The Impact of Data Merging on the Interpretation of Cross-Project Just-In-Time Defect Models

Dayi Lin, *Member, IEEE*, Chakkrit Tantithamthavorn, *Member, IEEE*, and Ahmed E. Hassan, *Fellow, IEEE*

Abstract—Just-In-Time (JIT) defect models are classification models that identify the code commits that are likely to introduce defects. Cross-project JIT models have been introduced to address the suboptimal performance of JIT models when historical data is limited. However, many studies built cross-project JIT models using a pool of mixed data from multiple projects (i.e., data merging)—assuming that the properties of defect-introducing commits of a project are similar to that of the other projects, which is likely not true. In this paper, we set out to investigate the interpretation of JIT defect models that are built from individual project data and a pool of mixed project data with and without consideration of project-level variances. Through a case study of 20 datasets of open source projects, we found that (1) the interpretation of JIT models that are built from individual projects varies among projects; and (2) the project-level variances cannot be captured by a JIT model that is trained from a pool of mixed data from multiple projects without considering project-level variances (i.e., a global JIT model). On the other hand, a mixed-effect JIT model that considers project-level variances represents the different interpretations better, without sacrificing performance, especially when the contexts of projects are considered. The results hold for different mixed-effect learning algorithms. When the goal is to derive sound interpretation of cross-project JIT models, we suggest that practitioners and researchers should opt to use a mixed-effect modelling approach that considers individual projects and contexts.

Index Terms—Just-in-time defect prediction, data merging, mixed-effect model, cross-project defect prediction

1 Introduction

A Just-In-Time (JIT) defect model is a classification model that identifies the code commits that are likely to introduce defects [23], [25], [35], [40], [45], [47]. Such JIT models are critically important for continuous quality assurance practices to early prioritize code commits with the highest defect-proneness for code review and testing due to limited quality assurance resources. In addition, knowledge that is derived from such JIT models are often used to continuously chart quality improvement plans to avoid past pitfalls (i.e., what commit-level metrics are associated with the likelihood of introducing defects?) [15], [24], [31], [42], [50].

Recent work raises concerns that the performance of JIT models is often suboptimal for software projects with limited historical training data [5], [22]. Moreover, such data are also unavailable in the initial software development phases of many projects. To address this challenge, Fukushima *et al*. [5] show that cross-project JIT models (i.e., models trained using historical data from other projects) are as accurate as JIT models that are trained on a single project (i.e., within-project JIT models). Recently, Kamei *et al*. [22] show that the performance of JIT models that are built using a pool of mixed project data (i.e., *merged data*) from several projects is comparable to within-project performance.

Despite the advantages of the data merging practice for building cross-project JIT models, prior work raises concerns that the distribution of metric values often varies across projects [59], [60]. The findings of these studies raise a critical concern that the interpretation of a cross-project JIT model that is built from a pool of mixed project data may not hold true for a JIT model that is built from an individual project. Yet, the impact of the data merging practice on the interpretation of JIT models remains largely unexplored.

In this paper, we set out to investigate the interpretation of three types of cross-project JIT models when compared to the interpretation of *local JIT models*—i.e., a JIT model that is built from an individual project:

1) *global JIT models*: a cross-project JIT model that is built from a pool of mixed project data assuming that the data is collected from the same project;
2) *project-aware JIT models*: a cross-project JIT model that is built from a pool of mixed project data with a consideration of the different projects that the data is from; and
3) *context-aware JIT models*: a cross-project JIT model that is built from a pool of mixed project data with a consideration of the different contextual factors of the different projects that the data is from (e.g., programming language).

To do so, we build project-aware and context-aware JIT models using a mixed-effect modelling approach (i.e., taking both commit-level and project-level factors into consideration when modelling) with two commonly-used classifiers, logistic

- *Dayi Lin is with the Centre for Software Excellence, Huawei, ON L3R 5A4, Canada. E-mail: dayi.lin@huawei.com.*
- *Chakkrit Tantithamthavorn is with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia. E-mail: chakkrit@monash.edu.*
- *Ahmed E. Hassan is with the School of Computing, Queen's University, Kingston, ON K7L 3N6, Canada. E-mail: ahmed@cs.queensu.ca.*

Manuscript received 13 November 2019; revised 5 March 2021; accepted 13 April 2021. Date of publication 19 April 2021; date of current version 15 August 2022. (Corresponding author: Dayi Lin.)
Recommended for acceptance by K. Sen.
Digital Object Identifier no. 10.1109/TSE.2021.3073920

0098-5589 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

regression and random forest. We also extensively evaluate the performance of our JIT models using 9 measures, including 4 threshold-dependent measures (i.e., Precision, Recall, F1-score, G-score), one threshold-independent measure (i.e., AUC), and 4 practical measures (i.e., Initial False Alarms, PCI@20 percent, P_{opt}, and P_{opt}@20%). Through a case study of 20 datasets of open source projects, we address the following six research questions:

(RQ1) Does the interpretation of the local JIT models vary?

Our results show that the most important metric (i.e., the metric that has the highest impact on the likelihood of a commit introducing defects) of the local JIT models and the baseline likelihood of introducing defects (i.e., the likelihood of introducing defects when we remove the impact from all commit-level metrics) vary among projects, suggesting that the interpretation of global JIT models that are trained from merged data may be misleading when project-level variances and the characteristics of the studied projects are not considered.

(RQ2) How consistent is the interpretation of the global JIT model when compared to the local JIT models?

Our results show the most important metric of the global JIT model is consistent with 55 percent of the studied projects' local JIT models, suggesting that the interpretation of global JIT models cannot capture the variation of the interpretation for all local JIT models. Moreover, the baseline likelihood of introducing defects of the global JIT model is not consistent with the local JIT models, suggesting that conclusions that are derived from the global JIT model are not generalizable to many of the studied projects.

(RQ3) How accurate are the predictions of the project-aware JIT model and the context-aware JIT model when compared to the global JIT model?

Our results show that project-aware and context-aware JIT models that consider project context achieve comparable performance across nine measured performance metrics with global JIT models that do not consider project context. The comparable performance allows us to further compare the interpretation among global, project-aware and context-aware JIT models in the following RQs.

(RQ4) How consistent is the interpretation of the project-aware JIT model when compared to the local JIT models and the global JIT model?

Our results show that the project-aware JIT model can provide a more representative interpretation than the global JIT model, while also providing a better fit to the merged dataset from different projects, with a 26 percent increase of R^2 compared to the average of local JIT models and a 86 percent increase of R^2 compared to a global JIT model.

(RQ5) How consistent is the interpretation of the context-aware JIT model when compared to the local JIT models and the global JIT model?

Our results show that the context-aware JIT model can provide both a more representative interpretation and a better fit to the dataset than the global JIT model, with a 35 percent increase of R^2 compared to the average of local JIT models and 100 percent increase of R^2 compared to a global JIT model. The inclusion of the contextual factors in the JIT model when using mixed-effect modelling approaches can yield more in-depth interpretation, while maintaining a good fit to the dataset.

(RQ6) Do our prior conclusions hold for other mixed-effect classifiers?

Similar to the project-aware and context-aware JIT models that are based on mixed-effect regression modelling, our results show that the project-aware and context-aware JIT models that use mixed-effect random forest modelling also achieved a similar performance compared to random forest based global JIT models. In addition, both project-aware and context-aware mixed-effect random forest models achieve a better goodness-of-fit compared to random forest based global JIT model.

Our findings suggest that irrespective of classifiers, using mixed-effect modelling with either logistic regression or random forest classifier to build JIT models can provide better interpretation by taking project-level variances into consideration, without sacrificing the performance of JIT models. When the goal is to derive sound interpretation of cross-project JIT models, we suggest that practitioners and researchers should opt to use a mixed-effect modelling approach that considers individual projects and contexts.

Novelty Statements. To the best of our knowledge, this paper is the first to investigate the performance and interpretation of the context-aware and project-aware JIT models using a mixed-effect modelling approach. In particular, this paper makes the following contributions:

1) An investigation of the variation of the interpretation of local JIT models.
2) A comparison of the interpretation of a global JIT model to the interpretation of local JIT models.
3) An investigation of the performance of project-aware and context-aware JIT models with respect to global JIT models.
4) A comparison of the interpretation of a project-aware JIT model to the interpretation of local JIT models.
5) A comparison of the interpretation of a context-aware JIT model to the interpretation of local JIT models.
6) An evaluation of mixed-effect regression models and mixed-effect random forest models for JIT models.
7) An improved implementation of generalized mixed-effect random forest models [29], which improves prediction of unseen projects during training, and improves training speed with a fast implementation of random forests provided by the `ranger` package in R.

In addition to our conceptual contributions, this paper is also the first to develop and provide detailed technical description of a mixed-effect modelling for cross-project JIT

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

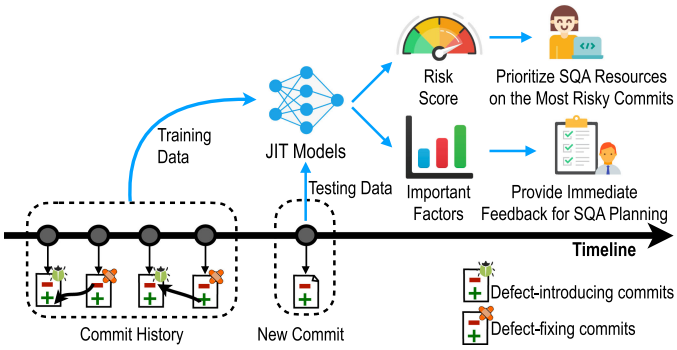

Fig. 1. The predictions of JIT defect models are used to early predict the risk of defect-introducing commits, while the interpretation of JIT defect models is used to provide immediate feedback on the most important characteristics of defect-introducing commits to support SQA planning.

defect models. We also provide a replication package [29] including our datasets and scripts for the community to evaluate the results.

Paper Organization. Section 2 introduces the importance of the interpretation of Just-In-Time defect models. Section 3 motivates the impact of the data merging on the interpretation of Just-In-Time defect models. Section 4 discusses and motivates our six research questions with respect to prior work. Section 5 presents the design of our case study. Section 6 presents the results of each research question. Section 7 provides practical guidelines for future studies. Section 8 discusses the threats to the validity of our study. Finally, Section 9 concludes the paper.

2 Background

2.1 Software Quality Assurance (SQA) Planning

Software Quality Assurance (SQA) planning is the process of developing proactive SQA plans to define the quality requirements of software products and processes [42]. Such proactive SQA plans will be used as guidance to prevent software defects that may slip through to future releases. However, the development of such SQA plans are still ad-hoc and based on practitioners' beliefs, e.g., Devanbu *et al.* [4] found that practitioners form beliefs based on their personal experience and these can vary across teams and projects. They also may not necessarily align with actual evidence derived from the projects. To cope with the continuous software development practices, researchers suggested to derive insights and lessons learned from Just-In-Time (JIT) Defect Models in order to better understand what factors are the most important to describe the characteristics of defect-introducing commits [23], [25], [35], [40], [45], [47].

2.2 Just-In-Time (JIT) Defect Prediction

A Just-In-Time (JIT) Defect Model is a classification model that is trained on the characteristics of commits in order to predict if a commit will introduce defects in the future [23], [25], [35], [40], [45], [47] and explain the characteristics of defect-introducing commits [15], [23], [31]. To date, JIT models have been widely adopted in many software organizations like Avaya [35], Cisco [47], and Blackberry [45]. In summary, Just-In-Time (JIT) defect models serve two main purposes as shown in Fig. 1.

To Predict. First, JIT defect models are used to early predict the risk of defect-introducing commits so developers can prioritize SQA resources on the most risky commits in a cost-effective manner. JIT defect models are trained using numerous factors [23], [40], e.g., the number of added lines, deleted lines, code churn, entropy. Prior studies found that various machine learning techniques (e.g., random forest and regression models) demonstrated a promising accuracy when predicting defect-introducing commits [5].

To Explain. Second, JIT defect models are used to provide immediate feedback on the most important characteristics of defect-introducing commits to not only support QA leads and managers develop Software Quality Assurance (SQA) plans, but also aid developers in pre-commit testings and code reviews [15]. Such feedback or insights are derived from the interpretation of the JIT models through model interpretation techniques (e.g., ANOVA for regression analysis, variable importance analysis for random forest). The goal of model interpretation is to better understand what are the most important characteristics of defect-introducing commits. Such data-informed insights can help QA leads and Managers to develop proactive software quality improvement plans to prevent pitfalls that lead to defects in the past [31], and aid developers in pre-commit testings and code reviews. For example, if churn (i.e., the amount of changed lines per commit) shares the strongest relationship with the likelihood of defect-introducing commits, QA leads and managers should establish a software quality improvement plan to strictly control churn (e.g., each commit should not contain churn that exceeds 1,000 lines of code) to control the quality of software process and mitigate the risk of introducing software defects to the code commits.

3 A Motivating Example

The interpretation of JIT defect models heavily relies on the dataset that was used in training. Traditionally, JIT models are often trained from an individual project (a.k.a. *a local JIT model*) so the interpretation is specific to the project that is used in training. However, when a company starts a new software project, the company may have limited access to the historical data that can be used to train JIT defect models for the new project. Thus, cross-project JIT defect models have been proposed to address this challenge of limited historical data [13], [26], [55], [62].

Prior studies proposed to merge data from different projects (i.e., a pool of mixed project data) to develop universal or cross-project JIT defect models (a.k.a. *a global JIT defect model*) [5], [22], [56]. The intuition is that a larger diverse pool of defect data from several other projects may provide a more robust model fit that will be applied better in a cross-project context. However, when datasets are combined from other projects, each project often has different data characteristics (e.g., different data distributions, different project characteristics). Thus, such global JIT defect models that are trained from a mixed-project dataset may produce misleading interpretations of JIT defect models when comparing to a local JIT defect model.

To illustrate the impact of data merging on the interpretation of JIT defect models, we conduct the following motivating analysis. We start by building three local JIT defect

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
(A Motivating Example) The Percentage Importance Scores of Three Local JIT Models (i.e., Postgres, Accumulo, and Django) When Comparing to a Global JIT Model That is Trained From All Projects

	Postgres	Accumulo	Django	Global
Entropy	16%	**87%**	34%	**52%**
NS	17%	3%	**51%**	16%
NF	**36%**	3%	1%	0%
Churn	1%	0%	0%	0%
LT	3%	5%	7%	0%
FIX	27%	1%	7%	32%
$\mathbf{R^2}$	0.31	0.20	0.23	0.14
AUC	0.66	0.78	0.75	0.71

models, where each of them is trained on a single project (i.e., Accumulo, Postgres, Django). We also build a global JIT model that is trained from a mixed-project dataset of 20 projects (see Table 2). These JIT models are built using logistic regression and the percentage importance scores are computed from an ANOVA Type-II analysis [17]. Finally, we compute the goodness-of-fit of the models using R^2 and compute a median AUC using an out-of-sample boostrap model validation technique [52]. Based on Table 1, we draw the following observation.

While the three local JIT models and the global JIT model achieve a comparable AUC and R^2, the interpretation of each JIT model is different. Table 1 shows that the most important metric is the number of changed files for the Postgres project, Entropy for the Accumulo project, and the number of subsystems for the Django project. This indicates that software quality improvement plans should be dependent on the project.

On the other hand, *the global JIT model that is trained from a pool of mixed project data cannot capture the variation.* Table 1 shows that the most important metric is Entropy for the global JIT model, indicating that such insights may not be applicable to all projects.

This motivating example highlights the need of a modern regression alternative that can capture the project variation when aiming to draw the generalization of the most important metrics. Below, we discuss how does misleading interpretation of JIT defect models impact practitioners and researchers.

Importance for Practitioners. The interpretation of JIT defect models plays a critical role to not only support QA leads and managers develop quality improvement plans, but also aid developers in code reviews and pre-commit testings [15], [16], [42]. Such interpretation could provide an immediate feedback of pitfalls that lead to software defects in the past so practitioners can avoid in the future. Unfortunately, when using the interpretation of the global JIT defect model which is trained from a pool of mixed project dataset, the interpretation derived from the global JIT defect model cannot capture the variation from project to project, producing misleading insights. Such misleading insights could lead to suboptimal software development policies, wasting time and resources when adopting in practice.

Importance for Researchers. Recently, researchers aim to draw generalized conclusions by deriving a conclusion from a large-scale study via the combination of datasets from multiple projects [5], [22]. However, such generalizations may not hold true for each project, posing a critical threat to the external validity of prior studies. Therefore, a modern regression alternative (i.e., a mixed-effect modelling approach) is needed to capture the project variation and context variation. However, there exists no study investigating if a mixed-effect modelling approach can capture the project variation and context variation for JIT defect models.

4 Research Questions

Prior work shows that data merging from multiple projects without considering project context tends to perform well for cross-project JIT models [5], [22]. However, recent work raised concerns that the distribution of software metrics often varies among project contexts (e.g., domain, size, and programming language) [59], [60], [61]. Such variation of distributions likely leads JIT models to produce different metrics that influence defect-introducing commits among projects. Yet, little is known whether metrics that influence defect-introducing commits vary among projects. Thus, we formulate the following research question:

> *RQ1: Does the interpretation of the local JIT models vary?*

Recently, Kamei *et al.* [22] suggest that data merging from multiple projects without considering project contexts tends to perform well for cross-project JIT models, suggesting that a simple data merging technique (i.e., a pool of mixed project data) would likely suffice for cross-project JIT models. Yet, little is known whether the interpretation of a global JIT model is consistent with local JIT models. Thus, we formulate the following research question:

> *RQ2: How consistent is the interpretation of the global JIT model when compared to the local JIT models?*

The practice of data merging without considering project-level variances has been widely used in many studies on cross-project JIT models [5], [22] and cross-project defect models [11], [12], [13], [55]. Such practice assumes that historical commits are collected from similar projects, which is likely not true. While prior studies reinforce the consideration of project-level variances for cross-project modelling [2], [33], [59], [60], Kamei *et al.* [22] argued that project-aware rank transformation does not work well for cross-project JIT models. In addition, Herbold *et al.* [12] also argued that project-aware data partitioning only yields a minor improvement for cross-project defect models.

Even though prior work has reinforced that project-level variances must be considered, little research has paid attention to a modern regression alternative, i.e., *the mixed-effect modelling approach* [1], especially in the context of JIT defect models. In addition, little is known whether the performance of project-aware JIT models (i.e., a JIT model that is trained on a mixed project data while considering project-level variances) and context-aware JIT models (i.e., a JIT model that is trained on a mixed project data while

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

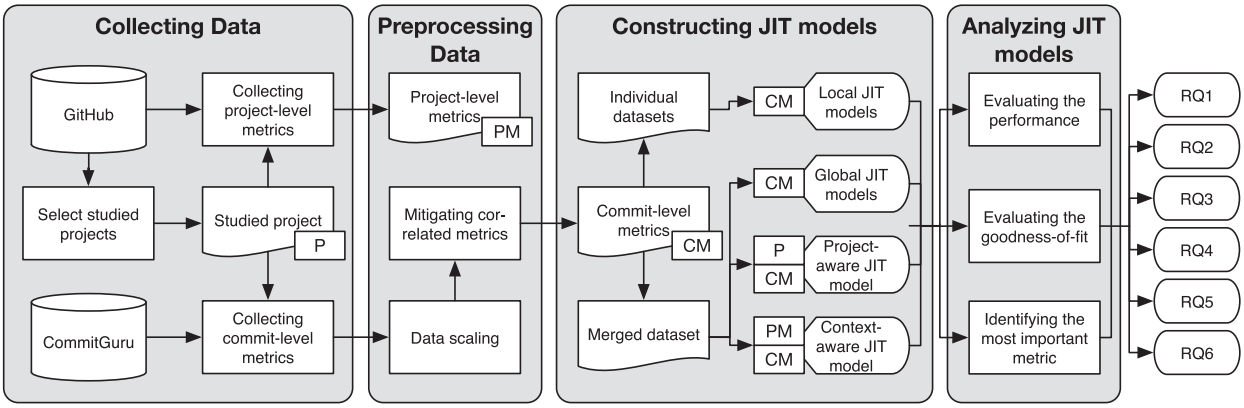

Fig. 2. An overview of our case study design.

considering project characteristics) that use mixed-effect modelling approaches are comparable to global JIT models. Thus, we formulate the following research question:

RQ3: How accurate are the predictions of the project-aware JIT model and the context-aware JIT model when compared to the global JIT model?

Recently, Hassan *et al.* [10] pointed out that the mixed-effect modelling approach is able to capture the variation of the interpretation of models among different datasets in the context of mobile apps reviews. Yet, little is known whether project-aware JIT models might produce more representative metrics that influence defect-introducing commits when compared to the global JIT model and the local JIT models. Thus, we formulate the following research question:

RQ4: How consistent is the interpretation of the project-aware JIT model when compared to the local JIT models and the global JIT model?

One limitation of the project-aware JIT model is that we cannot interpret the impact of project contexts (e.g., domain, size, and programming language) on the defect-proneness of the project. Prior work has shown that the distribution of software metrics often varies among project contexts [61]. Yet, it is unknown whether a context-aware JIT model might produce more representative interpretation when comparing to the global JIT model and the local JIT models. Thus, we formulate the following research question:

RQ5: How consistent is the interpretation of the context-aware JIT model when compared to the local JIT models and the global JIT model?

Many prior studies have explored different classifiers for defect modelling, such as logistic regression [23] and random forest [22]. In RQ1 to RQ5, we focus on the logistic regression classifier. To better understand if the conclusions from RQ1 to RQ5 hold for other classifiers like random forest, we formulate the following research question:

RQ6: Do our prior conclusions hold for other mixed-effect classifiers?

5 Case Study Design

In this section, we describe our selection criteria for the studied software projects, and the design of our case study to address the six research questions. Fig. 2 presents an overview of our case study design.

5.1 Collecting Data

5.1.1 Studied Software Projects

In order to address our research questions, we defined two important criteria for selecting studied software projects:

1) *Criterion 1 - Publicly-available datasets:* To foster replications of our study, we select studied software projects that are hosted in a publicly-available data repository (i.e., GitHub).
2) *Criterion 2 - Large and long-term development:* To ensure the quality of our studied projects and avoid including any small projects in GitHub, we selected studied software projects that are large and have been developed for a long period of time.

We randomly selected 20 open source projects that meet the criteria from GitHub for our study. Table 2 gives an overview of the studied projects. We collected the commits of each project from GitHub on February 14th, 2018. We used CommitGuru [44] to extract commit-level metrics (i.e., metrics that influence the likelihood of a commit introducing defects) and identified defect-introducing commits using the SZZ algorithm [46] for each project.

5.1.2 Collecting Commit-Level Metrics

Prior studies proposed many commit-level metrics that are associated with the likelihood of introducing defects [23], [25], [35], [45], [47]. Similar to Kamei *et al.* [22], we used CommitGuru [44] to collect eight metrics that span 3 categories. Table 3 provides a brief description of the commit-level metrics.

Diffusion category measures how distributed a commit is. A highly distributed commit is more complex and more prone to defects, as shown in prior work [9], [35]. We collected the number of modified subsystems (NS), the number of modified directories (ND), the number of modified files

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Summary of Studied Software Projects

Project name	Date of first commit	Lines of code	# of changes
accumulo	Oct 4, 2011	600,191	9,175 (21%)
angular	Jan 5, 2010	249,520	8,720 (25%)
brackets	Dec 7, 2011	379,446	17,624 (24%)
bugzilla	Aug 26, 1998	78,448	9,795 (37%)
camel	Mar 19, 2007	1,310,869	31,369 (21%)
cinder	May 3, 2012	434,324	14,855 (23%)
django	Jul 13, 2005	468,100	25,453 (42%)
fastjson	Jul 31, 2011	169,410	2,684 (26%)
gephi	Mar 2, 2009	129,259	4,599 (37%)
hibernate-orm	Jun 29, 2007	711,086	8,429 (32%)
hibernate-search	Aug 15, 2007	174,475	6,022 (35%)
imglib2	Nov 2, 2009	45,935	4,891 (29%)
jetty	Mar 16, 2009	519,265	15,197 (29%)
kylin	May 13, 2014	214,983	7,112 (25%)
log4j	Nov 16, 2000	37,419	3,275 (46%)
nova	May 27, 2010	430,404	49,913 (26%)
osquery	Jul 30, 2014	91,133	4,190 (23%)
postgres	Jul 9, 1996	1,277,645	44,276 (33%)
tomcat	Mar 27, 2006	400,869	19,213 (28%)
wordpress	Apr 1, 2003	390,034	37,937 (47%)

Parenthesized values show the percentage of defect-introducing commits.

(NF), and the distribution of modified code across each file (Entropy), to measure the diffusion of a commit. Similar to Hassan [9], we normalized the entropy by the maximum entropy $log_2 n$ to take the differences in the number of files n across changes into account.

Size category measures the size of a commit using the lines added (LA), lines deleted (LD), and lines total (LT). The intuition is that the size of a commit is a strong indicator of the commit's defect-proneness [36], [37].

Purpose category measures whether a commit fixes a defect. The intuition is that a commit that fixes a defect is more likely to introduce another defect [6], [41].

5.1.3 Collecting Project-Level Metrics

To investigate the impact of project-level variances on the interpretation of JIT models, we collected 9 project-level metrics, which were used in prior work [59], [61], [62]. We briefly outline the project-level metrics below. Among the 9 project-level metrics, 6 of them can be extracted from the version control systems (i.e., Language, NLanguage, LOC, NFILE, NCOMMIT, NDEV), and 3 of them require manual tagging (Audience, UI, Database). For each numeric project-level metrics (NLanguage, LOC, NFILE, NCOMMIT, NDEV), we separated the values into four groups based on the first, second, and third quartiles (i.e., least, less, more, most), as suggested by prior work [22].

TABLE 3
Summary of Commit-Level Metrics

Category	Name	Description
Diffusion	NS	Number of modified subsystems
	ND	Number of modified directories
	NF	Number of modified files
	Entropy	Distribution of modified code across each file
Size	LA	Lines of code added
	LD	Lines of code deleted
	LT	Lines of code in a file before the commit
Purpose	FIX	Whether or not the commit is a defect fix

Language (Java / JavaScript / Perl / Python / PHP / C / C ++): Programming language that is used the most in the project. We identified the programming language using `cloc`. We chose the programming language that is used in the largest number of files as the programming language of a project.

NLanguage (Least / Less / More / Most): Total number of programming languages that are used in the project. We consider a programming language being used in a project if more than 10 percent of the files are written in that programming language.

LOC (Least / Less / More / Most): Total lines of code of the source code in the project.

NFILE (Least / Less / More / Most): Total number of files in the project.

NCOMMIT (Least / Less / More / Most): Total number of commits in the VCS of the project.

NDEV (Least / Less / More / Most): Total number of unique developers in the system.

Audience (Developer / User): Whether the intended audience of the project is end users (e.g., wordpress), or development professionals (e.g., log4j).

UI (Toolkit / GUI / Non-interactive): The type of user interaction of the project. E.g., imglib2 is a toolkit, gephi has GUI, and cinder is Non-interactive.

Database (True / False): Whether the project store the data in a database or not.

5.2 Preprocessing Data

5.2.1 Data Scaling

We observed that most commit-level metrics are highly skewed, and in different scales. To address this issue, we

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

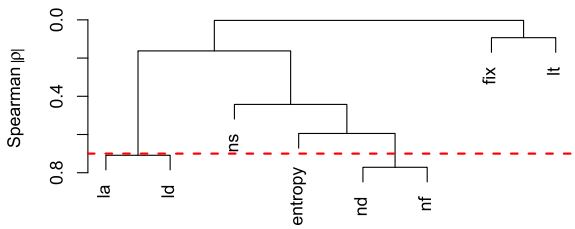

Fig. 3. Hierarchical overview of the correlation among the commit-level metrics. The dotted line shows the threshold ($|\rho| = 0.7$).

centred and scaled the commit-level metrics using the `scale` function in R, except for the "FIX" metric which is boolean.

5.2.2 Mitigating Correlated Metrics

Highly correlated metrics will produce incorrect interpretation of JIT models [17], [18], [48]. We used the Spearman correlation to mitigate collinearity (i.e., the correlation between 2 metrics), and the redundancy analysis to mitigate multicollinearity (i.e., the correlation across more than 2 metrics, in other words, the ability of a metric being described by 2 or more other metrics).

We calculated the Spearman correlation among all the studied commit-level metrics, and manually removed the highly correlated metrics with a Spearman correlation coefficient greater than 0.7. We chose 0.7 as the threshold, because this threshold has been widely-used in defect prediction research [15], [17], [19], [20], [31], [32], [48], [54]. In addition, Jiarpakdee *et al.* [15], [19] also found that the variation of the Spearman correlation coefficient between 0.5-0.7 has little impact on the conclusions. We avoid using other coefficients so the conclusions of our paper is strictly controlled and not biased toward the use of other coefficients. We chose Spearman correlation because it is resilient to data that are not normally distributed. Fig. 3 shows the hierarchically clustered Spearman ρ values of the commit-level metrics, from merged data across all the studied projects.

Fig. 3 shows that ND and NF are highly correlated. We removed ND and kept NF for our study, as suggested by prior work [23]. In addition, LA and LD are highly correlated. We replace LA and LD with a relative churn metric (i.e., $(LA + LD)/LT$), as suggested by prior work [22], [23], [37].

Redundant metrics (i.e., metrics that do not have a unique signal from other metrics) will interfere with each other and produce misleading interpretation of models. We used the `redun` function in R to detect redundant metrics. We found that after the correlation analysis, there exists no redundant metrics.

5.2.3 Class Imbalance

We observed that the class labels of our JIT datasets are imbalanced, i.e., only a small proportion of all commits introduces defects. Prior work in just-in-time cross-project defect prediction used class rebalancing techniques to improve model performance [22]. However, Tantithamthavorn *et al.* [48], [49] have shown that balancing data tends to shift the ranking of the most important metrics. To avoid introducing any possible bias in the interpretation of our JIT models, we did not apply data rebalancing techniques in our study.

5.3 Constructing JIT Models

To address our six research questions, we constructed four types of JIT models, i.e., local JIT models, global JIT models, project-aware JIT models, and context-aware JIT models using the logistic regression and random forest classifiers.

5.3.1 Local JIT Models

A local JIT model is a model that is built from an individual project. We construct local JIT models using logistic regression for each of the 20 studied projects where we used commit-level metrics as independent variables and whether a commit introduces defects as the dependent variable.

In a classic logistic regression model, the relationship between commit-level metrics (independent variables x_i) and the likelihood of the commit introducing defects (dependent variable y) can be described as

$$Ln\left(\frac{y}{1-y}\right) = \beta_0 + \beta_i x_i + \epsilon, \tag{1}$$

where ϵ is the standard error. The *coefficient* β_i indicates *the relationship between the ith commit-level metric and the likelihood of the commit introducing defects*, and the *intercept* β_0 indicates *the baseline likelihood of introducing defects of a project*.

We used the implementation of logistic regression from the `glm` function that is provided by the `stats` R package.

5.3.2 Global JIT Models

A global JIT model is trained using a pool of mixed project data assuming that data is collected from the same project. We first combined data from all the studied projects as a training dataset. We constructed a global JIT model using logistic regression (see Section 5.3.1) with the merged dataset using the same dependent and independent variables, and the same R package as the local JIT models. Since the goal of our paper focuses on the interpretation of cross-project JIT models, we trained our model using the full dataset.

As the global JIT model is trained using a merged dataset with data from all the studied projects, but the coefficient β_i and the intercept β_0 in the model are fixed, commits from different projects have the same relationship with the dependent variable y (the likelihood of a commit introducing defects). Such models are called fixed-effect models, and have been used in prior studies for JIT models [5], [22].

5.3.3 Project-Aware JIT Models

A fixed-effect JIT model with merged dataset has an assumption that the relationship between commit-level metrics and the likelihood of a commit introducing defects, and the baseline likelihood of introducing defects of a project, are the same across different projects, which is likely not true. To build a project-aware JIT model that considers the differences among different projects, we constructed a mixed-effect logistic regression model [1] for our study.

Unlike classic logistic regression models, a mixed-effect logistic regression model contains both fixed effects (independent variables at the commit level) and random effects

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

TABLE 4
The Model Statistics of the Local JIT Models for Each Studied Project

Project name	(Intercept)	Entropy		NS		NF		Relative churn		LT		FIX		R^2	AUC
	Coef.	Coef.	% χ^2	Coef.	% χ^2	Coef.	% χ^2	Coef.	% χ^2	Coef.	% χ^2	Coef.	% χ^2		
cinder	-1.95	↗	40***	↗	**48***	↘	0°	↗	0°	↗	0**	↗	12***	0.43	0.86
nova	-1.6	↗	38***	↗	**51***	↘	0°	↘	0°	↗	2***	↗	9***	0.38	0.84
postgres	-0.85	↗	16***	↗	17***	↗	**36***	↗	1***	↘	3***	↗	27***	0.31	0.65
angular	-1.53	↗	18***	↗	**64***	↗	0°	↗	1*	↗	0°	↗	17***	0.30	0.80
osquery	-1.61	↗	41***	↗	**43***	↗	1°	↗	0°	↗	11***	↗	4***	0.30	0.82
brackets	-1.52	↗	13***	↗	**85***	↘	0°	↗	0°	↗	0°	↗	2***	0.29	0.79
camel	-1.62	↗	**73***	↗	0°	↗	0°	↘	0°	↘	26***	↗	0**	0.24	0.68
django	-0.56	↗	34***	↗	**51***	↘	1***	↘	0°	↘	7***	↗	7***	0.23	0.76
accumulo	-1.57	↗	**87***	↗	3***	↗	3***	↘	0°	↗	5***	↗	1**	0.20	0.78
bugzilla	-0.98	↗	8***	↗	42***	↘	0°	↘	1*	↗	**46***	↗	3***	0.18	0.72
fastjson	-1.24	↗	**82***	↘	4**	↗	1°	↗	0°	↗	14***	↗	0°	0.18	0.72
jetty	-1.13	↗	**70***	↘	25***	↗	2***	↘	0°	↗	0°	↗	3***	0.18	0.73
hibernate-search	-0.75	↗	**48***	↘	38***	↗	4***	↗	1*	↘	9***	↗	0°	0.16	0.67
hibernate-orm	-0.87	↗	**95***	↘	2***	↗	2***	↗	0°	↘	1**	↗	0°	0.14	0.67
kylin	-1.28	↗	**86***	↗	14***	↗	0°	↘	0°	↘	0°	↗	0°	0.14	0.71
log4j	-0.29	↗	**41***	↗	17***	↗	0°	↘	0°	↘	33***	↗	8***	0.11	0.67
tomcat	-1.12	↗	**38***	↗	13***	↗	24***	↘	0°	↗	9***	↗	16***	0.11	0.64
gephi	-0.54	↗	**88***	↘	7***	↗	1*	↗	1°	↗	3**	↘	0	0.09	0.64
imglib2	-0.98	↗	**92***	↗	1°	↘	0°	↘	0°	↗	6***	↗	1°	0.09	0.67
wordpress	-0.4	↘	1***	↗	24***	↗	12***	↘	0°	↗	1°	↗	**63***	0.08	0.63
global JIT model	-1.10	↗	**52***	↗	16***	↗	0***	↘	0°	↗	0***	↗	32***	0.14	0.70

↗ : positive coefficients. ↘ : negative coefficients.
Statistical significance of χ^2: $^\circ p \geq 0.05$; $^* p < 0.05$; $^{**} p < 0.01$; $^{***} p < 0.001$.
The metric with the highest percentage of χ^2 for each project is highlighted in bold and red. We also report the goodness-of-fit of the local JIT models using R^2 and the predictive accuracy using AUC.

(independent variables at the project level), and therefore is able to represent different relationships between independent variables and dependent variables at different hierarchical levels (i.e., different projects).

There are two types of mixed-effect models: (1) random intercept models, and (2) random slope and intercept models. *Random intercept models* have different intercepts for independent variables at the project level, but fixed coefficients for independent variables at the commit level. On the other hand, *random slope and intercept models* allow different intercepts for independent variables at the project level, and different coefficients for independent variables at the commit level. As we suspect that commit-level metrics from different projects have different relationships with the likelihood of a commit introducing defects, we constructed a random slope and intercept model for the project-aware JIT model. A random slope and intercept model takes the following form:

$$Ln\left(\frac{y}{1-y}\right) = \beta_0 + \beta_i x_i + u_{j0} + u_{jk} z_k + \epsilon, \quad (2)$$

where y, x_i, β_i, ϵ has the same definition as Formula (1); u_{j0} is the jth random intercept; z_k is the kth random effect; and u_{jk} is the jth random slope for the kth random effect.

In particular, we use a unique identifier of a project (i.e., the project name) as the random intercept, and use Entropy as the random slope against project in our model, i.e., u_{j0} is the random intercept for jth project; z_1 is Entropy and u_{j1} is the random slope for the Entropy of jth project. We use the project name as the random intercept, so that the project-aware JIT model can give each project a different intercept, similar to the local JIT models (Section 5.3.1); instead of treating data from different projects the same and only computing one general intercept (β_0), as done when training a global JIT model (Section 5.3.2). We use Entropy as the random slope, since Table 4 shows that Entropy is the most important metric for 11 out of 20 studied projects. We do not include other metrics as random slopes, since excessive usage of random slope increases the model's number of degrees of freedom and therefore increases the risk of overfitting. The intuitive interpretation is that we let different projects to have different baseline likelihood of introducing defects ($\beta_0 + u_{j0}$), and allow Entropy to have a different relationship (u_{j1}) with the likelihood of a commit introducing defects for each project. We use the rest of the commit-level metrics (NS, NF, LT, FIX and relative churn) as fixed effects (x_i). We built the project-aware JIT models using the implementation of the mixed-effect regression model provided by the `glmer` function in the R package `lme4`.

5.3.4 Context-Aware JIT Models

In contrast to the project-aware JIT model, a context-aware JIT model further differentiates various contextual factors of different projects, instead of simply giving projects their unique intercepts. To construct a context-aware JIT model, we built a mixed-effect model (more specifically, a random slope and intercept model). In the context-aware JIT model, we introduced the 9 contextual factors (the 9 project-level metrics that are described in Section 5) as random intercepts (i.e., each level of each contextual factor has its unique intercept). All of the contextual factors are at project level instead of commit level, hence they impact the project's baseline likelihood of introducing defects (i.e., in the form of intercept instead of slope). As a result, the intercept for the jth project (u_{j0}) in the context-aware JIT model can be considered conceptually as the sum of the intercepts of the project's contextual factors

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

$$u_{j0} = \sum_{m=1}^{M} v_{jm0}, \quad (3)$$

where M is the total number of contextual factors (i.e., 9), and v_{jm0} is the coefficient of random intercept for jth project's mth contextual factor, obtained through the fitting of the context-aware JIT model. We calculated u_{j0} conceptually as the sum of all coefficients of random intercepts of a specific project, so that we can compare that to the local JIT models and the project-aware JIT model where each project has only one intercept value.

We used the same random slope and fixed effects in the context-aware JIT model as the project-aware JIT model. Again, we built the context-aware JIT models using the implementation of the mixed-effect regression model provided by the `glmer` function in the R package `lme4`.

5.4 Analyzing JIT Models

5.4.1 Evaluating the Performance of JIT Models

A defect-introducing commit can be classified by a JIT model as defect-introducing (true positive, TP) or non-defect-introducing (false negative, FN); while a non-defect-introducing commit can be classified by a JIT model as defect-introducing (false positive, FP) or non-defect-introducing (true negative, TN).

To measure the performance of the constructed JIT models, we employed nine performance metrics that are commonly used in prior work [14], [22], [23], defined as follows:

Precision measures the ratio of correctly predicted defect-introducing commits to all commits that are predicted as defect-introducing ($Precision = \frac{TP}{TP+FP}$).

Recall measures the ratio of correctly predicted defect-introducing commits to all defect-introducing commits ($Recall = \frac{TP}{TP+FN}$).

F1-score measures the harmonic mean of recall and precision. There exists a tradeoff between precision and recall. F1-score is a commonly used metric to combine precision and recall ($F1\text{-}score = \frac{2*Precision*Recall}{Precision+Recall}$).

G-score is an alternative metric to F1-score to avoid potential negative impact of imbalanced class on F1-score [27]. G-score is the harmonic mean between the probability of true positive and probability of true negative ($G\text{-}score = \frac{2*TP*TN}{TP+TN}$).

AUC measures the Area Under the Curve (AUC) of the Receiver Operating Curve (ROC). AUC is used to mitigate the potential bias of the choice of the probability threshold for precision and recall and mitigate the class imbalance that are commonly presented in our datasets [49]. As suggested by Mandrekar [30], an AUC value of 0.5 indicates that a model performs equally to random guessing or no discrimination, 0.7 to 0.8 is considered acceptable, 0.8 to 0.9 is considered excellent, and more than 0.9 is considered outstanding.

IFA measures the number of Initial First Alarms (IFA) that developers would encounter before they find the first defective commit. In contrast to the above mentioned measures, IFA takes into consideration the human aspect of quality assurance. A low IFA may increase developers' trust on the JIT model.

PCI@20% measures the Proportion of Commits Inspected when 20 percent of modified LOC by all commits are inspected. PCI@20% focuses on the effort that developers spend when inspecting the suggested defect-introducing commits by the JIT model.

P_{opt}: An optimal JIT model ranks defect-introducing commits by the decreasing actual bug density. When plotting the percentage of defect-introducing commits against the percentage of effort for both the optimal JIT model and the JIT model that is being evaluated, we can calculate the area between the two models' curves Δ_{opt}. P_{opt} is calculated as $1 - \Delta_{opt}$. Hence, a larger P_{opt} means a smaller difference between the optimal JIT model and the JIT model being evaluated.

P_{opt} @20%: Similar to the P_{opt} measure, P_{opt}@20% measures P_{opt} before the cutoff of 20 percent of effort.

5.4.2 Evaluating the Goodness-of-Fit of JIT Models

To measure how well the constructed models fit the data, we calculated the conditional coefficient of determination for generalized logistic regression models and mixed-effect models (R^2 or R^2_{GLMM}) [21], [38].

$$R^2_{GLMM} = \frac{\sigma_f^2 + \sum_{l=1}^{u} \sigma_l^2}{\sigma_f^2 + \sum_{l=1}^{u} \sigma_l^2 + \sigma_\epsilon^2 + \sigma_d^2},$$

where σ_f^2 is the variance of the fixed effects, and $\sum \sigma_l^2$ is the sum of all u variance components, σ_ϵ^2 is the variance due to the additive dispersion and σ_d^2 is the distribution-specific variance. We used the implementation of the R^2_{GLMM} fuction that is provided by the `MuMIn` R package.

5.4.3 Identifying the Most Important Metric

We used the χ^2 value of each commit-level metric that is obtained from the ANOVA Type-II to measure the impact of commit-level metrics on the likelihood of a commit introducing defects. The χ^2 value measures the impact of a particular independent variable on the dependent variable [32]. The larger the χ^2 value, the larger the impact that a commit-level metric has on the likelihood of a commit introducing defects. We also calculated the statistical significance ($p-$value) of χ^2. When p is less than a significance level (e.g., 5 percent), we can conclude that the independent variable has a statistically significant impact on the dependent variable. We used the ANOVA Type-II test because it yields a more stable ranking of metrics, as suggested by Tantithamthavorn *et al.* [48]. To more intuitively show the impact of each metric, we calculated the percentage of the χ^2 of each commit-level metric to the sum of all χ^2 values of a model, to rank the metrics by their impact for each model.

Similarly, we used the χ^2 values of the project-level metrics (i.e., random effects in project-aware JIT models and context-aware JIT models) obtained from the likelihood ratio test (LRT), to measure the impact of project-level metrics on the likelihood of a commit introducing defects. We used the likelihood ratio test rather than directly comparing the variance of random effects as suggested by Bolker *et al.* [3], as the variance of a random effect is not reliable when the sampling distribution is skewed. We also divided the p-value by 2 as suggested by Bolker *et al.*, as LRT-based null hypothesis tests are conservative when the null value (i.e., the variance of random effects) is on the boundary of the

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

feasible space (i.e., the variance of random effects cannot be less than 0) [39].

6 Case Study Results

In this section, we present the approach and results with respect to each research question.

RQ1: Does the interpretation of the local JIT models vary?

Approach. To address RQ1, we started from the 20 studied datasets. For each dataset, we constructed a local JIT model using logistic regression (see Section 5.3.1). Because the focus of this RQ is on interpretation, we trained each local JIT model with the whole dataset of the respective project. For each local JIT model, we extracted the coefficients of the intercept and independent variables. Following Section 5.4, we analyzed the goodness-of-fit of local JIT models; and identified the most important metric of defect-introducing commits. Table 4 presents the model statistics of the local JIT models. Below, we discuss the results with respect to (1) the goodness-of-fit; (2) the most important metric; and (3) the baseline likelihood of introducing defects of the local JIT models.

Goodness-of-Fit. To ensure that the interpretation that is derived from the local JIT models is accurate, we first evaluate the R^2 goodness-of-fit of the local JIT models and the predictive accuracy using AUC. We find that the R^2 values of local JIT models range from 0.08 to 0.43, while the AUC values of local JIT models range from 0.63-0.86. Table 4 confirms that our logistic regression can explain the variability of the data from 8 to 43 percent. Despite some projects achieving a low goodness-of-fit (i.e., the linear regression models cannot capture the non-linear relationship of the metrics and the outcome), our local JIT models still outperform random guessing (AUC = 0.5).

*Results. The most important metric of the local JIT models varies among projects.*Table 4 shows the model statistics of the local JIT models. The most important metric (i.e., the metric with the highest percentage of χ^2) is bolded and highlighted in red. We find that Entropy is the most important metric for 11 of 20 (55 percent) studied projects, while the number of subsystems (NS) is the most important metric for 6 of 20 (30 percent) projects. This finding suggests that the interpretation of cross-project JIT models (e.g., global JIT models) that are trained from a merged data may be misleading when project-level variances and the characteristics of the studied projects are not considered.

We suspect that the different interpretations of the local JIT models have to do with the nature of the project characteristics, instead of the goodness-of-fit of the models. We find that local JIT models with similar goodness-of-fit also produce different most important metrics. For example, both the postgres and osquery projects have a very similar goodness-of-fit with an R^2 value of 0.31 and 0.30, respectively. However, the most important metric for the postgres project is the number of files (NF), while the most important metric for the osquery project is the number of subsystem (NS). The inconsistency of the most important metric for local JIT models that share similar goodness-of-fit can also be observed for the camel and django projects, and for the bugzilla and fastjson projects.

The different interpretation of local JIT models is similar to the findings of Menzies *et al.* [34] who observed variations in the most important metric for each project when defect prediction models are trained using different samples from the same project data.

The Baseline Likelihoods of Introducing Defects Varies Among Projects. Table 4 shows that the local JIT models often have different intercepts (i.e., the baseline likelihood of introducing defects). The variance of the distribution of intercepts among local JIT models is 0.21, indicating that different projects have different baseline likelihood of introducing defects (e.g., some projects are more likely to have defect-introducing commits than others). This finding echoes the findings that the importance of the characteristics of the studied projects must be considered when constructing cross-project JIT models.

> *Summary: The most important metric of the local JIT models and the baseline likelihood of introducing defects vary among projects, suggesting that the interpretation of global JIT models that are trained from a merged data may be misleading when project-level variances and the characteristics of the studied projects are not considered.*

RQ2: How consistent is the interpretation of the global JIT model when compared to the local JIT models?

Approach. To address RQ2, we investigated the difference of the ANOVA importance scores of the global JIT model when compared to the local JIT models. We started from merging the 20 studied datasets into a single dataset (called a merged data). We then constructed a global JIT model using logistic regression (see Section 5.3.2). Because the focus of this RQ is on interpretation, we trained the global JIT model with the whole merged dataset. Similar to RQ1, we analyzed the goodness-of-fit of global JIT models; and identified the most important metric of defect-introducing commits. Table 4 presents the model statistics of the global JIT model. Below, we discuss the results with respect to (1) the goodness-of-fit; (2) the most important metric; and (3) the baseline likelihood of introducing defects of the global JIT models.

Goodness-of-Fit. The R^2 value of the global JIT model is 30 percent lower than the average R^2 values of the local JIT models. Table 4 shows that the R^2 values of the global JIT model is 0.14, indicating that the global JIT model can explain the variability of the data by 14 percent. On the other hand, Table 4 shows that the average R^2 values of the local JIT models is 0.207.

Results. The most important metric of defect-introducing commits that is derived from a global JIT model is not always consistent with the local JIT models. Table 4 shows the model statistics of the global JIT model. We find that entropy, the purpose of the commit (FIX), and the number of subsystems (NS) are the first, second, and third most important metric of defect-introducing commits when deriving from the global JIT model. As shown in the results of RQ1, Entropy is the most important metric of defect-introducing commits for only 11 (55 percent) local JIT models. On the other hand, the number of subsystem (NS) is the most important metric of defect-introducing commits for the other 6 (30 percent)

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

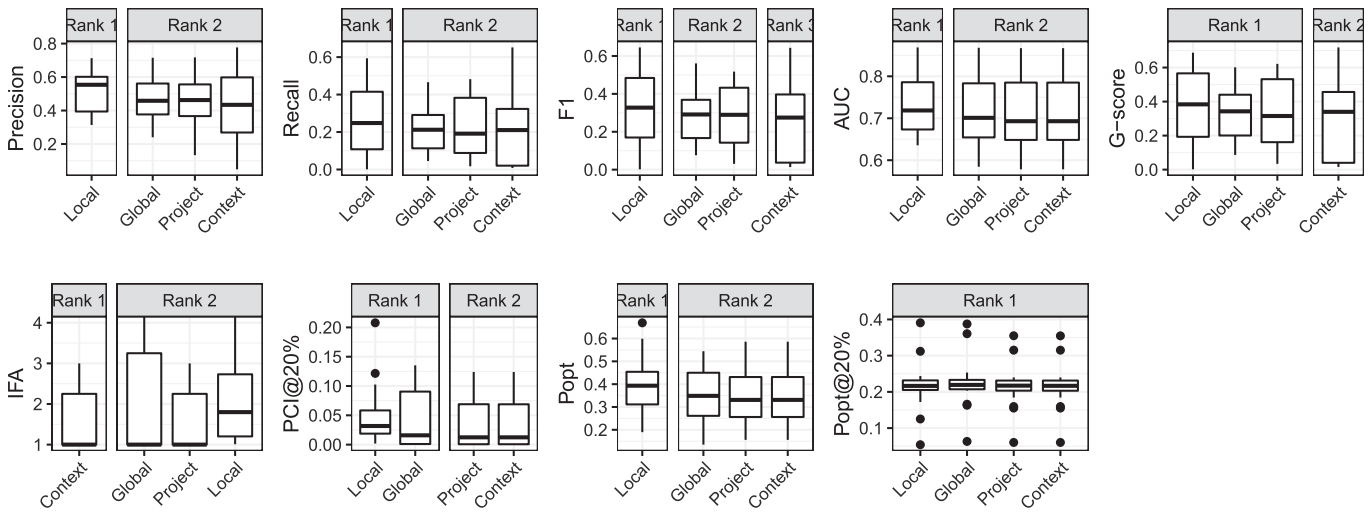

Fig. 4. The ScottKnott ESD ranking and the distributions of the nine performance measures of the global, project-aware, context-aware JIT models. Noted that the performance of local JIT models should be used as a baseline reference, as the local JIT models are not available in the real-world cross-project scenario when having limited access to historical data.

local JIT models—where NS appears at the third rank of the global JIT model.

The global JIT model cannot capture the variation of the baseline likelihood of introducing defects for all of the studied projects. Table 4 shows the intercept values (i.e., the baseline likelihood of introducing defects) of the global JIT model and the local JIT models. We find that the intercept value of the global JIT model is -1.1, while the intercept values of the local JIT models range from -1.95 to -0.29. This finding indicates that the global JIT model cannot capture the variation of the baseline likelihood of introducing defects for all of the studied projects. Such inconsistency of the baseline likelihood of introducing defects between the global JIT model and the local JIT models may produce misleading conclusions when interpreting the global JIT model.

The inconsistency between the interpretation of global and local JIT models is similar to the findings of Menzies *et al.* [33] who observed that in the context of release-level defect prediction, the interpretation of models that are trained from the merging of all projects' data is suboptimal compared to the interpretation of models that are trained from the merging of a cluster of similar projects' data.

Summary: The most important metric of the global JIT model is consistent for 55 percent of the local JIT models, suggesting that the interpretation of a global JIT model cannot capture the variation of the interpretation for all local JIT models. Moreover, the global JIT model cannot capture the variation of the baseline likelihood of introducing defects for all of the studied projects, suggesting that conclusions that are derived from the global JIT model are not generalizable to many of the studied projects.

RQ3: How accurate are the predictions of the project-aware JIT model and the context-aware JIT model when compared to the global JIT model?

Approach. To address RQ3, we evaluate the predictive accuracy of JIT models using nine performance measures (e.g., precision, recall) to investigate whether the predictive accuracy of project-aware and context-aware JIT models are comparable to global JIT models. Similar to prior work [22], we evaluate the JIT models using cross-project evaluation scenario as follows. First, one project is set aside as the testing project. Second, the rest of the studied projects are merged together as one large training dataset. Then, we train a global JIT model, a project-aware JIT model, and a context-aware JIT model (see Section 5.3). Then, we evaluate the models using the nine performance metrics (see Section 5.4.1). We repeat these steps for each of the 20 studied projects. In addition, we also evaluate the local JIT models as a baseline comparison. To do so, we apply an out-of-sample bootstrap model validation technique to estimate the model performance of within-project dataset [52], i.e., for each project, a local JIT model is trained on a bootstrap sample with replacement, and tested on the samples that do not appear in the bootstrap sample. Then, we compute the average performance value from the 100-repeated out-of-sample bootstrap validation.

Finally, we apply a ScottKnott ESD test to cluster the distributions into statistically distinct ranks with non-negligible effect size difference [51], [52], [53]. The Scott-Knott ESD test is designed to overcome the confounding factor of overlapping groups that are produced by other post-hoc tests, such as Nemenyi's test. In particular, Nemenyi's test produces overlapping groups of techniques, implying that there exists no statistically significant difference among the techniques. In contrast, the ScottKnott ESD test produces the ranking of the techniques while ensuring that (1) the magnitude of the difference for all of the distributions in each rank is negligible; and (2) the magnitude of the difference of distributions between ranks is non-negligible. The Scott-Knott ESD test is based on the ANOVA assumptions of the original ScottKnott test (e.g., normal distributions, homogeneous distributions, and the minimum sample size). Fig. 4 presents the ScottKnott ESD ranking and the distributions of the nine performance measures of the global, project-aware, context-aware JIT models, and the local JIT models (as a baseline).

Results. Project-aware and context-aware JIT models achieve comparable performance with global JIT models. The ScottKnott

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

TABLE 5
Model Statistics of the Project-Aware JIT Model (R^2 = 0.26)

Type	Variable	Variance	Coef.	χ^2	Pr($>\chi^2$)
Random slope	Entropy	0.34	-	16856.3	<2.2e-16***
Random intercept	Project	0.65	-	8208.6	<2.2e-16***
Fixed effect	FIX	-	0.53	3284.22	<2.2e-16***
	NS	-	0.27	3245.11	<2.2e-16***
	NF	-	0.05	76.80	<2.2e-16***
	LT	-	0.01	9.10	2.6e-3**
	Relative Churn	-	<0.01	0.15	0.69°

Statistical significance of χ^2:
$^{\circ}p \geq 0.05$; $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$.
The χ^2 value measures the impact of a particular independent variable on the dependent variable. The larger the χ^2 value, the larger the impact that a commit-level metric has on the likelihood of a commit introducing defects.

ESD rankings (see Fig. 4) confirm that the project-aware and context-aware JIT models achieve similar performance (with negligible to small effect size) compared to the global JIT models across all 9 performance metrics. In addition, the ScottKnott ESD rankings (see Fig. 4) also align with the finding in prior studies [5], [22], [55], that cross-project JIT models that are trained on a mixed-project dataset yield comparable performance compared to local JIT models. The comparable performance of project-aware and context-aware JIT models with global JIT models lays the foundation of the comparison among the derived interpretation from the three studied types of JIT models.

Summary: Project-aware JIT models and context-aware JIT models that consider project context achieve comparable performance across nine measured performance metrics with global JIT models that do not consider project context. The comparable performance allows us to further compare the interpretation among global JIT models, project-aware JIT models and context-aware JIT models in the following RQs.

RQ4: How consistent is the interpretation of the project-aware JIT model when compared to the local JIT models and the global JIT model?

Approach. To address RQ4, we started from the merged data in RQ2. Then, we constructed a project-aware JIT model using a mixed-effect logistic regression model (see Section 5.3.3). Because the focus of this RQ is on interpretation, we trained the project-aware JIT model with the whole merged dataset. Similar to RQ1 and RQ2, we analyzed the goodness-of-fit and identified the most important metric of the project-aware JIT model. Table 5 presents the model statistics of the project-aware JIT model. To analyze the consistency of the most important metric, we calculated the errors of the coefficient of the most important metric (i.e., Entropy) between the project-aware JIT model and the local JIT models, as well as between the project-aware JIT model and the global JIT model. Similarly, to analyze the consistency of the baseline likelihood of introducing defects, we calculated the errors of the intercept between the project-aware JIT model and the local JIT models, as well as between the project-aware JIT model and the global JIT model. Below, we discuss the results with respect to (1) the goodness-of-fit; (2) the errors of the important scores; and (3) the errors of the baseline likelihood of introducing defects.

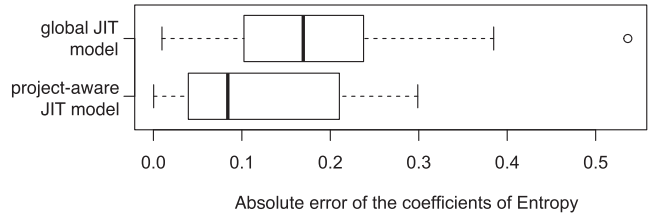

Fig. 5. Distribution of the absolute error of the coefficient of Entropy for the project-aware JIT model and for the global JIT model. The figure shows that the coefficient estimates of the most important metric (i.e., Entropy) that are derived from the project-aware JIT model are 53 percent more accurate than those of the global JIT model.

Goodness-of-Fit. The R^2 value of the project-aware JIT model is 86 and 26 percent higher than that of the global JIT model and the average R^2 values of the local JIT models, respectively. We find that the R^2 value of the project-aware JIT model is 0.26, while the R^2 value of the global JIT model is 0.14 and the average R^2 values of the local JIT models is 0.207. This finding indicates that the project-aware JIT model that considers project-level variances can explain the variability of the data 86 percent better than the global JIT model that is trained from a merged data.

Results. The coefficient estimates of the most important metric (i.e., Entropy) that are derived from the project-aware JIT model are 53 percent more accurate than those of the global JIT model. Fig. 5 shows the distribution of the absolute errors of the coefficient of Entropy for the project-aware JIT model and for the global JIT model, respectively. We observed that the global JIT model produces a median absolute error (MAE) of 0.17, while the project-aware JIT model produces an MAE of 0.08, which is 53 percent lower than the global JIT model. The results show that the project-aware JIT model provides a more accurate interpretation of the relationship between Entropy and the likelihood of introducing defects than that of the global JIT model.

In addition, Fig. 6 shows the variation of the relationship between Entropy and the likelihood of introducing defects for each studied project in the project-aware model. We find that the random slope of Entropy in the mixed-effect model is able to show different relationships between Entropy and the likelihood of introducing defects across projects.

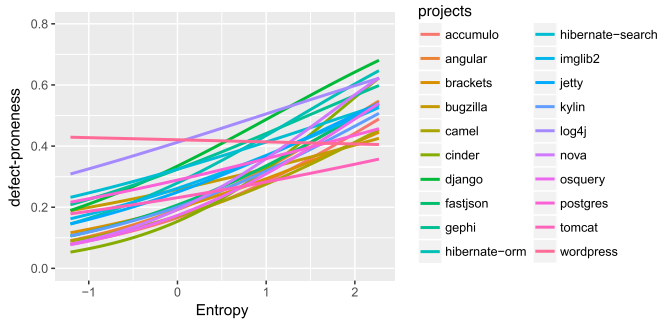

Fig. 6. Relationship between Entropy and defect-proneness for each studied project. Entropy ranges out of [0,1] due to data scaling. The random slope of Entropy in the mixed-effect model is able to show the different relationships between Entropy and the likelihood of a commit introducing defects across projects.

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

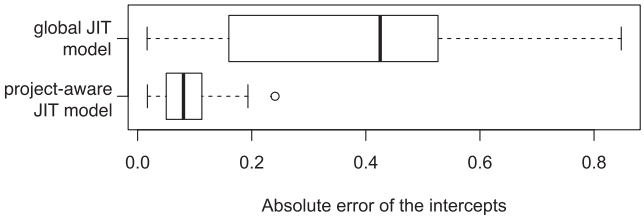

Fig. 7. Distribution of the absolute error of the intercept for the project-aware JIT model and for the global JIT model. The figure shows that the baseline likelihoods of introducing defects that are derived from the project-aware JIT model are 81 percent more accurate than those derived from the global JIT model.

On the other hand, the global JIT model can only provide a fixed estimate of the relationship between Entropy and defect-proneness for all the studied projects, which contradicts to the our prior finding that the most important metric of the local JIT models varies among projects. This finding indicates that *global JIT models must not be used to guide operational decisions.*

The baseline likelihoods of introducing defects that are derived from the project-aware JIT model are 81 percent more accurate than those of the global JIT model. Fig. 7 shows the distribution of the absolute errors of the intercept for the project-aware JIT model and for the global JIT model respectively.

We calculated that the global JIT model has a MAE of 0.43, while the project-aware JIT model has a lower MAE of 0.08. The results show that the project-aware JIT model provides a more accurate interpretation of the baseline likelihood of introducing defects than the global JIT model.

Summary: The coefficient estimates of the most important metric (i.e., Entropy) that are derived from the project-aware JIT model are 53 percent more accurate than those of the global JIT model. In addition, the baseline likelihoods of introducing defects that are derived from the project-aware JIT model are 81 percent more accurate than those of the global JIT model.

RQ5: How consistent is the interpretation of the context-aware JIT model when compared to the local JIT models and the global JIT model?

Approach. To address RQ5, we repeated the approach of RQ4, replacing the project-aware JIT model with a context-aware JIT model using a mixed-effect logistic regression model (see Section 5.3.4) that considers many contextual factors of different projects. Because the focus of this RQ is on interpretation, we trained the context-aware JIT model with the whole merged dataset. Table 6 presents the model statistics of the context-aware JIT model. Similar to RQ4, we analyzed (1) the goodness-of-fit of the context-aware JIT model; (2) the consistency of the most important metric; and (3) the consistency of the baseline likelihood of introducing defects. Below, we discuss the results with respect to (1) the goodness-of-fit; (2) the errors of the important scores; and (3) the errors of the baseline likelihood of introducing defects.

Goodness-of-Fit. The R^2 value of the context-aware JIT model is 100 and 35 percent higher than that of the global JIT model and the average R^2 values of the local JIT models, respectively. We find that the context-aware JIT model obtained a R^2 of 0.28, while the R^2 value of the global JIT model is 0.14 and the average R^2 values of the local JIT models is 0.207. This finding indicates that the context-aware JIT model that considers project-level variances and contextual factors can explain the variability of the data 100 percent better than the global JIT model that is trained from a merged data. In addition, the R^2 of the context-aware JIT model is also 8 percent higher than that of the project-aware JIT model.

TABLE 6
Summary of the Context-Aware JIT Model (R^2 = 0.28)

Type	Variable	Variance	Coef.	χ^2	Pr($>\chi^2$)
Random slope	Entropy	0.34	-	15009.00	$<$2.2e-16***
Random intercept	Language	0.22	-	6076.16	$<$2.2e-16***
	TLOC	0.12	-	1370.79	$<$2.2e-16***
	NFILE	0.09	-	1182.11	$<$2.2e-16***
	NCOMMIT	0.08	-	768.27	$<$2.2e-16***
	NDEV	0.24	-	300.36	$<$2.2e-16***
	Nlanguage	0.02	-	64.81	4.2e-16***
	UI	$<$0.01	-	0.18	0.50°
	Database	$<$0.01	-	0.00	0.50°
	Audience	$<$0.01	-	0.00	0.50°
Fixed effect	NS	-	0.27	3242.08	$<$2.2e-16***
	FIX	-	0.53	3197.47	$<$2.2e-16***
	NF	-	0.05	75.99	$<$2.2e-16***
	LT	-	0.01	9.41	2.2e-3**
	Relative Churn	-	$<$0.01	0.16	0.69°

Statistical significance of χ^2:
$^{\circ}p \geq 0.05$; $^{*}p < 0.05$; $^{**}p < 0.01$; $^{***}p < 0.001$.
The χ^2 value measures the impact of a particular independent variable on the dependent variable. The larger the χ^2 value, the larger the impact that a commit-level metric has on the likelihood of a commit introducing defects.

Results. The coefficient estimates of the most important metric (i.e., Entropy) that are derived from the context-aware JIT model are 44 percent more accurate than those of the global JIT model. Fig. 8 shows the distribution of the error of the coefficients of Entropy for context-aware JIT model and for global JIT model, respectively. We calculated that the context-aware JIT model has a MAE of 0.09, which is 44 percent lower than the MAE for global JIT model (0.16), that are calculated in RQ4. The results show that the context-aware JIT model provides a more accurate interpretation of the relationship between Entropy and the likelihood of a commit introducing defects than the global JIT model.

We also observed that project-aware and context-aware JIT models produce similar coefficient estimates of Entropy, as well as other commit-level metrics. We calculated that the median absolute difference of the estimated coefficients of Entropy from the two models is 0.006. The observation suggests that when taking the contextual factors into consideration, the mixed-effect model can yield stable representation of the relationship between commit-level metrics and the likelihood of a commit introducing defects.

The baseline likelihood of introducing defects that are derived from the context-aware JIT model are 64 percent more accurate than those of the global JIT model. Fig. 9 shows the distribution of the errors of the intercept for the project-aware JIT model and for the global JIT model respectively. We calculated that the context-aware JIT model has a lower MAE of 0.15, which is lower than the MAE for global JIT model (0.42) that are calculated in RQ4. The results show that the

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

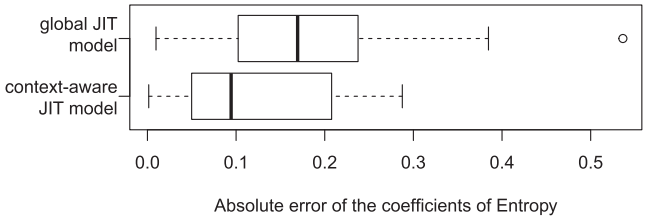

Fig. 8. Distribution of the absolute error of the coefficients of Entropy for the context-aware JIT model and for the global JIT model. The figure shows that the coefficient estimates of the most important metric (i.e., Entropy) that are derived from the context-aware JIT model are 44 percent more accurate than those derived from the global JIT model.

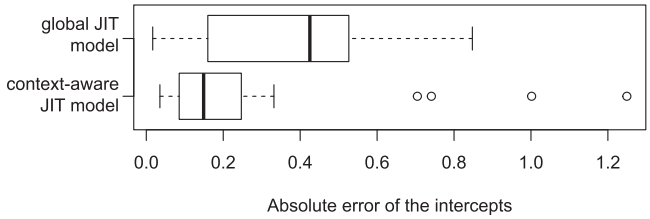

Fig. 9. Distribution of the absolute error of the intercept for the context-aware JIT model and for the global JIT model. The figure shows that the baseline likelihood of introducing defects that are derived from the context-aware JIT model are 64 percent more accurate than those derived from the global JIT model.

context-aware JIT model provides a more accurate interpretation of the baseline likelihood of introducing defects of projects than the global JIT model. We also observed that the MAE for the context-aware JIT model is higher than the project-aware JIT model, indicating that there may be other project factors other than the 9 contextual factors that contributes to the differences among projects.

The main programming language, total lines of code, the number of files, the number of commits, the number of developers, and the number of programming languages are the statistically significant context factors. Table 6 provides a summary of the model statistics for the context-aware JIT model. Table 6 shows that programming languages, total lines of code, the number of files, the number of commits, the number of developers, and the number of programming languages have a p-value < 0.05, indicating that the language, the total lines of code, the number of files, the number of commits, the number of developers, and the number of programming languages have statistically significant impact on the likelihood of introducing defects. For example, projects that are written in Python tend to have the highest likelihood of introducing defects. This finding suggests that the consideration of contextual factors in the JIT model can provide a more in-depth understanding of the properties of defect-introducing commits without impacting the goodness-of-fit of the model.

Summary: The coefficient estimates of the most important metric (i.e., Entropy) that are derived from the context-aware JIT model are 44 percent more accurate than those of the global JIT model. The baseline likelihood of introducing defects that are derived from the context-aware JIT model are 64 percent more accurate than those of the global JIT model. In addition, the consideration of contextual factors in the JIT model can provide a more in-depth understanding of the properties of defect-introducing commits without impacting the goodness-of-fit of the model.

RQ6: Do our prior conclusions hold for other mixed-effect classifiers?

Approach. In the previous RQs, we used logistic regression to construct global JIT models, and mixed-effect logistic regression to construct project-aware JIT models and context-aware JIT models. To investigate if the findings in our prior RQs hold for other mixed-effect classifiers, in this RQ, we evaluate the studied JIT models using random forest classifiers. In particular, we construct global JIT models using random forest classifiers, and construct project-aware and context-aware JIT models using mixed-effect random forest classifiers. Below we describe the modelling techniques in detail.

Random forest is an ensemble classifier that consists of multiple decision trees. Each decision tree in a random forest is trained with a randomly selected subset of the training data and features. A random forest classifies the dependent variable by taking the majority vote of the decision trees. In a classification task (e.g., JIT defect modelling), the ratio of the positive votes from the decision trees in a random forest can be used as the predicted probability of a commit being defect-introducing. In contrast to generalized linear modelling that is used in the prior RQs, random forest models a non-linear relationship between dependent and independent variables, and has been widely used along with generalized linear models in the defect modelling domain [22], [23].

Applying mixed-effect modelling on non-linear models is a relatively new research area. Hajjem *et al.* [7] conducted the initial work on applying mixed-effect modelling on a non-linear model. In 2017, Hajjem *et al.* [8] extended their prior work and proposed generalized mixed-effect regression trees, to allow binary outcomes (i.e., classification). Recently, Wang *et al.* [57] implemented generalized mixed-effect random forest based on Hajjem *et al.*'s work. Recall that in Sections 5.3.3 and 5.3.4, we defined a generalized linear mixed-effect model (mixed-effect logistic regression) as

$$Ln\left(\frac{y}{1-y}\right) = \beta_0 + \beta_i x_i + \sum_{m=1}^{M} v_{jm0} + u_{jk} z_k + \epsilon.$$

In this formula, $\beta_0 + \beta_i x_i$ represents the fixed effects (commit-level metrics), while $\sum_{m=1}^{M} v_{jm0} + u_{jk} z_k$ represents the random effects (project-level metrics). Let $F(x_i) = \beta_0 + \beta_i x_i$, and $G(z_k) = \sum_{m=1}^{M} v_{jm0} + u_{jk} z_k$, we can get the general form of mixed-effect models

$$Ln\left(\frac{y}{1-y}\right) = F(x_i) + G(z_k) + \epsilon. \quad (4)$$

For a generalized linear mixed-effect model, $F(x_i)$ is a linear function. When replacing $F(x_i)$ with a non-linear function (e.g., a random forest model), Formula (4) becomes a generalized non-linear mixed-effect model (e.g., a generalized mixed-effect random forest). Note that the random effect $G(z_k)$ remains a linear function.

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

We followed the same process as stated in RQ3 to calculate the nine performance metrics for the random forest based global, project-aware and context-aware JIT models. In addition, we calculated the goodness-of-fit of the three type of random forest based JIT models using the method stated in Section 5.4.2.

We improved the implementation of generalized mixed-effect random forest from Wang *et al.* [57] using a fast implementation of random forest provided by the `ranger` package in R. In addition, we also improved Wang *et al.*'s implementation's strategy of predicting an unseen project during training using our strategy that is explained in our replication package [29]. The generalized mixed-effect random forest model was trained using the default parameter settings of the random forest implemented in the `ranger` package.

Results. Random forest based project-aware and context-aware JIT models achieve comparable performance to random forest based global JIT model. Table 7 shows the cross-project performance of random forest based global, project-aware, and context-aware JIT models. Similar to what we observed in RQ3, the project-aware and context-aware JIT models that are constructed using generalized mixed-effect random forest yield comparable performance to the random forest based global JIT models.

Random forest based project-aware and context-aware JIT models have a better goodness-of-fit than random forest based global JIT model. The R^2 of the random forest based global JIT model on the merged dataset is 0.29, while the R^2 of the random forest based project-aware JIT model is 0.41, and the R^2 of the random forest based context-aware JIT model is 0.40. The random forest based project-aware and context-aware JIT model achieve a 41 and 38 percent better goodness-of-fit compared to the random forest based global JIT model. The observation is consistent with our findings in RQ4 and RQ5.

Summary: Similar to the project-aware and context-aware JIT models that are based on generalized linear mixed-effect modelling, the project-aware and context-aware JIT model that use generalized mixed-effect random forest modelling also achieve a similar performance compared to random forest based global JIT model. In addition, generalized mixed-effect random forest based project-aware and context-aware JIT model also achieve a better goodness-of-fit compared to the random forest based global JIT model.

TABLE 7
Cross-Project Performance Metrics of Random Forest Based JIT Models

	Global JIT models		Project-aware JIT models		Context-aware JIT models	
	Median	Mean	Median	Mean	Median	Mean
Precision	0.62	0.58	0.58	0.60	0.59	0.63
Recall	0.39	0.42	0.31	0.35	0.38	0.35
F1-score	0.46	0.46	0.38	0.39	0.45	0.35
G-score	0.53	0.54	0.47	0.46	0.53	0.41
AUC	0.72	0.73	0.74	0.75	0.74	0.75
IFA	1.00	1.05	1.00	1.40	1.00	1.45
PCI@20%	0.07	0.08	0.07	0.07	0.07	0.07
P_{opt}	0.45	0.47	0.41	0.41	0.41	0.41
P_{opt}@20%	0.27	0.27	0.27	0.27	0.28	0.27

The table shows that random forest based project-aware and context-aware JIT models achieve comparable performance to random forest based global JIT model.

7 Practical Guidelines

Based on the results of our study, we derive the following practical guideline:

When the goal is to derive sound interpretations from JIT models that are trained from a mixed-project dataset,

1) *The project-aware JIT models using a mixed-effect modelling approach (with only one project-level random intercept) should be used to consider the project factors.*
2) *The context-aware JIT models using a mixed-effect modelling approach (with multiple random intercepts for project-level contextual factors) should be used to consider the variation of the context factors and understand the impact of the contextual factors on the risk of defect-introducing commits without impacting the goodness-of-fit of the models.*

Below, we discuss the implications of our guideline to practitioners and researchers.

Implications to Practitioners. When having limited access to data, the use of mixed-effect modelling approach to train on a mixed-project dataset allows practitioners to produce more accurate insights that are able to capture the different project and context characteristics (e.g., programming languages). Such accurate insights could help developers prioritize pre-commit testing efforts and code review focuses, as well as QA leads and managers to develop the most effective quality improvement plans.

Implications to Researchers. When the goal is to develop an empirically-grounded theory from a mixed-project dataset in order to draw a general conclusion, the mixed-effect modelling approach should be used to allow researchers to gain a deeper understanding whether specific conclusions are sensitive to particular project or context characteristics, or not. Recent studies have employed mixed-effect modelling approaches. For example, Hassan *et al.* [10] employed a mixed-effect modelling approach to examine the relationship between the characteristics of mobile app reviews and the likelihood of a developer responding to a review. Thongtanunam *et al.* [54] employed a mixed-effect modelling approach to examine the relationship between the characteristics of code review dynamics and the likelihood of a patch introducing defects in the context of modern code review. The empirical evidence of our paper is another supporting data point that mixed-effect modelling approaches can capture both project and contextual factors. Hence, we advocate the use of mixed-effect modelling approaches in future studies.

8 Threats to Validity

In this section, we discuss the threats to validity of our study.

Internal Validity. Recently, Rodríguez-Pérez *et al.* [43] raised concerns that there are different variants of the SZZ algorithm. In this paper, we identified defect-introducing commits using CommitGuru [44], which uses the complete

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

SZZ algorithm [46]. The SZZ algorithm is commonly used in prior work on JIT defect prediction [22], [23], [58]. The algorithm first identify defect-fixing commits by matching commits with bug reports labeled as fixed in the issue tracking system, and then employ the "diff" and "blame" functionality of VCS to determine the defect-introducing lines, and locate the defect-introducing commit that modifies the defect-introducing lines. This variant has two common limitations. First, it is possible that some commits may not be linked or incorrectly linked to the issue reports. Second, it is possible that the "diff" and "blame" functionality may involve cosmetic changes, comment changes, or new blank lines. Future research should consider addressing these limitations with an improved variation of the SZZ algorithm.

External Validity. We used 20 open source projects in our study. Hence, our results may not be generalizable to other projects. However, we selected projects that have large amount of commits to combat potential bias in our results. And our study shows that there exists at least one set of projects which merging the datasets would lead to inaccurate interpretation. Nonetheless, additional replication studies are needed to verify our results.

Construct Validity. Although we studied eight popular commit-level metrics from the literature [22], there are many other commit-level metrics that can be included in our study. However, as the goal of our study is to compare the interpretation of different types of cross-project JIT models under the same settings (i.e., same training data and same commit-level metrics), other metrics can be included in future work (e.g., code review metrics [31], test smell metrics [28]).

In addition, we studied a limited number (9) of project-level metrics that were used in prior work to study the context of projects [59], [61]. We would like to note that the selection of the project-level metrics depends on the problem, the context, and the operationalization of the hypotheses. Thus, the goal of our work is not to draw the generalization that Language is always the most important contextual factor of JIT defect models. Instead, one of the main goals of our work is to highlight the benefits of considering project characteristics when constructing JIT models from a mixed-project dataset to investigate the impact of the most contextual factors on the likelihood of a commit being a defect-introducing one.

9 Conclusion

In this paper, we investigated the impact of data merging on the interpretation of cross-project JIT defect models. In particular, we investigated the interpretation of three types of cross-project JIT models (i.e., a global JIT model, a project-aware model, and a context-aware model) in comparison with the interpretation of local JIT models. Through a case study of 20 open source projects, we make the following key observations:

1) The most important metric (i.e., the metric that has the highest impact on the likelihood of a commit introducing defects) of the local JIT models and the baseline likelihood of introducing defects (i.e., the likelihood of introducing defects when all commit-level metrics are 0) vary among projects.
2) The most important metric of the global JIT model is consistent with 55 percent of the studied projects' local JIT models, suggesting that the interpretation of global JIT models cannot capture the variation of the interpretation for all local JIT models.
3) The project-aware JIT model can provide both a more representative interpretation and a better fit to the dataset than the global JIT model.
4) The context-aware JIT model can provide both a more representative interpretation and a better fit to the dataset than the global JIT model.

These findings lead us to draw the following suggestion: *when training a defect model with a pool of mixed project data, one should opt to use a mixed-effect modelling approach that considers individual projects and contexts*.

Finally, we would like to emphasize that the impact of data merging on the interpretation of cross-project JIT defect models does not necessarily apply to all studies, all scenarios, all datasets, and all analytical models in software engineering. Instead, the key message of our study is to shed light that the simple data merging practice impacts the interpretation of cross-project JIT models, as our research shows that there exists a set of projects for which merging the datasets would lead to incorrect conclusions. Thus, researchers and practitioners should consider using mixed-effect modelling. On the other hand, irrespective of learning algorithms, using mixed-effect modelling approach to build JIT models can provide better interpretation by taking project-level variances into consideration, without sacrificing the performance of JIT models. Thus, future studies should consider the mixed-effect modelling approach when the goal is to derive sound interpretation.

Acknowledgments

The work of Chakkrit Tantithamthavorn was supported by Australian Research Council through Discovery Early Career Researcher Award (DECRA) under Grant DE200100941.

References

[1] A. Agresti, *Categorical Data Analysis*. Gainesville, FL, USA: Wiley, 2013.
[2] N. Bettenburg, M. Nagappan, and A. E. Hassan, "Towards improving statistical modeling of software engineering data: Think locally, act globally!" *Empirical Softw. Eng.*, vol. 20, no. 2, pp. 294–335, 2015.
[3] B. M. Bolker *et al.*, "Generalized linear mixed models: A practical guide for ecology and evolution," *Trends Ecol. Evol.*, vol. 24, no. 3, pp. 127–135, 2009.
[4] P. Devanbu, T. Zimmermann, and C. Bird, "Belief & evidence in empirical software engineering," in *Proc. IEEE/ACM Int. Conf. Softw. Eng.*, 2016, pp. 108–119.
[5] T. Fukushima, Y. Kamei, S. McIntosh, K. Yamashita, and N. Ubayashi, "An empirical study of just-in-time defect prediction using cross-project models," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 172–181.
[6] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows," in *Proc. ACM/IEEE 32nd Int. Conf. Softw. Eng.*, 2010, pp. 495–504.
[7] A. Hajjem, F. Bellavance, and D. Larocque, "Mixed-effects random forest for clustered data," *J. Statist. Comput. Simul.*, vol. 84, no. 6, pp. 1313–1328, 2014.
[8] A. Hajjem, D. Larocque, and F. Bellavance, "Generalized mixed effects regression trees," *Statist. Probability Lett.*, vol. 126, pp. 114–118, 2017.

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

[9] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, 2009, pp. 78–88.

[10] S. Hassan, C. Tantithamthavorn, C.-P. Bezemer, and A. E. Hassan, "Studying the dialogue between users and developers of free apps in the Google Play Store," *Empirical Softw. Eng.*, 2018, pp. 1275–1312.

[11] S. Herbold, A. Trautsch, and J. Grabowski, "A comparative study to benchmark cross-project defect prediction approaches," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 811–833, Sep. 2017.

[12] S. Herbold, A. Trautsch, and J. Grabowski, "Global vs. local models for cross-project defect prediction," *Empirical Softw. Eng.*, vol. 22, no. 4, pp. 1866–1902, 2017.

[13] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 2, pp. 111–114, Feb. 2019.

[14] Q. Huang, X. Xia, and D. Lo, "Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2017, pp. 159–170.

[15] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Trans. Softw. Eng.*, early access, 2020, doi: 10.1109/TSE.2020.2982385.

[16] J. Jiarpakdee, C. Tantithamthavorn, and J. Grundy, "Practitioners' perceptions of the goals and visual explanations of defect prediction models," 2021, *arXiv:2102.12007*.

[17] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on the interpretation of defect models," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 320–331, Feb. 2021.

[18] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "Autospearman: Automatically mitigating correlated metrics for interpreting defect models," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2018, pp. 92–103.

[19] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "AutoSpearman: Automatically mitigating correlated software metrics for interpreting defect models," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2018, pp. 92–103.

[20] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "The impact of automated feature selection techniques on the interpretation of defect models,"in *Proc. Empirical Softw. Eng.*, 2020, pp. 3590–3638.

[21] P. C. Johnson, "Extension of Nakagawa & Schielzeth's R2GLMM to random slopes models," *Methods Ecol. Evol.*, vol. 5, no. 9, pp. 944–946, 2014.

[22] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 2072–2106, 2016.

[23] Y. Kamei *et al.*, "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2013.

[24] C. Khanan *et al.*, "JITBot: An explainable just-in-time defect prediction bot," in *Proc. 35th IEEE/ACM Int. Conf. Automat. Softw. Eng.*, 2020, pp. 1336–1339.

[25] S. Kim, E. J. Whitehead Jr., and Y. Zhang, "Classifying software changes: Clean or buggy?," *IEEE Trans. Softw. Eng.*, vol. 34, no. 2, pp. 181–196, Mar./Apr. 2008.

[26] B. A. Kitchenham, E. Mendes, and G. H. Travassos, "Cross versus within-company cost estimation studies: A systematic review," *IEEE Trans. Softw. Eng.*, vol. 33, no. 5, pp. 316–329, May 2007.

[27] R. Krishna and T. Menzies, "Bellwethers: A baseline method for transfer learning," *IEEE Trans. Softw. Eng.*, vol. 45, no. 11, pp. 1081–1105, Nov. 2019.

[28] S. Lambiase, A. Cupito, F. Pecorelli, A. De Lucia, and F. Palomba, "Just-in-time test smell detection and refactoring: The darts project," in *Proc. 28th Int. Conf. Program Comprehension*, 2020, pp. 441–445.

[29] D. Lin, C. Tantithamthavorn, and A. E. Hassan, "Replication package of our paper," 2019. [Online]. Available: https://github.com/SAILResearch/suppmaterial-19-dayi-risk_data_merging_jit

[30] J. N. Mandrekar, "Receiver operating characteristic curve in diagnostic test assessment," *J. Thoracic Oncol.*, vol. 5, no. 9, pp. 1315–1316, 2010.

[31] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 5, pp. 412–428, May 2018.

[32] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the QT, VTK, and ITK projects," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 192–201.

[33] T. Menzies *et al.*, "Local versus global lessons for defect prediction and effort estimation," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 822–834, Jun. 2013.

[34] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[35] A. Mockus and D. M. Weiss, "Predicting risk of software changes," *Bell Labs Tech. J.*, vol. 5, no. 2, pp. 169–180, 2000.

[36] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. 30th Int. Conf. Softw. Eng.*, 2008, pp. 181–190.

[37] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 284–292.

[38] S. Nakagawa and H. Schielzeth, "A general and simple method for obtaining R2 from generalized linear mixed-effects models," *Methods Ecol. Evol.*, vol. 4, no. 2, pp. 133–142, 2013.

[39] J. C. Pinheiro and D. M. Bates, "Mixed-effects models in S and S-plus," New York, NY, USA: Springer, 2000.

[40] C. Pornprasit and C. Tantithamthavorn, "JITLine: A simpler, better, faster, finer-grained just-in-time defect prediction," 2021, *arXiv:2103.07068*.

[41] R. Purushothaman and D. E. Perry, "Toward understanding the rhetoric of small source code changes," *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 511–526, Jun. 2005.

[42] D. Rajapaksha, C. Tantithamthavorn, J. Jiarpakdee, C. Bergmeir, J. Grundy, and W. Buntine, "SQAPlanner: Generating data-informed software quality improvement plans," 2021, *arXiv:2102.09687*.

[43] G. Rodríguez-Pérez, G. Robles, and J. M. González-Barahona, "Reproducibility and credibility in empirical software engineering: A case study based on a systematic literature review of the use of the SZZ algorithm," *Inf. Softw. Technol.*, vol. 99, pp. 164–176, 2018.

[44] C. Rosen, B. Grawi, and E. Shihab, "Commit guru: Analytics and risk prediction of software commits," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 966–969.

[45] E. Shihab, A. E. Hassan, B. Adams, and Z. M. Jiang, "An industrial study on the risk of software changes," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, 2012, pp. 1–11.

[46] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," *ACM SIGSOFT Software Eng. Notes*, vol. 30, no. 4, pp. 1–5, 2005.

[47] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," in *Proc. IEEE 37th Int. Conf. Softw. Eng.*, 2015, pp. 99–108.

[48] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *Proc. 40th Int. Conf. Softw. Eng.: Softw. Eng. Pract.*, 2018, pp. 286–295.

[49] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1200–1219, Nov. 2020.

[50] C. Tantithamthavorn, J. Jiarpakdee, and J. Grundy, "Explainable AI for software engineering," 2020, *arXiv: 2012.01614*.

[51] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proc. Int. Conf. Softw. Eng.*, 2016, pp. 321–332.

[52] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An empirical comparison of model validation techniques for defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 43, no. 1, pp. 1–18, Jan. 2017.

[53] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 45, no. 7, pp. 683–711, Jul. 2019.

[54] P. Thongtanunam and A. E. Hassan, "Review dynamics and their impact on software quality," *IEEE Trans. Softw. Eng.*, early access, 2020, doi: 10.1109/TSE.2020.2964660.

[55] 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 Softw. Eng.*, vol. 14, no. 5, pp. 540–578, 2009.

[56] B. Turhan, A. Tosun, and A. Bener, "Empirical evaluation of mixed-project defect prediction models," in *Proc. 37th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, 2011, pp. 396–403.

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.

[57] J. Wang *et al.*, "Imputing gene expression in uncollected tissues within and beyond GTEX," *Amer. J. Hum. Genet.*, vol. 98, no. 4, pp. 697–708, 2016.
[58] S. Yathish, J. Jiarpakdee, P. Thongtanunam, and C. Tantithamthavorn, "Mining software defects: Should weconsider affected releases?," in *Proc. IEEE/ACM Int. Conf. Softw. Eng.*, 2019, pp. 654–665.
[59] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 182–191.
[60] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Softw. Eng.*, vol. 21, no. 5, pp. 2107–2145, 2016.
[61] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. E. Hassan, "How does context affect the distribution of software maintainability metrics?," in *Proc. 29th IEEE Int. Conf. Softw. Maintenance*, 2013, pp. 350–359.
[62] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf., ACM SIGSOFT Symp. Found. Softw. Eng.*, 2009, pp. 91–100.

Dayi Lin (Member, IEEE) received the PhD degree in computer science from Software Analysis and Intelligence Lab, Queen's University, Canada. He is currently a senior researcher with the Centre for Software Excellence, Huawei, Canada. His research interests include mining software repositories, empirical software engineering, game engineering, and software engineering for machine learning systems.

Chakkrit Tantithamthavorn (Member, IEEE) is currently a 2020 ARC DECRA fellow and a lecturer in software engineering with the Faculty of Information Technology, Monash University, Melbourne, Australia. He has authored or coauthored several top-tier software engineering venues, such as *IEEE Transactions on Software Engineering*, *Springer Journal of Empirical Software Engineering*, and International Conference on Software Engineering. His current research focuses on the development of practical and explainable analytics to prevent future software defects.

Ahmed E. Hassan (Fellow, IEEE) received the PhD degree in computer science from the University of Waterloo. He is currently an ACM SIGSOFT influential educator, an NSERC Steacie fellow, the Canada research chair in software analytics, and the NSERC/BlackBerry software engineering chair with the School of Computing, Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. He spearheaded the creation of the Mining Software Repositories conference and its research community. He is or was on the editorial boards of *IEEE Transactions on Software Engineering*, *Springer Journal of Empirical Software Engineering*, and *PeerJ Computer Science*.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Queen's University. Downloaded on July 08,2023 at 00:01:17 UTC from IEEE Xplore. Restrictions apply.