.g., remove) correlated metrics prior to constructing defect models.

Demonstration. To assess the impact of correlated metrics on the importance scores of metrics in a model, we analyze the importance scores of each metric in a model when correlated metrics are included in the model. We start with 4 metrics to be included in the models, i.e., CC_avg, CC_max, PAR_max, FOUT_max. To analyze the correlation between metrics, we apply the variable clustering analysis (VarClus). VarClus is a hierarchical clustering view of the correlation between metrics [60]. We use the implementation of the variable clustering analysis as provided by the varclus function of the Hmisc R package [21]. We use a Spearman correlation ($|\rho|$) threshold of 0.7 to identify correlated metrics (see Figure 3).

Table 3: The percentage of the importance scores for two different model specifications of a logistic regression model. Each bracketed value indicates the position of a metric in the model specification.

	M1 (AUC=0.78)		**M2** (AUC=0.78)	
Metrics	**Position**	**ANOVA**	**Position**	**ANOVA**
CC_max	[1]	74%	[2]	19%
CC_avg	[2]	2%	[1]	58%
PAR_max	[3]	16%	[3]	16%
FOUT_max	[4]	7%	[4]	7%

Our correlation analysis confirms that the average and the maximum McCabe's Cyclomatic Complexity metrics (i.e., CC_avg, and CC_max) are highly correlated. Hence, we construct two different logistic regression models with different model specifications:

- (M1) bugs~CC_max+CC_avg+PAR_max+FOUT_max and
- (M2) bugs~CC_avg+CC_max+PAR_max+FOUT_max

We then examine the importance scores of the two regression models using the commonly-used ANOVA Type-I interpretation technique.

As shown in Table 3, CC_max is the most important metric when it appears at the first position of the model (M1). However, CC_avg is the most important metric when it appears at the first position of the model (M2). Table 3 shows that the importance score of CC_max substantially drops from 74% to 19% when appearing at the second position of the model, indicating that the conclusions of many prior studies can be altered by simply re-ordering metrics in the model specification if correlated metrics are not properly mitigated.

Avoidance Strategy. Correlated metrics must be mitigated (e.g., removed) in models [28].

3.2 Model Construction

Pitfall 3—Class rebalancing techniques improve model performance.

Description. Defect models are often trained on *imbalanced datasets* (i.e., datasets where the proportion of defective and clean modules is not equally represented). However, imbalanced datasets are highly susceptible to producing inaccurate prediction models [25]. When training a defect model using an imbalanced dataset, traditional classification techniques often fail to accurately identify the minority class (e.g., defective modules). Thus, class rebalancing techniques are often applied to improve model performance. However, Turhan [80] points out that applying class rebalancing techniques may lead to bias in learned concepts (i.e., *concept drift*) — the resampled training dataset is not representative of the original dataset. Concept drift appears when the class distributions of training and testing datasets are different. Thus, class rebalancing techniques may impact the interpretation of defect models.

Demonstration. To assess the impact of class rebalancing techniques on the performance and interpretation, we analyze the AUC and F-measure performance, and the importance ranking of metrics of models when class rebalancing techniques are applied and when they are not applied. Since we find that correlated metrics have a large impact on model interpretation (see Pitfall 2), we first remove correlated metrics from the running-example dataset. To generate training datasets, we use the 100-repeated out-of-sample bootstrap validation technique [76] (the descriptions and rationale are provided in Pitfall 6). A model is trained using the bootstrap sample and tested using the rows that do not appear in the bootstrap sample. Prior to constructing a logistic regression model, we apply over-sampling and under-sampling techniques only on the training datasets, while the testing data is not rebalanced. The over-sampling technique randomly samples with replacement (i.e., replicating) the minority class (e.g., defective class) to be the same size as the majority class (e.g., clean class). The under-sampling technique randomly samples (i.e., reducing) the majority class (e.g., clean class) in order to reduce the number of majority modules to be the same number as the minority class (e.g., defective class). To apply the over-sampling technique, we use the implementation of the upSample function that is provided by the caret R package [40]. To apply the under-sampling technique, we use the implementation of the downSample function that is provided by the caret R package [40]. We then use an ANOVA Type-I test to estimate the importance scores of all metrics of each trained model [14]. Since the analysis is repeated 100 times, each metric will have several importance scores (i.e., one score for each of the repetitions). Hence, we apply the Scott-Knott Effect Size Difference (ESD) test (v2.0) [71, 77] to produce rankings of metrics of the models that class rebalancing techniques are applied and not applied.

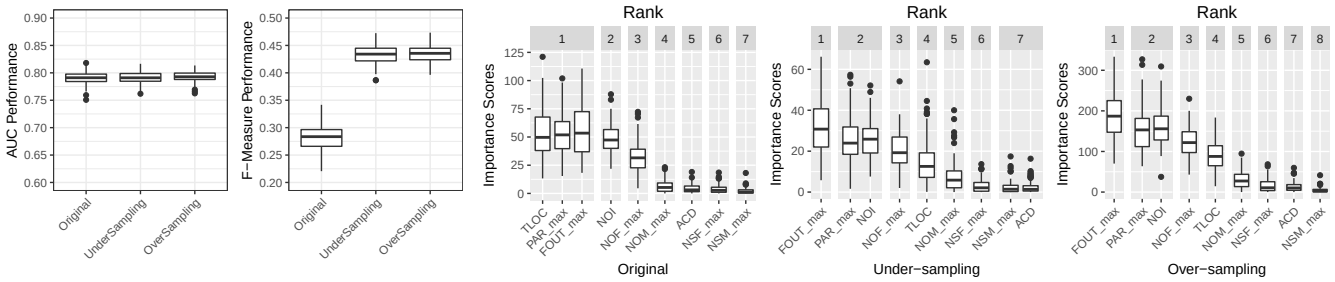
Figure 4a shows that class rebalancing techniques substantially improve F-measure (with a commonly-used probability threshold of 0.5), but have a minimal impact on AUC performance. Moreover, Figure 4b shows that class rebalancing techniques tend to shift the ranking of metric importances. For instance, we find that the second-most important metric (i.e., TLOC) appears at the fourth rank when class rebalancing techniques are applied.

Avoidance Strategy. This result and recent more rigorous analysis [72] suggest that class rebalancing techniques should be avoided when one plans to solely use an analytical model to guide decisions.

Pitfall 4—Not experimenting with different learners or using default parameter settings for learners.

Description. There exists a variety of learners that can be used to construct defect models (e.g., logistic regression, random forest, and neural network). Plenty of prior software engineering studies show that random forest is one of the best performing classification technique in multitude of software engineering data [41]. Furthermore, Tantithamthavorn *et al.* [74] and Fu *et al.* [15] point out that most software analytical models in literature often rely on the default settings of modelling toolkits. However, prior work [79] points out that the default parameter settings of random forest and naïve bayes are often suboptimal.

Hence, nowadays practitioners (and researchers) primarily use random forest learners, and in many cases fret over their use of optimal parameters. Instead, one must draw attention to well known observations in the machine learning community about the "no free lunch theorem", which emphasizes that no single modelling



(a) The AUC and F-Measure performance.

(b) The ScottKnott ESD rankings of metrics.

Figure 4: The AUC and F-Measure performance, and the ScottKnott ESD rankings of metrics of models with and without applying class rebalancing techniques.

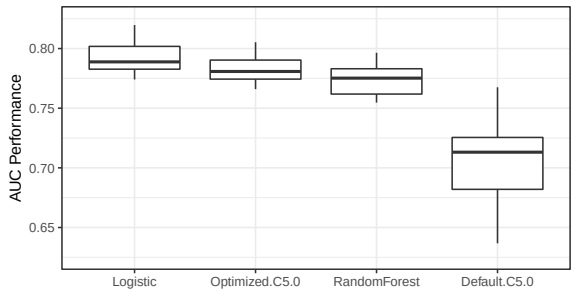


Figure 5: The AUC performance distributions of the logistic regression, optimized C5.0, random forest, and default C5.0 models.

technique is able to perform well in all settings and datasets. Indeed, practitioners should explore a multitude of learners for their studies—keeping in mind that the primary goal of prior studies of comparing learners (e.g., [17, 41]) is to highlight the commonly well performing learners instead of dictating the use of particular learners. Also, if possible, practitioners should consider optimizing the parameters of their learners—keeping in mind again that prior studies (e.g., [15, 74, 77]) show that automated parameter optimization (e.g., grid search, random search, and differential evolution) substantially improve model performance in a limited number of learners (e.g., C5.0, neural network learners).

Demonstration. To assess the impact of different learners and parameter settings, we examine the AUC performance of a defect model. Similar to prior analysis, we select the 5 most important metrics that are derived from Figure 4b to be included in the models, i.e., `FOUT_max`, `NOI`, `TLOC`, `PAR_max`, and `NOF_max`. Then, we construct our models using 3 commonly-used classification techniques (learners), i.e., logistic regression, random forest, and C5.0 (i.e., C4.5/J48). Since the C5.0 classification technique is sensitive to parameter settings, we construct the C5.0 classification technique with 2 parameter settings: a default setting and an optimal parameter setting. We use the 100-repeated out-of-sample bootstrap validation technique to estimate the AUC performance of the models.

In contrast to prior findings [41], Figure 5 shows that random forest is not always the top-performing learner [15, 17, 74]. Moreover,

Table 4: Confusion matrix for predicting defect-prone code entities.

	Actual	
Classified as	**Defective**	**Non-Defective**
Defective	TP	FP
Non-defective	FN	TN

Figure 5 shows that applying automated parameter optimization for the C5.0 model substantially improves its AUC performance [74].

Avoidance Strategy. Given the availability of automated parameter optimization in commonly-used research toolkits (e.g., Caret for R, MultiSearch for Weka, GridSearch for Scikit-learn), one should experiment with different learners and find optimal settings for parameter-sensitive analytical learners (e.g., C5.0 and neural network), instead of solely relying on observations from prior studies.

3.3 Model Validation

> **Pitfall 5**—Using threshold-dependent performance measures (e.g, F-measure) to measure the performance of a model.

Description. The performance of analytical models can be measured using a variety of threshold-dependent (e.g., precision, recall, and F-measure) and threshold-independent (e.g, Area Under the receiver operating characteristic Curve (AUC)) performance measures. Unlike threshold-independent measures, threshold-dependent measures often rely on a probability threshold (e.g., 0.5) for generating a confusion matrix (see Table 4). Precision measures the proportion of code entities that are classified as defective, which are actually defective ($\frac{TP}{TP+FP}$). Recall measures the proportion of actually defective code entities that were classified as such ($\frac{TP}{TP+FN}$). F-measure is the geometric mean of Precision and Recall ($\frac{2*precision*recall}{precision+recall}$). However, such threshold-dependent measures (e.g., Precision, Recall, and F-measure) can lead to different and conflicting conclusions.

Demonstration. To assess the impact of probability thresholds, we examine the F-measure performance of the four defect models

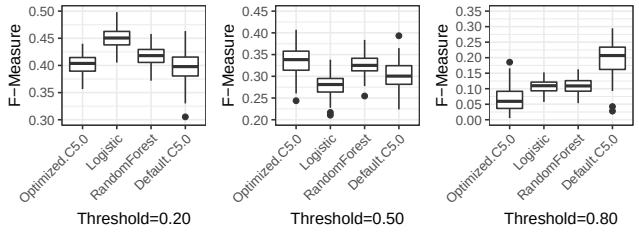


Figure 6: The F-measure performance of logistic regression, random forest, optimized C5.0, and default C5.0 models with different probability thresholds.

from Pitfall 4 using different probability thresholds. We use the 100-repeated out-of-sample bootstrap validation technique to estimate the F-measure performance of the models. We then experiment with different probability thresholds, i.e., 0.2, 0.5, and 0.8.

Figure 6 shows that the ranking of the learners is sensitive to the used probability threshold. We find that the optimized C5.0 model is the top-performing learner for a commonly-used probability threshold of 0.5. However, the logistic regression model is the top-performing learner for a probability threshold of 0.2, and the default C5.0 model is the top-performing learner for a probability threshold of 0.8.

Avoidance Strategy. Threshold-independent measures (e.g., AUC) should be used in lieu of threshold-dependent measures (e.g., F-measure).

Pitfall 6—Using 10-folds cross-validation for model validation.

Description. There exists a variety of model validation techniques that can be used to estimate the performance of a model. *Hold-out validation* randomly splits a dataset into training and testing corpora according to a given proportion (e.g., 30% holdout for testing). *Cross-validation* extends the idea of holdout validation by repeating the splitting process several times, through the random partitioning of the data into k folds of roughly equal size where each fold contains roughly the same proportions of the binary classes [16, 68]. We then use the training corpus to construct an analytical model, while the testing corpus is used to estimate the model performance. While such cross-validation techniques are commonly used in many software analytical modelling efforts, more powerful approaches like *bootstrap validation* that leverages aspects of statistical inference [12, 20, 24, 66] have been rarely explored in software engineering research. Unlike cross-validation techniques, a model is still trained using a bootstrap sample (i.e., a sample that is randomly drawn with replacement from a dataset), and the model is tested using the data points that do not appear in the bootstrap sample [11]. The process of resampling with replacement is repeated several times (e.g., 100 repetitions). The key intuition is that the relationship between a dataset and the theoretical population from which it is derived is asymptotically equivalent to the relationship between the bootstrap samples and the actual dataset.

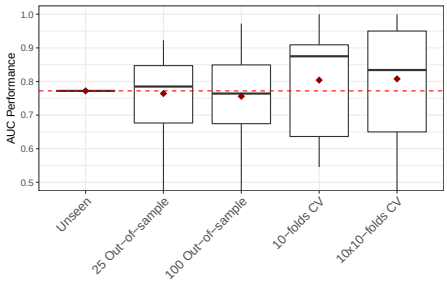


Figure 7: Model performance on unseen dataset versus the performance estimates that are produced by the 25-repeated out-of-sample bootstrap, 100-repeated out-of-sample bootstrap, 10-folds cross-validation, and 10x10-folds cross-validation techniques.

Demonstration. To assess the impact of model validation techniques, we examine the AUC performance estimates of defect models that are produced using the 25-repeated out-of-sample bootstrap, 100-repeated out-of-sample bootstrap, 10-folds cross-validation (most-commonly used technique nowadays) and 10x10-folds cross-validation. Our prior work [76] shows that cross-validation is susceptible to producing unstable performance estimates for datasets with low EPV values (i.e., EPV < 3). *Events Per Variable* (EPV) [53, 76] is 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 software metrics that are used to construct the model (i.e., the variables).

We split the running-example dataset into two subsamples, i.e., a subsampled dataset with an EPV of 3 for applying the model validation techniques and another dataset for measuring the performance of unseen dataset (i.e, ground-truth). To generate performance estimates, we apply the 4 different model validation techniques using the subsampled dataset. To generate the performance for the unseen dataset, we train a logistic regression model using the subsampled dataset, and compute the performance of the unseen dataset.

We find that the out-of-sample bootstrap validation technique produces the most accurate and most stable performance estimates for datasets with low EPV values [76]. Figure 7 shows that the unseen performance is 0.77, while the performance estimates are 0.76, 0.76, 0.80, and 0.81 for the 25-repeated out-of-sample bootstrap, 100-repeated out-of-sample bootstrap, 10-folds cross-validation, and 10x10-folds cross-validation techniques, respectively.

Avoidance Strategy. One should avoid using the 10-folds and 10x10 folds validation techniques (even though they have been used, and unfortunately continue to be used extensively in literature), but instead, opt to use the out-of-sample bootstrap validation technique, especially, for datasets with EPV < 3. Such datasets are commonly present in the literature [76]. It is worth noting that the computational cost of the 100-repeated out-of-sample bootstrap validation is the same as the 10x10-folds cross-validation (i.e., it took 25 seconds to build the 100 logistic regression models of Figure 5). Finally, the EPV values of datasets must be reported, especially when the used datasets are not shared.

3.4 Model Interpretation

Pitfall 7—Using ANOVA Type-I when interpreting a model.

Description. The Analysis of Variance (ANOVA) is a statistical test that examines the importance of two or more software metrics on the outcome (e.g., defect-proneness). While ANOVA Type-I is one of the most commonly-used interpretation techniques and the default interpretation technique for a regression model in R, other types of ANOVA (e.g., Type-II) have been rarely used in software engineering literature [28]. Moreover, the conclusions of analytical models must be insensitive to the model specification—i.e., conclusions should not rely on the ordering of metrics in a model specification. However, in some instances, conclusions may change by simply rearranging the ordering of metrics in a model's specification.

Demonstration. To assess the impact of the types of the ANOVA techniques and their sensitivity of model specification, we examine the importance scores of logistic regression models that are produced by ANOVA Type-I and Type-II. According to a correlation analysis of Figure 3, we construct two different logistic regression models using 9 non-correlated metrics with different model specifications:

- (M1) one model where the TLOC metric appears at the first position and
- (M2) another model where the TLOC metric appears at the last position.

Then, we apply an ANOVA Type-I test using the implementation of the anova function that is provided by the `stats` R package, and an ANOVA Type-II test using the implementation of the Anova function that is provided by the `car` R package. We then examine the importance scores of the TLOC metric from the two different models and different types of ANOVA.

Importance scores are sensitive to the ordering of metrics in a model specification when interpreting the model using ANOVA Type-I, even if correlated metrics are removed. Table 5 shows that TLOC is the highest ranked metric when TLOC is at the first position of the logistic regression (M1). Conversely, TLOC is among the lowest ranked metric when TLOC is at the last position of the logistic regression model (M2).

Different variants of the ANOVA test produce different importance scores. Table 5 shows that the NOM_max metric is the most important metric when the model is interpreted using the ANOVA Type-I test for the logistic regression model (M2). On the other hand, the TLOC metric is the most important metric when the model is interpreted using the ANOVA Type-II test for the logistic regression model (M2), and consistent with the ANOVA Type-II test for the logistic regression model (M1).

The different ranking of the most important metrics has to do with the different calculation of the ANOVA types. Type-I ANOVA computes the importance of each metric in a sequential order. Hence, Type-I attributes as much variance as it can to the first metric in a model's specification before attributing residual variance to the second metric in the model's specification. Thus, the importance (i.e., produced ranking) of metrics is dependent on the ordering of metrics in the model specification. On the other hand, the calculation of Type-II is equivalent to the calculation of Type-I where a metric under examination appears at the last position of the model. The intuition is that Type-II is evaluated after all of the other metrics have been accounted for.

Table 5: The percentage of the importance scores of two different model specifications for each of the two logistic regression models.

	M1		M2	
Metrics	**Type 1**	**Type 2**	**Type 1**	**Type 2**
TLOC	76%	22%	6%	22%
PAR_max	7%	21%	10%	21%
FOUT_max	8%	20%	28%	20%
NOI	5%	20%	6%	20%
NOF_max	4%	12%	5%	12%
NOM_max	0%	2%	33%	2%
ACD	0%	1%	5%	1%
NSF_max	0%	1%	5%	1%
NSM_max	0%	0%	2%	0%

Avoidance Strategy. This result and recent more rigorous analysis [28] suggest that one must not use ANOVA Type-I (the default in R), when testing hypotheses and developing analytical models. Instead, ANOVA Type-II, which is independent to the ordering of metrics in a model specification, should be used. ANOVA Type-III should be used when dealing with models with interaction terms. Furthermore, the exact specification of a defect model must be reported and the exact type of an ANOVA test should be indicated whenever an ANOVA analysis is performed.

Pitfall 8—Interpreting a model using the coefficients of its variables.

Description. Software metrics often originate from different distributions. For example, a total lines of code (TLOC) metric ranges from 0 to 50,000 TLOC, while a proportion of major developers (MDEV) ranges from 0 to 1. The big difference of such distributions often impact the coefficients of a regression model—e.g., the coefficients of TLOC could be 0.001, while the coefficients of MDEV could be 10. Interpreting coefficients from a regression model that is trained using data with different distribution ranges and units is likely to provide misleading conclusion—e.g., the MDEV metric is more important than TLOC, which is likely not true. Many novice modellers face this pitfall.

Demonstration. To assess the impact of interpreting logistic regression models using metric coefficients, we examine the coefficients and ANOVA Type-II importance scores of each of the metrics using a model that is trained using the original dataset versus a model that is trained using the scaled and centered dataset.

Irrespective of data transformation, the ANOVA Type-II test always produced consistent test statistics. Table 6 shows that the statistics of the ANOVA Type-II test are the same for models that are trained using original data and scaled-and-centered data. On

Table 6: The percentage of the importance scores of two model specifications of a logistic regression model:

	Model with original data		Model with scaled+centered data	
Metrics	Coefficients	ANOVA	Coefficients	ANOVA
TLOC	0%	22%	25%	22%
PAR_max	10%	21%	12%	21%
FOUT_max	1%	20%	14%	20%
NOI	-81%	20%	-19%	20%
NOF_max	-4%	12%	-16%	12%
NOM_max	1%	2%	6%	2%
ACD	-3%	1%	-3%	1%
NSF_max	0%	1%	4%	1%
NSM_max	0%	0%	-2%	0%

the other hand, coefficients are often impacted by data transformation. Table 6 shows that, when scaling and centering the data, the percentage importance of the TLOC metric shifts from 0% to 25%.

Avoidance Strategy. The magnitude of the coefficients of regression models should not be interpreted. Alternatively (and preferably), more advanced interpretation techniques should be used (e.g., an ANOVA Type-II/III test for logistic regression models, or a variable importance analysis for random forest models).

4 CHALLENGES ABOUT DEFECT MODELLING

In this section, we discuss some of the most notable challenges that we faced while working with industrial partners on modelling projects. Researchers must be careful about managing the expectations of practitioners by carefully discussing these challenges early on. Such careful discussions ensure that industrial partners have a broader view of findings and how to interpret them for their own peculiar settings.

We do note that access to clean data for performing defect analytics projects is a key challenge that often hinders progress in such projects [73]. However, we do not delve into the "limited access to good data" challenge in this paper due to space limitations. There are also many other challenges that are well documented elsewhere [4, 47, 70]. For this paper, we focus on challenges associated with defect modelling in particular and more specifically ones that we faced working in prior industrial engagements.

> **Challenge 1**—Using defect models to guide operational decisions instead of simply predicting defects.

A common misunderstanding of studies that have a defect model is that predicting defects is the main goal of such studies. Such a misunderstanding often leads practitioners (and researchers in some cases) to doubt the value of many published studies with defect models, since practitioners might feel that the models are either predicting well-known modules to be defective (i.e., providing little practical value), or that the produced predictions are at a very high level of granularity (e.g., file- or subsystem- vs method-level).

Instead, apart from being used for prediction, defect models are especially essential for carefully examining phenomenas and hypotheses of interest in an effort to derive empirical theories and/or to guide operational decisions surrounding software quality within large corporations [3, 5, 22, 43, 45, 78]. The following are examples of such studies: Bird *et al.* [5] explore the risks of distributed development, Bettenburg *et al.* [3] examine the impact of social interaction on software quality, and recent work [43, 45, 78] examines best practices for code reviewing. While such studies examine several metrics relative to defect proneness, the goal of such studies is to identify actionable insights in order to predict bugs. Hence, the use of statistical and machine learning models (e.g., logistic regression, or random forest), that are interpretable, is more desirable over black-box machine learning models like neural networks and deep learners.

It is also worth noting that modelling defect-proneness or counts might not always be the target of such models. In particular, the goal of models are often centred around quality with the definition of what constitutes quality varying considerably across organizations and even across products and teams within a single organization. For instance, in one engagement [62], the concept of change risk emerged as a very important concept that is worth modelling instead of simply modelling defect-proneness—e.g., risk was defined as any change that might lead to a schedule slippage.

> **Challenge 2**—Replicating published studies in an industrial setting.

While there is a large tendency in academic publications to demonstrate that findings are generalizable, all too often practitioners are not too concerned about the generalization of a particular finding [6, 7]. Instead, a major concern for practitioners is their ability to easily adopt published approaches to their own context (e.g., by replicating a particular study on their own project) [64].

Hence, the unavailability of well-documented modelling scripts from published studies is often one of the biggest obstacle that practitioners face when attempting to adopt published studies into their own industrial setting. Such an obstacle is a much larger concern that the availability of the datasets or the generality of reported findings. Hence, we believe that efforts similar to the PROMISE repository (which provides access to datasets) are needed for the archival of scripts that are used in research studies. It is worth noting that while data availability is not possible in many instances, due to, for example NDA agreements, the restricted availability of modelling scripts is hard to justify. Yet, nowadays, the scripts are rarely offered.

5 CONCLUSIONS

In this paper, we demonstrate a collection of pitfalls in defect modelling and discuss challenges about defect modelling. We wish to emphasize that our work is not exhaustive. Instead, it simply documents the most common pitfalls and challenges that we faced as part of several industrial engagements over the past decade.

Most importantly, we would like to emphasize that we do not seek to claim the generality of our observations. Instead, the key message of this paper is that there are some settings where such

pitfalls can impact the results of defect models (as we demonstrated using our simple running-example). Hence, it is of great importance that practitioners and researchers are well aware of such pitfalls and are on the lookout to avoid them in their work. We do provide our R script online[1] to assist practitioners and researchers that are interested in studying and avoiding such pitfalls.

We wrap up this paper by highlighting three observations from our industrial engagements in defect analytics projects:

- It is essential to manage the expectation of practitioners early on about the benefits of the analytical modelling of defect (e.g., its inability to demonstrate causal relationships).
- The absence of high-quality data to develop analytical models is too often the biggest challenge in starting a defect modelling project in an industrial setting.
- The unavailability of well-documented modelling scripts from most published studies is quite often the biggest obstacle that practitioners face nowadays when attempting to adopt published studies into their own industrial setting. Such an obstacle is a much larger concern than the generality of reported findings or even the availability of the datasets. Hence, we believe that efforts similar to the PROMISE repository (which provides access to datasets) are needed for the archival of scripts that are used in published studies.

ACKNOWLEDGEMENT

We thank Bram Adams, Cor-Paul Bezemer, Daniel M. German, Gustavo Ansaldi Oliva, Emad Shihab, and Thomas Zimmermann for their constructive feedback on earlier versions of this paper.

REFERENCES

[1] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 1–10.
[2] Ayse Bener, Ayse Tosun Misirli, Bora Caglayan, Ekrem Kocaguneli, and Gul Calikli. 2015. Lessons Learned from Software Analytics In Practice. *The art and science of analyzing software data, 1st edn. Elsevier, Waltham* (2015), 453–489.
[3] Nicolas Bettenburg and Ahmed E Hassan. 2013. Studying the impact of social interactions on software quality. *Empirical Software Engineering (EMSE)* 18, 2 (2013), 375–431.
[4] Christian Bird, Tim Menzies, and Thomas Zimmermann. 2015. *The Art and Science of Analyzing Software Data*. Elsevier.
[5] Christian Bird, Brendan Murphy, and Harald Gall. 2011. Don't Touch My Code ! Examining the Effects of Ownership on Software Quality. In *Proceedings of the European Conference on Foundations of Software Engineering (ESEC/FSE)*. 4–14.
[6] Lionel Briand. 2012. Embracing the engineering side of software engineering. *IEEE software* 29, 4 (2012), 96–96.
[7] Lionel Briand, Domenico Bianculli, Shiva Nejati, Fabrizio Pastore, and Mehrdad Sabetzadeh. 2017. The Case for Context-Driven Software Engineering Research: Generalizability Is Overrated. *IEEE Software* 34, 5 (2017), 72–75.
[8] Bora Caglayan, Burak Turhan, Ayse Bener, Mayy Habayeb, Andriy Miransky, and Enzo Cialini. 2015. Merits of organizational metrics in defect prediction: an industrial replication. In *Proceedings of the International Conference on Software Engineering (ICSE)*, Vol. 2. 89–98.
[9] Jeffrey C Carver, Natalia Juristo, Maria Teresa Baldassarre, and Sira Vegas. 2014. Replications of software engineering experiments. *Empirical Software Engineering (EMSE)* 19, 2 (2014), 267–276.
[10] Shyam R Chidamber and Chris F Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering (TSE)* 20, 6 (1994), 476–493.
[11] Bradley Efron. 1983. Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation. *J. Amer. Statist. Assoc.* 78, 382 (1983), 316–331.
[12] Bradley Efron and Robert J. Tibshirani. 1993. *An Introduction to the Bootstrap*. Springer US, Boston, MA.
[13] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. 2017. The Evolution of Continuous Experimentation in Software Product Development. In *International Conference on Software Engineering (ICSE)*. 770–780.
[14] RA Fisher. 1925. Intraclass correlations and the analysis of variance. *Statistical Methods for Research Workers* (1925), 187–210.
[15] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76, Supplement C (2016), 135 – 146.
[16] Seymour Geisser. 1975. Sample Reuse Method The Predictive with Applications. *J. Amer. Statist. Assoc.* 70, 350 (1975), 320–328.
[17] Baljinder Ghotra, Shane McIntosh, and Ahmed E. Hassan. 2015. Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 789–800.
[18] Yossi Gil and Gal Lalouche. 2017. On the correlation between size and metric validity. *Empirical Software Engineering (EMSE)* (2017), In Press.
[19] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2012. A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering (TSE)* 38, 6 (nov 2012), 1276–1304.
[20] Frank E. Harrell Jr. 2002. *Regression Modeling Strategies* (1st ed.).
[21] Frank E. Harrell Jr. 2013. Hmisc: Harrell miscellaneous. R package version 3.12-2. *Software available at URL: http://cran.r-project.org/web/packages/Hmisc* (2013).
[22] Ahmed E Hassan. 2009. Predicting faults using the complexity of code changes. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 78–88.
[23] Ahmed E. Hassan and Richard C. Holt. 2005. The Top Ten List: Dynamic Fault Prediction. In *Proceedings of the International Conference on Software Maintenance (ICSM)*. 263–272.
[24] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The elements of statistical learning*.
[25] Haibo He and Edwardo A Garcia. 2009. Learning from Imbalanced Data. *Transactions on Knowledge and Data Engineering (TKDE)* 21, 9 (2009), 1263–1284.
[26] Israel Herraiz, Daniel M German, and Ahmed E Hassan. 2011. On the Distribution of Source Code File Sizes.. In *Proceedings of the International Conference on Software Technologies (ICSOFT)*. 5–14.
[27] Andreas Jedlitschka and Dietmar Pfahl. 2005. Reporting guidelines for controlled experiments in software engineering. In *Proceedings of the International Symposium on Empirical Software Engineering (ESEM)*. 10.
[28] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Ahmed E. Hassan. 2018. The Impact of Correlated Metrics on Defect Models. (2018). arXiv:arXiv:1801.10271
[29] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, Akinori Ihara, and Kenichi Matsumoto. 2016. A study of redundant metrics in defect prediction datasets. In *Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on*. IEEE, 51–52.
[30] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A Large-Scale Empirical Study of Just-In-Time Quality Assurance. *IEEE Transactions on Software Engineering (TSE)* 39, 6 (2013), 757–773.
[31] Katja Kevic, Brendan Murphy, Laurie Williams, and Jennifer Beckmann. 2017. Characterizing experimentation in continuous deployment: a case study on bing. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 123–132.
[32] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The emerging role of data scientists on software development teams. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 96–107.
[33] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2017. Data Scientists in Software Teams: State of the Art and Challenges. *IEEE Transactions on Software Engineering (TSE)* PP (2017), 1–1. Issue 99.
[34] Barbara Kitchenham. 2004. Procedures for performing systematic reviews. *Keele, UK, Keele University* 33, 2004 (2004), 1–26.
[35] Barbara Kitchenham, Hiyam Al-Khilidar, Muhammed Ali Babar, Mike Berry, Karl Cox, Jacky Keung, Felicia Kurniawati, Mark Staples, He Zhang, and Liming Zhu. 2008. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering (EMSE)* 13, 1 (2008), 97–121.
[36] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report.
[37] Barbara Kitchenham and Emilia Mendes. 2009. Why comparative effort prediction studies may be invalid. In *Proceedings of the International Conference on Predictor Models in Software Engineering (PROMISE)*. 4.
[38] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering (TSE)* 28, 8 (2002), 721–734.
[39] Andrew J Ko, Thomas D Latoza, and Margaret M Burnett. 2015. A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering (EMSE)* 20, 1 (2015), 110–141.
[40] Max Kuhn. 2015. caret: Classification and Regression Training. http://CRAN.R-project.org/package=caret. (2015).

[1] https://sailresearch.github.io/analytics-pitfalls/index.html

[41] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. 2008. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering (TSE)* 34, 4 (2008), 485–496.

[42] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E James Whitehead Jr. 2013. Does bug prediction support human developers? findings from a google case study. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 372–381.

[43] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2014. The Impact of Code Review Coverage and Code Review Participation on Software Quality. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*. 192–201.

[44] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2016. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[45] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2016. An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Software Engineering* 21, 5 (2016), 2146–2189.

[46] Andrew Meneely, Pete Rotella, and Laurie Williams. 2011. Does adding manpower also affect quality?: an empirical, longitudinal analysis. In *Proceedings of the European Conference on Foundations of Software Engineering (ESEC/FSE)*. 81–90.

[47] Tim Menzies, Laurie Williams, and Thomas Zimmermann. 2016. *Perspectives on Data Science for Software Engineering*. Morgan Kaufmann.

[48] Audris Mockus and David M Weiss. 2000. Predicting Risk of Software Changes. *Bell Labs Technical Journal* 5, 6 (2000), 169–180.

[49] Nachiappan Nagappan and Thomas Ball. 2007. Using software dependencies and churn metrics to predict field failures: An empirical case study. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 364–373.

[50] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. 2006. Mining metrics to predict component failures. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 452–461.

[51] Nachiappan Nagappan, Andreas Zeller, Thomas Zimmermann, Kim Herzig, and Brendan Murphy. 2010. Change Bursts as Defect Predictors. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*. 309–318.

[52] Thomas J Ostrand, Elaine J Weyuker, and Robert M Bell. 2005. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering (TSE)* 31, 4 (apr 2005), 340–355.

[53] Peter Peduzzi, John Concato, Elizabeth Kemper, Theodore R Holford, and Alvan R Feinstein. 1996. A Simulation Study of the Number of Events per Variable in Logistic Regression Analysis. *Journal of Clinical Epidemiology* 49, 12 (1996), 1373–1379.

[54] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic Mapping Studies in Software Engineering. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE)*. 68–77.

[55] Romain Robbes, René Vidal, and María Cecilia Bastarrica. 2013. Are software analytics efforts worthwhile for small companies? The case of Amisoft. *IEEE software* 30, 5 (2013), 46–53.

[56] Pete Rotella and Sunita Chulani. 2011. Implementing quality metrics and goals at the corporate level. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*. 113–122.

[57] Pete Rotella, Sunita Chulani, and Devesh Goyal. 2015. Predicting field reliability. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (FSE)*. 986–989.

[58] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering (EMSE)* 14, 2 (2009), 131.

[59] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. 2012. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons.

[60] WS Sarle. 1990. The VARCLUS Procedure. SAS/STAT User's Guide. SAS Institute. *Inc., Cary, NC, USA* (1990).

[61] Emad Shihab. 2012. *An Exploration of Challenges Limiting Pragmatic Software Defect Prediction*. Ph.D. Dissertation. Queen's University.

[62] Emad Shihab, Ahmed E. Hassan, Bram Adams, and Zhen Ming Jiang. 2012. An Industrial Study on the Risk of Software Changes. In *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE)*. 62.

[63] Emad Shihab, Audris Mockus, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. 2011. High-impact defects: a study of breakage and surprise defects. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 300–310.

[64] Junji Shimagaki, Yasutaka Kamei, Shane McIntosh, Ahmed E Hassan, and Naoyasu Ubayashi. 2016. A Study of the Quality-Impacting Practices of Modern Code Review at Sony Mobile. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 212–221.

[65] Forrest J Shull, Jeffrey C Carver, Sira Vegas, and Natalia Juristo. 2008. The role of replications in empirical software engineering. *Empirical Software Engineering (EMSE)* 13, 2 (2008), 211–218.

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

[67] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 120–131.

[68] M Stone. 1974. Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society* 36, 2 (1974), 111–147.

[69] Ming Tan, Lin Tan, Sashank Dara, and Caleb Mayeux. 2015. Online Defect Prediction for Imbalanced Data. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 99–108.

[70] Chakkrit Tantithamthavorn. 2016. Towards a Better Understanding of the Impact of Experimental Components on Defect Prediction Modelling. In *In Proceedings of the International Conference on Software Engineering (ICSE)*. 867–870.

[71] Chakkrit Tantithamthavorn. 2017. ScottKnottESD: The Scott-Knott Effect Size Difference (ESD) Test. https://cran.r-project.org/web/packages/ScottKnottESD/. (2017).

[72] Chakkrit Tantithamthavorn, Ahmed E. Hassan, and Kenichi Matsumoto. 2018. The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models. (2018). arXiv:arXiv:1801.10269

[73] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, Akinori Ihara, and Kenichi Matsumoto. 2015. The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 812–823.

[74] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated Parameter Optimization of Classification Techniques for Defect Prediction Models. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 321–322.

[75] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. Comments on "Researcher Bias: The Use of Machine Learning in Software Defect Prediction". *IEEE Transactions on Software Engineering (TSE)* 42, 11 (2016), 1092–1094.

[76] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2017. An Empirical Comparison of Model Validation Techniques for Defect Prediction Models. *IEEE Transactions on Software Engineering (TSE)* 43, 1 (2017), 1–18.

[77] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2018. The Impact of Automated Parameter Optimization on Defect Prediction Models. *IEEE Transactions on Software Engineering (TSE)* (2018).

[78] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. 2016. Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 1039–1050.

[79] Ayse Tosun and Ayse Bener. 2009. Reducing false alarms in software defect prediction by decision threshold optimization. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 477–480.

[80] Burak Turhan. 2011. On the dataset shift problem in software engineering prediction models. *Empirical Software Engineering (EMSE)* 17, 1-2 (2011), 62–74.

[81] Dongmei Zhang, Yingnong Dang, Jian-Guang Lou, Shi Han, Haidong Zhang, and Tao Xie. 2011. Software Analytics as a Learning Case in Practice: Approaches and Experiences. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering (MALETS)*. 55–58.

[82] Feng Zhang, Ahmed E Hassan, Shane McIntosh, and Ying Zou. 2017. The Use of Summation to Aggregate Software Metrics Hinders the Performance of Defect Prediction Models. *IEEE Transactions on Software Engineering (TSE)* 43, 5 (2017), 476–491.

[83] Thomas Zimmermann and Nachiappan Nagappan. 2008. Predicting defects using network analysis on dependency graphs. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 531–540.

[84] Thomas Zimmermann and Nachiappan Nagappan. 2009. Predicting defects with program dependencies. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 435–438.

[85] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. 2009. Cross-project Defect Prediction. In *Proceedings of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 91–100.

[86] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. 2007. Predicting Defects for Eclipse. In *Proceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE)*. 9–19.