

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/335429044>

Mining Software Defects: Should We Consider Affected Releases?

Conference Paper · May 2019

DOI: 10.1109/ICSE.2019.00075

CITATIONS

148

READS

799

4 authors, including:



Patanamon Thongtanunam

University of Melbourne

80 PUBLICATIONS 1,892 CITATIONS

[SEE PROFILE](#)



Chakkrit Tantithamthavorn

Monash University (Australia)

150 PUBLICATIONS 6,312 CITATIONS

[SEE PROFILE](#)

Mining Software Defects: Should We Consider Affected Releases?

Suraj Yatish[✉], Jirayus Jiarpakdee[✉], Patanamon Thongtanunam[✉], Chakkrit Tantithamthavorn[✉]
✉ The University of Adelaide. ✉ Monash University. ✉ The University of Melbourne.

Email: suraj.yatish@adelaide.edu.au, jirayus.jiarpakdee@monash.edu,
patanamon.thongtanunam@unimelb.edu.au, chakkrit.tantithamthavorn@monash.edu

Abstract—With the rise of the Mining Software Repositories (MSR) field, defect datasets extracted from software repositories play a foundational role in many empirical studies related to software quality. At the core of defect data preparation is the identification of post-release defects. Prior studies leverage many heuristics (e.g., keywords and issue IDs) to identify post-release defects. However, such the heuristic approach is based on several assumptions, which pose common threats to the validity of many studies. In this paper, we set out to investigate the nature of the difference of defect datasets generated by the heuristic approach and the realistic approach that leverages the earliest affected release that is realistically estimated by a software development team for a given defect. In addition, we investigate the impact of defect identification approaches on the predictive accuracy and the ranking of defective modules that are produced by defect models. Through a case study of defect datasets of 32 releases, we find that the heuristic approach has a large impact on both defect count datasets and binary defect datasets. Surprisingly, we find that the heuristic approach has a minimal impact on defect count models, suggesting that future work should not be too concerned about defect count models that are constructed using heuristic defect datasets. On the other hand, using defect datasets generated by the realistic approach lead to an improvement in the predictive accuracy of defect classification models.

Index Terms—Mining Software Repositories, Empirical Software Engineering, Software Quality, Defect Prediction Models.

I. INTRODUCTION

Defect datasets play a foundational role in many empirical software engineering studies. The first line of studies is the development of empirical-grounded theories that are related to software quality. For example, prior studies statistically analyze defect datasets to investigate the impact that code attributes [34, 35, 37, 38, 40], object-oriented design [5], development activities [50, 55], code smells [26, 27, 46], continuous integration practices [69, 70], code review practices [33, 52, 66, 67], and human factors [7, 8, 12, 39, 49] have on software quality. The second line of studies is the development of defect models that leverage the knowledge of pitfalls that lead to software defects in the past. For example, prior studies develop defect prediction models that learn historical defect characteristics in order to predict the most defective modules [10, 19, 22, 36, 71, 73].

At the core of defect data preparation is the identification of post-release defects (i.e., modules that are changed to address a defect report after software is released). The widely-used *heuristic approach* to identify post-release defects is based on the extraction of specific keywords and issue IDs in commit

logs in the Version Control System (VCS) using regular expression [15, 17, 25, 56, 75] within a specific post-release window period (e.g., 6 months).

However, such the heuristic approach is based on several assumptions [4, 6, 42, 43, 61, 72], which pose common threats to the validity of many studies. First, the heuristic approach assumes that all defect-fixing commits within the specific post-release window period only affect the release of interest. However, it is likely that some defect reports that are addressed within the specific post-release window period did not actually affect the release of interest. Second, the heuristic approach assumes that all defect-fixing commits that affect the release of interest only occur within the specific post-release window period. Yet, some defect reports that actually affect the release of interest may be addressed after the specific window period.

Recently, da Costa *et al.* [14] suggest that the affected-release field in an Issue Tracking System (ITS) should be considered when identifying defect-introducing commits. To generate defect datasets in a more realistic scenario, one should use *the realistic approach*, i.e., the use of the earliest affected release that is realistically estimated by a software development team for a given defect. However, the affected-release field is rarely considered when identifying post-release defects of a software module. Yet, little is known about how much impact do these approaches can have on defect datasets and defect models that many studies rely upon.

In this paper, we set out to investigate (1) the nature of the difference of defect datasets that are generated by the heuristic and realistic approaches; and its impact on (2) the predictive accuracy and (3) the ranking of defective modules that are produced by both defect count models and defect classification models. First, we generate 4 defect datasets based on 2 types of defect datasets (i.e., defect count datasets and binary defect datasets) using 2 defect identification approaches (i.e., the heuristic approach and the realistic approach). Then, we construct defect models using defect datasets generated by the heuristic approach and the realistic approach. We focus on 2 types of defect models, i.e., defect count and defect classification models. Finally, we evaluate the impact of the heuristic approach with respect to the predictive accuracy (e.g., AUC) and the ranking of defective modules (e.g., P@20%LOC) that are produced by defect models. Through a case study of 32 releases that span across 9 large-scale open-source software systems, we address the following research questions:

(RQ1) How do heuristic defect datasets and realistic defect datasets differ?

The heuristic approach impacts the defect counts of 89% of defective modules in defect count datasets and mislabels 55% of defective modules in binary defect datasets, raising concerns related to the validity and the reliability of defect models that are constructed using heuristic defect datasets.

(RQ2) How do defect identification approaches impact the predictive accuracy of defect models?

Surprisingly, there is no statistically significant difference in the predictive accuracy between defect count models that are constructed using heuristic and realistic defect datasets. On the other hand, defect classification models that are constructed using realistic defect datasets lead an increase in the predictive accuracy that is rather modest.

(RQ3) How do defect identification approaches impact the ranking of defective modules produced by defect models?

The heuristic approach has a negligible to small impact on the ranking of defective modules produced by both defect count models and defect classification models.

Our results suggest that future work should not be too concerned about defect count models that are constructed using heuristic defect datasets. However, using defect datasets generated by the realistic approach lead to an improvement in the predictive accuracy of defect classification models.

Contributions. To the best of our knowledge, the main contributions of this paper are as follows:

- (1) An introduction to the realistic approach of generating post-release defect datasets using the earliest affected release that is realistically estimated by a software development team for a given defect while relaxing the assumptions of using a post-release window period (Section II-B).
- (2) An evaluation framework of defect identification approaches (Section IV).
- (3) An investigation of the nature of the difference of defect datasets that are generated by the heuristic approach and the realistic approach (RQ1).
- (4) An investigation of the impact of post-release defect identification approaches on the predictive accuracy and the ranking of defective modules that are produced by defect models (RQ2 and RQ3).
- (5) A new collection of highly-curated benchmark defect datasets of 32 releases that span across 9 open-source software systems using the heuristic and realistic approaches [60] and a replication package at Zenodo [74].

Paper organization. Section II situates this paper with respect to the related work. Section III describes our selection criteria of the studied systems. Section IV elaborates on the design of our case study, while Section V presents the results with respect to our three research questions. Section VI discusses the threats to the validity of our study. Finally, Section VII draws conclusions.

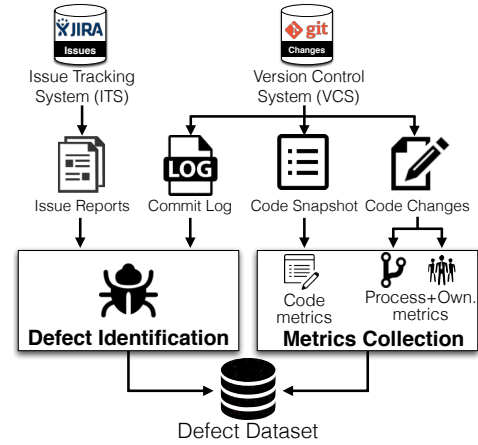


Fig. 1: An overview of defect data preparation steps.

II. RELATED WORK AND RESEARCH QUESTIONS

Defect data preparation involves several key steps. Figure 1 provides an overview of defect data preparation steps. First, one must extract issue reports, which describe defects, feature requests, or general maintenance tasks from an Issue Tracking System (ITS, e.g., JIRA). Second, one must extract commit logs, code snapshots, and historical code changes that are recorded in a Version Control System (VCS, e.g., Git). Third, one must extract software metrics of each module (e.g., size, and complexity) from the VCS. To extract such software metrics, one must focus on development activities that occur prior to a release of interest that corresponds to the release branch of a studied system to ensure that activities that we study correspond to the development and maintenance of the official software releases. Finally, one must identify and label modules as defective if they have been affected by code changes that address issue reports that were classified as defects after the software is released (i.e., post-release defects). On the other hand, modules that are not changed to address any issue reports are labelled as clean.

Prior studies proposed several approaches to identify post-release defects, which we describe below.

A. The heuristic approach

Post-release defects are defined as modules that are changed to address a defect report within a post-release window period (e.g., 6 months) [15, 17, 25, 56, 75]. The commonly-used approach is to infer the relationship between the defect-fixing commits that are referred in commit logs and their associated defect reports. To identify defect-fixing commits, Fischer *et al.* [17] are among the first to introduce a heuristic approach by using regular expression to search for specific keywords (e.g., `fix(e[ds])?`, `bugs?`, `defects?`) and issue IDs (e.g., `[0-9]+`) in commit logs after the release of interest.

To illustration, we use Figure 2 to provide an example of defect identification of the heuristic approach for the release `v1.0`. First, the heuristic approach will apply a collection of regular expression patterns to search through the commit logs

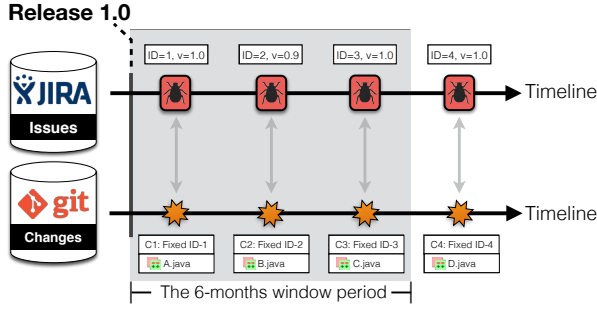



Fig. 2: An illustrative example of defect identification approaches, where ID indicates an issue report ID, v indicates affected release(s), C indicates a commit hash,  indicates changed file(s), and the grey area indicates the 6-month period after a release of interest (i.e., Release 1.0).

to find defect-fixing commits within a specific window period (i.e., 6 months). According to Figure 2, the heuristic approach will label commits C1, C2, and C3 as defect-fixing commits, since their commit logs match with the collection of regular expression patterns that contain specific keywords and issue IDs. Then, the heuristic approach will label modules A.java, B.java, and C.java as defective modules. The remaining module (i.e., D.java) will be labelled as clean.

Unfortunately, such the heuristic approach has several limitations. First, the heuristic approach assumes that all defect-fixing commits that are identified within the post-release window period of the release of interest (i.e., $v1.0$) are the defect-fixing commits that affect the release of interest. However, Figure 2 shows that the commit C2 was changed to address the issue report ID-2 where the release $v0.9$ was affected. Thus, the module B.java that was labelled as defective by the heuristic approach is actually clean. Second, the heuristic approach assumes that all defect-fixing commits that affect the release of interest (i.e., $v1.0$) only occur within the post-release window period of the release of interest. Figure 2 shows that the commit C4 was made to address the issue report ID-4 where the release $v1.0$ was affected, however the commit C4 was made outside the post-release window period. Thus, the module D.java that was labelled as clean by the heuristic approach is actually defective.

B. The realistic approach

Recently, da Costa *et al.* [14] suggest that the affected-release field in an ITS should be considered when identifying defect-introducing changes. To address the limitations of the heuristic approach, one should use the realistic approach, i.e., the use of the earliest affected release that is realistically estimated by a software development team for a given defect to generate post-release defects. The affected-release field in an ITS allows developers to record the actual releases of the system that are affected by a given defect. To identify defect-fixing commits, the realistic approach retrieves defect reports that affect a release of interest in the ITS. Since modern ITSs, like JIRA, provide traceable links between defect reports and

code commits in VCS, the realistic approach can identify modules that are changed to address those defect reports. Then, the modules that are changed to address those defect reports are then identified as defective, otherwise clean.

To illustration, Figure 2 shows that there are three issue reports (i.e., ID-1, ID-3, and ID-4) that affect the studied release $v1.0$. Then, commits C1, C3, and C4 which are linked to those three issue reports are identified as defect-fixing commits by the realistic approach. Finally, modules A.java, C.java, and D.java which are impacted by commits C1, C3, and C4 are labelled as defective modules.

C. Challenges of mining software defects

Software repositories are often noisy (e.g., incorrect, missing information) [9, 32]. For example, Aranda and Venolia [2] point out that ITS and VCS repositories are noisy sources of data. Thus, the usage of such noisy data may pose a critical threat to the validity in many empirical studies.

Prior studies show that noise may creep into defect datasets and impact defect models if care is not taken when collecting data from software repositories. The first line of studies includes an investigation of the impact of noises that are generated by issue report misclassification. For example, Antoniol *et al.* [1] and Herzig *et al.* [21] find that many issue reports are misclassified. However, recent work shows that issue report mislabelling rarely impacts the precision of defect models or the interpretation of the most important software metrics [61]. The second line of studies includes studies an investigation of the impact of noises that are generated by defect identification approaches. For example, Bachmann *et al.* [4] find that the defect reports are not recorded in the commit logs and thus are not visible to the automated linking tools used to extract defect datasets, which generate many missing links between code changes and defect reports. Bird *et al.* [6] point out that more experienced developers are more likely to explicitly link issue reports to the corresponding code changes. Nguyen *et al.* [43] show that such noises also exist in commercial datasets. Furthermore, Rahman *et al.* [50] argue that the size of the defect datasets matters more than the dataset quality. To address this challenge, prior studies [42, 72] introduce the textual similarity approaches between issue reports and commit messages to automatically recover the missing links between issue reports to the corresponding code changes.

D. Research Questions

Recent work [14] suggests that the affected-release field in an ITS should be considered when identifying defect-introducing changes. Our illustrative example in Figure 2 shows that even if issue reports are addressed within a period after the software is recently released, they do not necessary affect that software release. Moreover, our illustrative example shows that some issue reports that realistically affect the studied release are addressed after the specific window period. Thus, the heuristic approach may generate noise into defect datasets when compared to the realistic approach. Yet, little is known about the nature of the difference of defect datasets

that are generated by the heuristic and realistic approaches. Thus, we formulate the following research question:

(RQ1) How do heuristic defect datasets and realistic defect datasets differ?

Prior studies have shown that the missing links between issue reports and code changes often have a large negative impact on the performance of defect models. For example, Kim *et al.* [28] demonstrate that random noise that is generated by missing links has a large negative impact on the predictive accuracy of defect models. On the other hand, Herzig *et al.* [21] and Kovalenko *et al.* [29] demonstrate that noise that is generated by issue report mislabelling and branches has little impact on the predictive accuracy of defect models, respectively. However, prior work only focuses on the impact of missing links on the predictive accuracy of defect models. Thus, we formulate the following research question:

(RQ2) How do defect identification approaches impact the predictive accuracy of defect models?

In addition to the predictive accuracy, Herzig *et al.* [21] find that noise that is generated by issue report mislabelling can impact the ranking of defective modules produced by defect models. Thus, we formulate the following research question:

(RQ3) How do defect identification approaches impact the ranking of defective modules produced by defect models?

III. STUDIED SOFTWARE SYSTEMS

In this section, we discuss our selection criteria and the studied systems that satisfy these criteria. In selecting the studied datasets, we identify two important criteria that needed to be satisfied:

- (C1) Issue-Driven Development.** Software development of the studied systems must be driven by the use of Issue Tracking System with high numbers of closed or fixed issue reports.
- (C2) Traceability.** The issue reports for each studied system must be traceable, i.e., an issue report must establish a link to the code change that addresses it. Hence, we only study systems where a large proportion of issue reports can be linked to the code changes that address them.

To satisfy criterion 1, we first select a set of systems from the corpus of software systems [44, 45] which extensively use JIRA Issue Tracking System. To satisfy criterion 2, we select 32 releases that span across 9 open-source software systems, i.e., ActiveMQ, Camel, Derby, Groovy, HBase, Hive, JRuby, Lucene, and Wicket. Table I provides an overview of the studied systems that satisfy our selection criteria where the studied systems vary in size, domain, and defective ratio in order to combat potential bias in our conclusions.

IV. CASE STUDY DESIGN

In this section, we describe the design of our case study that we perform to address our research questions. Figure 3 provides an overview of our case study design, which we describe in details below.

A. Data Extraction

(DE1) Extract issue data. For each studied system, we extract issue reports from the JIRA ITS. Particularly, we extract the unique identifier (IssueID), the issue report type (e.g., a bug or a new feature), and the affected releases (i.e., the releases that are affected by a given issue report).

(DE2) Identify post-release defects. We apply 2 defect identification approaches, which we describe below.

The heuristic approach. We first retrieve the code repository of the release of interest. Similar to prior studies [15, 17, 25, 56, 75], we apply a collection of regular expression patterns on commit logs in VCS to search for specific keywords and issue IDs within the 6-month period after the release of interest to identify post-release defect counts of each module. Then, we compute the post-release defect counts of each module by counting the number of defect reports that are associated with that module.

The realistic approach. As described in Section II-B, we first retrieve defect reports in ITS that affect the release of interest. Since JIRA ITS can automatically link issue reports with code changes, we compute the post-release defect counts of each module as the number of defect reports that are associated with that module and affect the release of interest.

(DE3) Extract metrics. Prior studies introduced many software metrics that impact software quality. Since we want to study a manageable set of software metrics, we focus on code, process, and ownership metrics. In total, we extract 65 metrics that consist of 54 code metrics, 5 process metrics, and 6 ownership metrics. Below, we describe each category of the studied software metrics.

Code metrics describe the relationship between code properties and software quality. We compute code metrics using the Understand tool from SciTools along 3 dimensions, i.e., complexity (e.g., McCabe Cyclomatic), volume (e.g., lines of code), and object-oriented (e.g., a coupling between object classes). Due to the space limitation, we refer the detailed explanation of each code metric to the SciTools website. Since some software metrics are computed at the method level, we select three aggregation schemes (i.e., min, max, and average) to aggregate these metrics to the file level. Table II provides a summary of the studied code metrics.

Process metrics describe the relationship between development activities and software quality. For each module, we collect the number of commits (COMM), the number of lines added (ADDED_LINES), the number of lines deleted (DEL_LINES), the number of active developers (ADDEV), and the number of distinct developers (DDEV). Similar to Rahman *et al.* [50], we normalize the values of the lines added and the lines deleted of a module by the total lines added and lines deleted.

TABLE I: A statistical summary of the studied systems.

Name	Description	#DefectReports	No. of Files	Defective Rate	KLOC	Studied Releases
ActiveMQ	Messaging and Integration Patterns server	3,157	1,884-3,420	6%-15%	142-299	5.0.0, 5.1.0, 5.2.0, 5.3.0, 5.8.0
Camel	Enterprise Integration Framework	2,312	1,515-8,846	2%-18%	75-383	1.4.0, 2.9.0, 2.10.0, 2.11.0
Derby	Relational Database	3,731	1,963-2,705	14%-33%	412-533	10.2.1.6, 10.3.1.4, 10.5.1.1
Groovy	Java-syntax-compatible OOP for JAVA	3,943	757-884	3%-8%	74-90	1.5.7, 1.6.0.Beta_1, 1.6.0.Beta_2
HBase	Distributed Scalable Data Store	5,360	1,059-1,834	20%-26%	246-534	0.94.0, 0.95.0, 0.95.2
Hive	Data Warehouse System for Hadoop	3,306	1,416-2,662	8%-19%	287-563	0.9.0, 0.10.0, 0.12.0
JRuby	Ruby Programming Lang for JVM	5,475	731-1,614	5%-18%	105-238	1.1, 1.4, 1.5, 1.7
Lucene	Text Search Engine Library	2,316	8,05-2,806	3%-24%	101-342	2.3.0, 2.9.0, 3.0.0, 3.1.0
Wicket	Web Application Framework	3,327	1,672-2,578	4%-7%	109-165	1.3.0.beta1, 1.3.0.beta2, 1.5.3

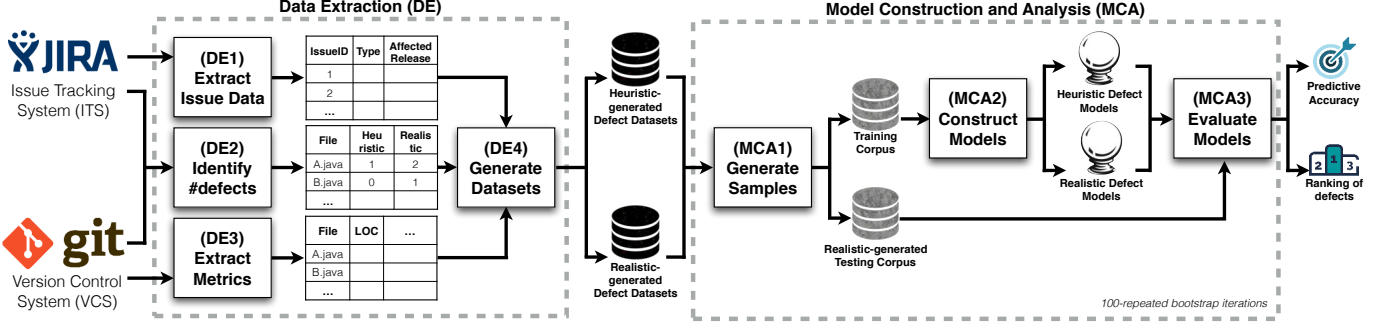


Fig. 3: An overview of our case study design.

Ownership metrics describe the relationship between the ownership of modules and software quality [8, 49, 66]. Following the traditional code ownership heuristics of Bird *et al.* [8], for each module, we first measure the ownership of each developer using the proportion of the code changes made by the developer on the total code changes. We consider two granularities of code changes, i.e., lines of code level (LINE), and commit level (COMMIT). Second, we classify developers into two levels of ownership of a module, as recommended by Bird *et al.* [8]. Developers with low code ownership (i.e., less than 5% code contribution on a module) are considered as minor authors. On the other hand, developers with high code ownership (i.e., more than 5% code contribution on a module) are considered as major authors. In our study, we collect the number of the owner (i.e., the developer with the highest code contribution on a module), minor authors, and major authors with respect to the two granularities of code changes.

Table III provides a summary of the studied process and ownership metrics.

(DE4) Generate defect datasets. We generate 2 types of defect datasets, i.e., defect count datasets and binary defect datasets. For defect count datasets, we combine the number of post-release defects from DE2 and the extracted software metrics from DE3. For binary defect datasets, we discretize the continuous defect counts of each module into “defective” class if a module has at least one defect, otherwise “clean” class. In total, we produce 4 datasets (i.e., 2 types of defect datasets and 2 types of defect identification approaches).

B. Model Construction and Analysis

We construct defect models using the defect datasets that are extracted from each studied system. We then analyze the predictive accuracy of these models. Figure 3 provides an

overview of our model analysis approach, which consists of four steps. We describe each step below.

(MCA1) Generate bootstrap samples. To ensure that our conclusions are statistically sound and robust, we use the out-of-sample bootstrap validation technique to generate training samples and estimate the model performance, since Tantihamthavorn *et al.* [57, 64] demonstrated that it produces the most stable performance estimates when comparing to other model validation techniques like k-fold cross-validation. We first generate bootstrap sample of sizes N with replacement from the original datasets. The generated sample is also of size N . We construct models using the generated bootstrap samples, while we measure the performance of the models using the samples that do not appear in the generated bootstrap samples. On average, 36.8% of the original dataset will not appear in the bootstrap samples, since the samples are drawn with replacement [16]. We repeat the out-of-sample bootstrap process for 100 times and report their average performance.

(MCA2) Construct defect models. To address RQ2 and RQ3, we focus on 2 types of defect models, i.e., defect count models and defect classification models. For each type of models, we apply 2 machine learning techniques (i.e, regression and random forest) on each training sample, since they often produce highly accurate predictions [23, 31, 62, 65]. For regression models, we construct a model formula in an additive fashion ($y = x_1 + x_2$) without considering model interactions. We note that we do not rebalance the training samples to avoid any concept drifts [59]. Prior to constructing defect models, we remove irrelevant and mitigate correlated metrics [24] using the implementation as provided by the `AutoSpearman` function of the `Rnalytica` R package [60].

A **defect count model** is a regression model that is used

TABLE II: A summary of the studied code metrics.

Granularity	Metrics	Count
File	AvgCyclomatic, AvgCyclomaticModified, AvgCyclomaticStrict, AvgEssential, AvgLine, AvgLineBlank, AvgLineCode, AvgLineComment, CountDeclClass, CountDeclClassMethod, CountDeclClassVariable, CountDeclFunction, CountDeclInstanceMethod, CountDeclInstanceVariable, CountDeclMethod, CountDeclMethodDefault, CountDeclMethodPrivate, CountDeclMethodProtected, CountDeclMethodPublic, CountLine, CountLineBlank, CountLineCode, CountLineCodeDecl, CountLineCodeExe, CountLineComment, CountSemicolon, CountStmt, CountStmtDecl, CountStmtExe, MaxCyclomatic, MaxCyclomaticModified, MaxCyclomaticStrict, RatioCommentToCode, SumCyclomatic, SumCyclomaticModified, SumCyclomaticStrict, SumEssential	37
Class	CountClassBase, CountClassCoupled, CountClassDerived, MaxInheritanceTree, PercentLackOfCohesion	5
Method	CountInput_{Min, Mean, Max}, CountOutput_{Min, Mean, Max}, CountPath_{Min, Mean, Max}, MaxNesting_{Min, Mean, Max}	12

TABLE III: A summary of the studied process and ownership metrics.

Metrics	Description
<i>Process Metrics</i>	
COMM	The number of Git commits
ADDED_LINES	The normalized number of lines added to the module
DEL_LINES	The normalized number of lines deleted from the module
ADEV	The number of active developers
DDEV	The number of distinct developers
<i>Ownership Metrics</i>	
MINOR_COMMIT	The number of unique developers who have contributed less than 5% of the total code changes (i.e., Git commits) on the module
MINOR_LINE	The number of unique developers who have contributed less than 5% of the total lines of code on the module
MAJOR_COMMIT	The number of unique developers who have contributed more than 5% of the total code changes (i.e., Git commits) on the module
MAJOR_LINE	The number of unique developers who have contributed more than 5% of the total lines of code on the module
OWN_COMMIT	The proportion of code changes (i.e., Git commits) made by the developer who has the highest contribution of code changes on the module
OWN_LINE	The proportion of lines of code written by the developer who has the highest contribution of lines of code on the module

to predict the number of post-release defects of a module. To construct defect count models, we apply both linear regression and random forest regression techniques. We use the implementation of the linear regression technique as provided by the `lm` function of the `base R` package [48]. We use the implementation of the random forest regression technique as provided by the `randomForest` R package [11].

A **defect classification model** is a classification model that is used to predict the probability of defect-proneness. To construct defect count models, we apply both logistic regression and random forest classification techniques. We use the implementation of logistic regression as provided by the `glm` function of the `stats R` package [48]. We use the implementation of the random forest regression technique as provided by the `randomForest` R package [11].

(MCA3) Evaluate defect models. We evaluate defect models along two dimensions, i.e., predictive accuracy, and ranking of defective modules.

Predictive accuracy measures indicate the ability of defect models to predict defective modules. We use 2 predictive accuracy measures to evaluate defect count models and 4 predictive accuracy measures to evaluate defect classification models, which we describe below.

For defect count models, we use the Mean Absolute Error

(MAE) and the Standardized Accuracy (SA), as suggested by prior work [30, 53, 54, 75]. MAE measures the mean of the absolute values between the predicted values and the actual values (i.e., $mean(|predicted - actual|)$). A low MAE value indicates that a defect count model can accurately predict the number of post-release defects. SA is the standardization of the MAE measure that is relative to random guessing (i.e., $SA = (1 - \frac{MAE}{MAE_{R_0}}) * 100$, where MAE_{R_0} is the MAE of random guessing). An SA value indicates how accurate a defect count model is when compared to random guessing. An SA value ranges between negative values (worse than random guessing), 0 (no better than random guessing), and positive values (better than random guessing).

For defect classification models, we use one threshold-independent measure, and 3 threshold-dependent measures, i.e., the Area Under the receiver operator characteristic Curve (AUC), Precision, Recall, and F-measure. AUC measures the ability of classifiers in discriminating between defective and clean modules, as suggested by prior studies [18, 31, 50, 58]. The value of AUC measure ranges between 0 (worst performance), 0.5 (no better than random guessing), and 1 (best performance) [20]. Precision measures the percentage of the predicted defective modules that are truly defective (i.e., $\frac{TP}{TP+FP}$). Recall measures the percentage of defective modules that are correctly classified by defect models (i.e., $\frac{TP}{TP+FN}$). F-measure is a harmonic mean of precision and recall measures (i.e., $\frac{2 \cdot precision \cdot recall}{precision + recall}$). Similar to prior studies [3, 76], we use the default probability value of 0.5 as a threshold value to compute the confusion matrix for threshold-dependent measures, i.e., if a module has a predicted probability above 0.5, it is considered defective; otherwise, the module is considered clean.

To calculate the aforementioned predictive accuracy measures, we use the implementation as provided by the `performance.calculation` function of the `Rnalytica` R package [60].

Defect ranking accuracy measures indicate the accuracy of defect models while considering the cost-effectiveness and developers' effort. In practice, it is not feasible and practical for developers to review and go through all of the modules to search for defects. Thus, developers often prioritize their limited resources on the most defective modules [50, 76].

For both defect count models and defect classification models, we use 3 measures, i.e., the precision at top 20% lines of code (P@20%LOC), the recall at top 20% lines of code (R@20%LOC), and the Spearman correlation coef-

ficient. $P@20\%LOC$ measures the precision of the ranking of the most defective modules in the top 20% lines of code. $R@20\%LOC$ measures the recall of the ranking of the most defective modules in the top 20% lines of code. Spearman correlation measures the correlation between the predicted defect counts and the actual defect counts. A high value of Spearman correlation coefficients indicates that a defect model can correctly assign the most defective modules in the top ranking of defective modules. We note that we only apply the Spearman correlation test on defect count datasets.

V. CASE STUDY RESULTS

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

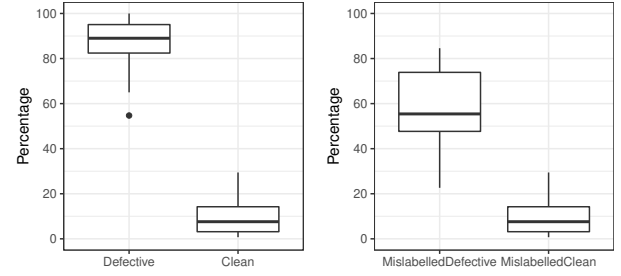
(RQ1) How do heuristic defect datasets and realistic defect datasets differ?

Approach. To answer RQ1, we investigate the difference between defect datasets that are generated using the heuristic approach and those that are generated using the realistic approach. We analyze along 2 perspectives of defect datasets, i.e., defect count datasets and binary defect datasets.

(RQ1-a) Defect count datasets. We start with the heuristic defect count datasets and the realistic defect count datasets (cf. DE4). For each dataset, we calculate the percentage of modules in heuristic defect datasets that have different defect counts from realistic defect datasets. Figure 4a presents the percentage of defective modules (i.e., modules in heuristic defect datasets that have at least one post-release defect) and clean modules (i.e., modules in heuristic defect datasets that do not have a post-release defect) that have different numbers of defect counts.

(RQ1-b) Binary defect datasets. We start with the heuristic binary defect datasets and the realistic binary defect datasets (cf. DE4). Then, we calculate the percentage of modules in heuristic defect datasets that are mislabelled (e.g., a module that is identified as clean by the heuristic approach but it is not identified as such by the realistic approach and vice versa). Figure 4b presents the percentage of defective modules in heuristic defect datasets that are mislabelled and the percentage of clean modules in heuristic defect datasets that are mislabelled.

Results. (RQ1-a) At the median, 89% of defective modules in heuristic defect datasets have different defect counts from those in realistic defect datasets. Figure 4a shows that 54%-100% of defective modules in heuristic defect datasets have different defect counts from those in realistic defect datasets. We also further investigate the nature of the difference of defect counts in defective modules. Figure 5 shows the percentage of defective modules where defect counts in heuristic defect datasets are higher, equal, and less than those in realistic defect datasets. We find that 22%-100% of defective modules in heuristic defect datasets have a higher defect count than realistic defect datasets, indicating that the heuristic approach identifies a large proportion of false post-release defects. The higher number of defect counts



(a) Defect Count Datasets (b) Binary Defect Datasets

Fig. 4: (a) The percentage of defective modules (left) and clean modules (right) in heuristic defect datasets that have different numbers of defect counts. (b) The percentage of defective modules in heuristic defect datasets that are mislabelled (left) and the percentage of clean modules in heuristic defect datasets that are mislabelled (right).

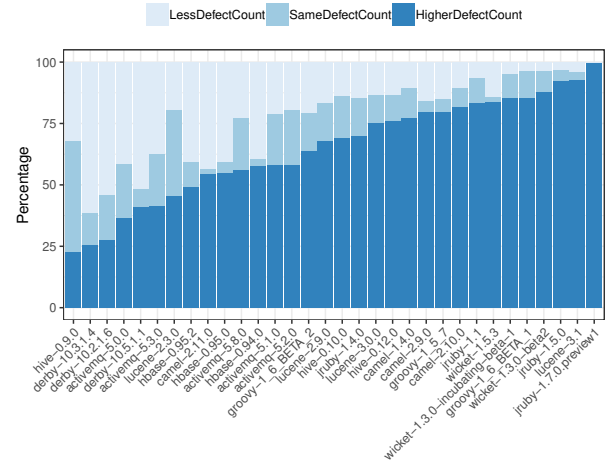


Fig. 5: The percentage of defective modules where defect counts in heuristic defect datasets are higher (i.e., false defect reports), equal, and less (i.e., missing defect reports) than those in realistic defect datasets.

of defective modules in heuristic defect datasets has to do with the inconsideration of the affected-release metadata of the heuristic approach. In other words, there are issue reports that are addressed within the 6-month period but they do not affect the release of interest. Moreover, we find that 0%-61% of defective modules in heuristic defect datasets have a lower defect count in realistic defect datasets, indicating that the heuristic approach misses a large proportion of realistic post-release defects. The lower number of defect counts of defective modules in heuristic defect datasets has to do with the window thresholds of the post-release period that are made by the heuristic approach. In other words, there are issue reports that affect the release of interest but they are addressed after the 6-month period of the release of interest.

On the other hand, at the median, 7% of clean modules

in heuristic defect datasets have different defect counts from those in realistic defect datasets. Figure 4a shows that 0%-29% of clean modules in heuristic defect datasets have different defect counts from those in realistic defect datasets, suggesting that when the modules are identified as clean by the heuristic approach, it is less likely that the modules will miss the realistic post-release defects.

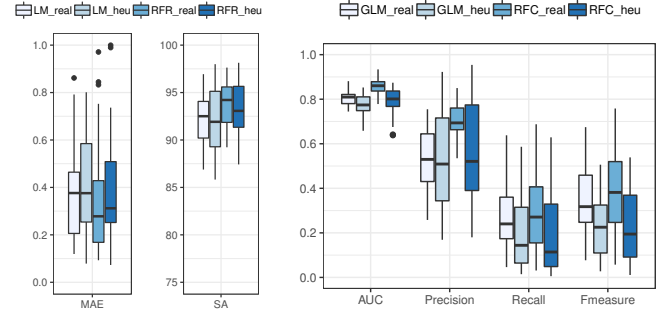
(RQ1-b) At the median, 55% of defective modules and 8% of clean modules in heuristic defect datasets are mislabelled. Figure 4b shows that 22%-84% of defective modules in heuristic defect datasets are actually clean in realistic defect datasets. The high number of mislabelled defective modules in heuristic defect datasets has to do with the discretization of the defect counts. As shown in Figure 5 that many of defective modules in heuristic datasets have false post-release defects, it is more likely that many of clean modules in realistic datasets are labelled as defective by the heuristic approach. On the other hand, Figure 4b shows that 0%-29% of clean modules in heuristic defect datasets are actually defective in realistic defect datasets. This finding is consistent with our previous finding that only a few of clean modules in heuristic defect datasets have different defect counts from those in realistic defect datasets.

The heuristic approach impacts the defect counts of 89% of defective modules in defect count datasets and mislabels 55% of defective modules in binary defect datasets, raising concerns related to the validity and the reliability of defect models that are constructed using heuristic defect datasets.

(RQ2) How do defect identification approaches impact the predictive accuracy of defect models?

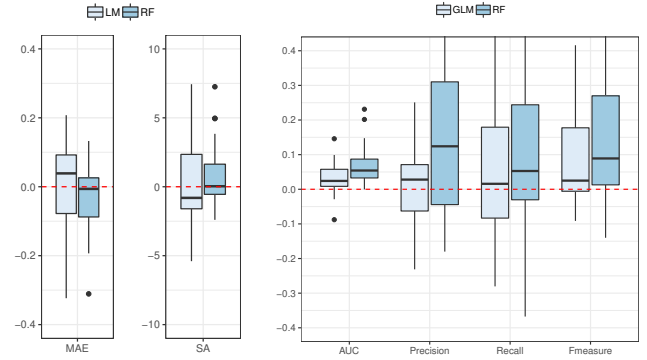
Approach. To address RQ2, we analyze the predictive accuracy of defect models that are constructed using heuristic defect datasets and realistic defect datasets. To do so, for each defect dataset, we first generate 100 bootstrap samples (cf. MCA1). We then construct defect count models and defect classification models using these bootstrap samples (cf. MCA2) and evaluate the predictive accuracy of the models (cf. MCA3). Figure 6 shows the distributions of the predictive accuracy for both defect count models and defect classification models. The positive values of SA measure for defect count models (Figure 6a) and the AUC values of above 0.5 for defect classification models (see Figure 6b) indicate that our defect models perform better than random guessing.

To quantify the impact of defect identification approaches on the predictive accuracy of defect models, we use Wilcoxon signed-rank tests ($\alpha = 0.05$) to detect whether there is a statistically significant difference in the predictive accuracy between the defect models that are constructed using heuristic defect datasets and those that are constructed using realistic defect datasets. We use Wilcoxon signed-rank tests instead of paired t-tests to relax the assumptions of normal distributions. To quantify the magnitude of the difference, we use Cliff's $|\delta|$ effect size [51] with the interpretation of Romano *et al.* [51]:



(a) Defect Count Models (b) Defect Classification Models

Fig. 6: The distributions of predictive accuracy estimates of (a) defect count models and (b) defect classification models that are constructed using heuristic and realistic defect datasets for both regression and random forest techniques.



(a) Defect Count Models (b) Defect Classification Models

Fig. 7: The distributions of predictive accuracy difference of (a) defect count models and (b) defect classification models that are constructed using realistic and heuristic defect datasets (i.e., $\text{Measure}_{\text{realistic}} - \text{Measure}_{\text{heuristic}}$).

negligible for $|\delta| \leq 0.147$, small for $|\delta| \leq 0.33$, medium for $|\delta| \leq 0.474$, and large for $|\delta| \geq 0.474$.

In addition, we report the difference of the predictive accuracy between the defect models that are constructed using heuristic defect datasets and those that are constructed using realistic defect datasets (i.e., $\text{Measure}_{\text{realistic}} - \text{Measure}_{\text{heuristic}}$) as shown in Figure 7.

Results. Surprisingly, there is no statistically significant difference in the predictive accuracy between defect count models that are constructed using heuristic defect datasets and realistic defect datasets. Table IV shows the results of Wilcoxon signed-rank tests and Cliff's $|\delta|$ effect size of the differences for each predictive accuracy measure. For both MAE and SA measures of the defect count models, the statistical and effect size test results show that the differences are not statistically significant (p-value ≥ 0.05) with negligible to small effect sizes. Figure 7a shows the distributions of the predictive accuracy difference between the defect count

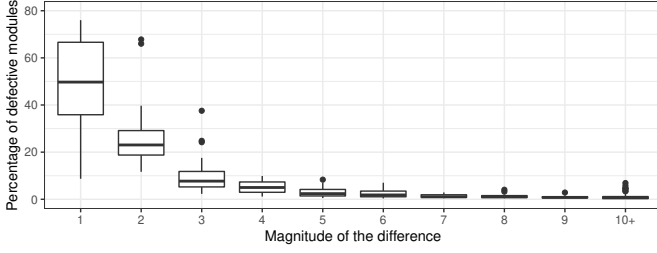
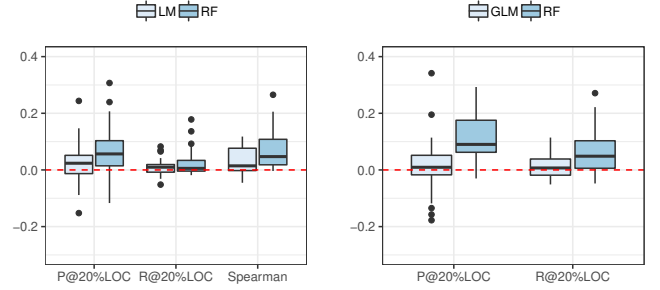


Fig. 8: The magnitude of the differences for defect counts of defective modules in heuristic defect datasets.

models that are constructed using heuristic and realistic defect datasets. We observe that the distributions of the difference for all of the predictive accuracy measures are centered at 0. We find that the difference for MAE, at the median, is 0.039 and 0 for linear regression models and random forest regression models, respectively. Similarly, we find that the difference for SA, at the median, is -0.80 and 0.08 for linear regression models and random forest regression models, respectively. We suspect that the negligible difference for MAE and SA has to do with the majority of one defect count difference in defective modules of heuristic defect datasets (see Figure 8), indicating that defect count models do not have as much of an impact on the one defect count difference. Thus, we recommend that *future studies should not be too concerned about the quality of heuristic-generated defect datasets when constructing defect count models.*

On the other hand, defect classification models that are constructed using realistic defect datasets lead to an increase in the accuracy of defect models that is rather modest. Table IV shows that the differences are statistically significant ($p\text{-value} \leq 0.05$) with medium to large effect sizes for both F-measure and AUC measures. Figure 7b shows the distributions of the predictive accuracy difference between defect classification models that are constructed using realistic and heuristic defect datasets. We find that the difference for AUC at the inter-quartile range (i.e., from the 1st to 3rd quantiles) is 0-0.06 and 0.03-0.08 for logistic regression models and random forest classification models, respectively. Similarly, we find that the difference for F-measure at the inter-quartile range is 0-0.18 and 0-0.27 for logistic regression models and random forest classification models, respectively. However, Figure 7b shows that the median difference for AUC measure is 0.03-0.05, indicating that the realistic approach leads to an increase in the accuracy of defect models that is rather modest.

Surprisingly, there is no statistically significant difference in the predictive accuracy between defect count models that are constructed using heuristic and realistic defect datasets. On the other hand, defect classification models that are constructed using realistic defect datasets lead an increase in the predictive accuracy that is rather modest.



(a) Defect Count Models (b) Defect Classification Models

Fig. 9: The distributions of defect ranking accuracy difference of (a) defect count models and (b) defect classification models that are constructed using realistic and heuristic defect datasets (i.e., $\text{Measure}_{\text{realistic}} - \text{Measure}_{\text{heuristic}}$).

(RQ3) How do defect identification approaches impact the ranking of defective modules produced by defect models?

Approach. To address RQ3, we analyze the ranking of defective modules produced by defect models that are constructed using heuristic and realistic defect datasets. Similar to RQ2, we generate 100 bootstrap samples (*cf.* MCA1), construct defect count models and defect classification models (*cf.* MCA2), and evaluate the ranking of defective modules produced by the models (*cf.* MCA3). We use three ranking measures, i.e., precision at top 20% lines of code (P@20%LOC), recall at top 20% lines of code (R@20%LOC), and Spearman correlation (ρ). To quantify the impact of defect identification approaches on the ranking of defective modules, we use Wilcoxon signed-rank tests ($\alpha = 0.05$) to check whether there is a statistically significant difference. To quantify the magnitude of the difference, we use Cliff's $|\delta|$ effect size. In addition, we report the difference of the ranking measures between the defect models that are constructed using heuristic defect datasets and those that are constructed using realistic defect datasets (i.e., $\text{Measure}_{\text{realistic}} - \text{Measure}_{\text{heuristic}}$) as shown in Figure 9.

Results. The heuristic approach has a negligible impact on the ranking of defective modules produced by defect count models. Table IV shows the results of Wilcoxon signed-rank tests and the Cliff's $|\delta|$ effect size of the differences for each ranking measure. While our statistical tests suggest that these differences are statistically significant ($p\text{-value} \leq 0.05$), the differences have a negligible Cliff's $|\delta|$ effect size. Figure 9a shows the distributions of the difference of the ranking measures between the defect count models that are constructed using heuristic and realistic defect datasets. We observe that the distributions of the difference in the ranking measures are centered around 0. For defect count models, we find that the difference of P@20%LOC is 0.3 and 0.4 percentage points for linear regression models and random forest regression models, respectively. We find that the difference of R@20%LOC is 0.01 and 0.01 percentage points for linear regression models and random forest regression models, respectively. We find that

TABLE IV: Results of the Wilcoxon signed-rank tests and the Cliff’s $|\delta|$ effect size tests for the predictive accuracy (RQ2) and the ranking of defective modules (RQ3) between heuristic and realistic defect models.

Techniques	Predictive Accuracy (RQ2)						Ranking of Defective Modules (RQ3)				
	Defect Count		Defect Classification				Defect Count			Defect Classification	
	MAE	SA	AUC	Precision	Recall	F-measure	P@20%LOC	R@20%LOC	Spearman	P@20%LOC	R@20%LOC
Regression	N ^o	N ^o	M***	N ^o	S ^o	M*	N***	N***	N***	N***	N***
Random Forest	S ^o	S ^o	L***	S*	M*	L***	N***	N***	N***	S***	S***

Statistical significance: $\circ p \geq .05$, $* p < .05$, $** p < .01$, and $*** p < .001$.

Effect size: Negligible (N) Cliff’s $|\delta| \leq 0.147$, Small (S) Cliff’s $|\delta| \leq 0.33$, Medium (M) Cliff’s $|\delta| \leq 0.474$, and Large (L) otherwise.

the difference of Spearman is 0.01 and 0.05 percentage points for linear regression models and random forest regression models, respectively. These results lead us to conclude that the impact of the heuristic approach on the ranking measures of defect count models is small, which confirms our suggestion in RQ2.

Similarly, the heuristic approach has a negligible to small impact on the ranking of defective modules produced by defect classification models. While our statistical tests suggest that these differences are statistically significant (p -value ≤ 0.05), these differences have a negligible to small Cliff’s $|\delta|$ effect size. Figure 9b shows the distributions of the difference of the ranking measures between the defect classification models that are constructed using heuristic and realistic defect datasets. We observe that the distributions of the difference in the ranking measures are centered around 0. For defect classification models, we find that the median difference of P@20%LOC is 0.01 and 0.09 percentage point for logistic regression models and random forest classification models, respectively. We find that the median difference of R@20%LOC is 0.01 and 0.05 percentage points for logistic regression models and random forest classification models, respectively. These results lead us to conclude that the heuristic approach does not have much impact on the ranking of defective modules that are produced by defect classification models, which confirms our suggestion in RQ2.

The heuristic approach has a negligible to small impact on the ranking of defective modules produced by both defect count models and defect classification models.

VI. THREATS TO VALIDITY

Like any empirical studies, the settings of experimental design may impact the results of our study [57, 63]. We now discuss threats to the validity of our study.

A. Internal Validity

The realistic estimation of the earliest affected release for a given defect that is suggested by da Costa *et al.* [14] heavily relies on the software development team of a software project. Thus, we are aware that the estimation of the earliest affected release that is provided by the development team may be incomplete or wrongly estimated. Nevertheless, it is impractical to ask domain experts to verify whether all of the issue reports of the nine studied systems are correctly filled.

B. Construct Validity

We studied a limited number of regression and classification techniques. Thus, our results may not be generalized to other regression and classification techniques. Nonetheless, other regression and classification techniques can be explored in future work. We provide a detailed methodology and studied defect datasets for others who would like to re-examine our findings using unexplored techniques.

Recent work [47] raised concerns that F-measure neglects the True Negative predictions. To mitigate any potential biases in our results, we also use an AUC measure, which is a threshold-independent measure.

C. External Validity

The studied defect datasets are part of several systems from Apache Software Foundation (ASF) of the JIRA ITS. However, we studied a limited number of software systems. Thus, the results may not generalize to other datasets, domains, ecosystems. Nonetheless, additional replication studies in a proprietary setting and other ecosystems may prove fruitful.

The conclusions of our case study rely on one defect prediction scenario (i.e., within-project defect models). However, there are a variety of defect prediction scenarios in the literature (e.g., cross-project defect prediction [13, 68, 76], just-in-time defect prediction [25], heterogenous defect prediction [41]). Therefore, the conclusions may differ for other scenarios. Thus, future research should revisit our study in other scenarios of defect models.

VII. CONCLUSIONS

In this paper, we investigate (1) the nature of the difference of defect datasets that are generated by the heuristic and realistic approaches; and its impact on (2) the predictive accuracy and (3) the ranking of defective modules that are produced by defect models. Through a case study of defect datasets of 32 releases, we find that the heuristic approach has a large impact on both defect count datasets and binary defect datasets. Surprisingly, we find that the heuristic approach has a minimal impact on defect count models, suggesting that future work should not be too concerned about defect count models that are constructed using heuristic defect datasets. On the other hand, using defect datasets generated by the realistic approach lead to an improvement in the predictive accuracy of defect classification models.

REFERENCES

- [1] G. Antoniol, K. Ayari, M. D. Penta, and F. Khomh, "Is it a Bug or an Enhancement? A Text-based Approach to Classify Change Requests," in *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research (CASCON)*, 2008, pp. 1–15.
- [2] J. Aranda and G. Venolia, "The Secret Life of Bugs: Going Past the Errors and Omissions in Software Repositories," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2009, pp. 298–308.
- [3] 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.
- [4] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The Missing Links : Bugs and Bug-fix Commits Categories and Subject Descriptors," in *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE)*, 2010, pp. 97–106.
- [5] V. R. Basili, L. C. Briand, and W. L. Melo, "A Validation of Object-oriented Design Metrics as Quality Indicators," *Transactions on Software Engineering (TSE)*, vol. 22, no. 10, pp. 751–761, 1996.
- [6] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and Balanced? Bias in Bug-Fix Datasets," in *Proceedings of the joint meeting of the European Software Engineering Conference and the Foundations of Software Engineering (ESEC/FSE)*, 2009, pp. 121–130.
- [7] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? an empirical case study of windows vista," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2009, pp. 518–528.
- [8] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: examining the effects of ownership on software quality," in *Proceedings of the joint meeting of the European Software Engineering Conference and the Foundations of Software Engineering (ESEC/FSE)*, 2011, pp. 4–14.
- [9] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The Promises and Perils of Mining Git," in *Proceedings of the International Working Conference on Mining Software Repositories (MSR)*, 2009, pp. 1–10.
- [10] D. Bowes, T. Hall, M. Harman, Y. Jia, F. Sarro, and F. Wu, "Mutation-Aware Fault Prediction," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, 2016, pp. 330–341.
- [11] L. Breiman, A. Cutler, A. Liaw, and M. Wiener, "randomForest : Breiman and Cutler's Random Forests for Classification and Regression. R package version 4.6-12." *Software available at URL: <https://cran.r-project.org/web/packages/randomForest>*, 2006.
- [12] B. Caglayan, B. Turhan, A. Bener, M. Habayeb, A. Miransky, and E. Cialini, "Merits of organizational metrics in defect prediction: an industrial replication," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015, pp. 89–98.
- [13] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective Cross-project Defect Prediction," in *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST)*, 2013, pp. 252–261.
- [14] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan, "A framework for evaluating the results of the szz approach for identifying bug-introducing changes," *Transactions on Software Engineering (TSE)*, vol. 43, no. 7, pp. 641–657, 2017.
- [15] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating Defect Prediction Approaches : A Benchmark and an Extensive Comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.
- [16] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. Boston, MA: Springer US, 1993.
- [17] M. Fischer, M. Pinzger, and H. Gall, "Populating a Release History Database from Version Control and Bug Tracking Systems," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, 2003, pp. 23–32.
- [18] 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 International Conference on Software Engineering (ICSE)*, 2015, pp. 789–800.
- [19] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," *Transactions on Software Engineering (TSE)*, vol. 38, no. 6, pp. 1276–1304, 2012.
- [20] 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.
- [21] K. Herzig, S. Just, and A. Zeller, "It's not a Bug, it's a Feature: How Misclassification Impacts Bug Prediction," in *Proceedings of the International Conference on Software Engineering (ICSE)*, no. Section XII, 2013, pp. 392–401.
- [22] R. Hosseini, B. Turhan, and D. Gunarathna, "A Systematic Literature Review and Meta-analysis on Cross Project Defect Prediction," *Transactions on software Engineering (TSE)*, 2017.
- [23] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The Impact of Correlated Metrics on Defect Models," *Transactions on Software Engineering (TSE)*, p. To Appear, 2019.
- [24] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "AutoSpearman: Automatically Mitigating Correlated Metrics for Interpreting Defect Models," in *Proceeding of the International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 92–103.
- [25] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A Large-Scale Empirical Study of Just-In-Time Quality Assurance," *Transactions on Software Engineering (TSE)*, vol. 39, no. 6, pp. 757–773, 2013.
- [26] F. Khomh, M. Di Penta, and Y.-G. Gueheneuc, "An Exploratory Study of the Impact of Code Smells on Software Change-proneness," in *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, 2009, pp. 75–84.
- [27] F. Khomh, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change-and fault-proneness," *Empirical Software Engineering*, vol. 17, no. 3, pp. 243–275, 2012.
- [28] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with Noise in Defect Prediction," in *Proceedings of the International Conference on Software engineering (ICSE)*, 2011, pp. 481–490.
- [29] V. Kovalenko, F. Palomba, and A. Bacchelli, "Mining file histories: Should we consider branches?" in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2018, pp. 202–213.
- [30] W. B. Langdon, J. Dolado, F. Sarro, and M. Harman, "Exact Mean Absolute Error of Baseline Predictor, MARP0," *Information and Software Technology*, vol. 73, pp. 16–18, 2016.
- [31] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," *Transactions on Software Engineering (TSE)*, vol. 34, no. 4, pp. 485–496, 2008.
- [32] G. A. Lieben and M. Shepperd, "Data sets and data quality in software engineering," in *Proceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE)*, 2008, pp. 39–44.
- [33] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The Impact of Code Review Coverage and Code Review Participation on Software Quality," in *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, 2014, pp. 192–201.
- [34] T. Menzies, J. Greenwald, and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors," *Transactions on Software Engineering (TSE)*, vol. 33, no. 1, pp. 2–13, 2007.
- [35] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Automated Software Engineering (ASE)*, vol. 17, no. 4, pp. 375–407, 2010.
- [36] T. Menzies and T. Zimmermann, "Software Analytics: So What?" *Software*, vol. 30, no. 4, pp. 31–37, 2013.
- [37] N. Nagappan and T. Ball, "Use of Relative Code Churn Measures to Predict System Defect Density," *Proceedings of the International Conference on Software Engineering (ICSE)*, pp. 284–292, 2005.

- [38] N. Nagappan, T. Ball, and A. Zeller, "Mining Metrics to Predict Component Failures," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2006, pp. 452–461.
- [39] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2008, pp. 521–530.
- [40] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change Bursts as Defect Predictors," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, 2010, pp. 309–318.
- [41] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous Defect Prediction," *Transactions on Software Engineering (TSE)*, vol. 44, no. 9, pp. 874–896, 2017.
- [42] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Multi-layered approach for recovering links between bug reports and fixes," in *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE)*, 2012, pp. 1–11.
- [43] T. H. Nguyen, B. Adams, and A. E. Hassan, "A Case Study of Bias in Bug-Fix Datasets," in *Proceedings of the Working Conference on Reverse Engineering (WCORE)*, 2010, pp. 259–268.
- [44] M. Ortu, G. Destefanis, B. Adams, A. Murgia, M. Marchesi, and R. Tonelli, "The jira repository dataset: Understanding social aspects of software development," in *Proceedings of the International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*, 2015, p. 1.
- [45] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams, "The emotional side of software developers in jira," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2016, pp. 480–483.
- [46] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshyanyk, and A. De Lucia, "Mining version histories for detecting code smells," *Transactions on Software Engineering (TSE)*, vol. 41, no. 5, pp. 462–489, 2015.
- [47] D. M. Powers, "What the F-measure doesn't measure: Features, Flaws, Fallacies and Fixes," *arXiv preprint arXiv:1503.06410*, 2015.
- [48] R Core Team, "R: A language and environment for statistical computing," <http://www.R-project.org/>, R Foundation for Statistical Computing, Vienna, Austria, 2013.
- [49] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2011, pp. 491–500.
- [50] —, "How, and Why, Process Metrics are Better," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2013, pp. 432–441.
- [51] J. Romano, J. Kromrey, J. Coraggio, and J. Skowronek, "Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys," in *Annual meeting of the Florida Association of Institutional Research*, 2006.
- [52] S. Ruangwan, P. Thongtanunam, A. Ihara, and K. Matsumoto, "The Impact of Human Factors on the Participation Decision of Reviewers in Modern Code Review," *Empirical Software Engineering (EMSE)*, p. In press, 2018.
- [53] F. Sarro and A. Petrozziello, "LP4EE: Linear Programming as a Baseline for Software Effort Estimation," *Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, p. No. 12, 2018.
- [54] M. Shepperd and S. MacDonell, "Evaluating Prediction Systems in Software Project Estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.
- [55] E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan, "Understanding the Impact of Code and Process Metrics on Post-release Defects: A Case Study on the Eclipse Project," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2010, p. No. 4.
- [56] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *Software Engineering Notes (SEN)*, vol. 30, no. 4, p. 1, 2005.
- [57] C. Tantithamthavorn, "Towards a Better Understanding of the Impact of Experimental Components on Defect Prediction Modelling," in *Companion Proceeding of the International Conference on Software Engineering (ICSE)*, 2016, pp. 867–870.
- [58] C. Tantithamthavorn and A. E. Hassan, "An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges," in *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, 2018, pp. 286–295.
- [59] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The Impact of Class Rebalancing Techniques on the Performance and Interpretation of Defect Prediction Models," *Transactions on Software Engineering (TSE)*, 2018.
- [60] C. Tantithamthavorn and J. Jiarpakdee, "Rnalytica: An R package of JIRA defect datasets and tool suites for explainable software analytics," <https://github.com/awsm-research/Rnalytica>.
- [61] 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 (ICSE)*, 2015, p. 812–823.
- [62] 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.
- [63] —, "Comments on 'Researcher Bias: The Use of Machine Learning in Software Defect Prediction'," *Transactions on Software Engineering (TSE)*, vol. 42, no. 11, pp. 1092–1094, 2016.
- [64] —, "An Empirical Comparison of Model Validation Techniques for Defect Prediction Models," *Transactions on Software Engineering (TSE)*, vol. 43, no. 1, pp. 1–18, 2017.
- [65] —, "The Impact of Automated Parameter Optimization on Defect Prediction Models," *Transactions on Software Engineering (TSE)*, p. In Press, 2018.
- [66] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "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)*, 2016, pp. 1039–1050.
- [67] —, "Review Participation in Modern Code Review," *Empirical Software Engineering (EMSE)*, vol. 22, no. 2, pp. 768–817, 2017.
- [68] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.
- [69] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE)*, 2015, pp. 805–816.
- [70] C. Vassallo, F. Palomba, A. Bacchelli, and H. C. Gall, "Continuous code quality: Are we (really) doing that?" in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2018, pp. 790–795.
- [71] S. Wang, T. Liu, and L. Tan, "Automatically Learning Semantic Features for Defect Prediction," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2016, pp. 297–308.
- [72] 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)*, 2011, pp. 15–25.
- [73] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *Transactions on Software Engineering (TSE)*, vol. 42, no. 10, p. 977, 2016.
- [74] S. Yatish, J. Jiarpakdee, P. Thongtanunam, and C. Tantithamthavorn, "Replication Package for ICSE2019 Mining Software Defects: Should We Consider Affected Releases?" <https://doi.org/10.5281/zenodo.2549359>, 2019.
- [75] T. Zimmermann and N. Nagappan, "Predicting subsystem failures using dependency graph complexities," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, 2007, pp. 227–236.
- [76] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project Defect Prediction," in *Proceedings of the joint meeting of the European Software Engineering Conference and the Foundations of Software Engineering (ESEC/FSE)*, 2009, pp. 91–100.