Towards Explainable Software Defect Prediction Models to Support SQA Planning

Jirayus Jiarpakdee

Supervisors:
Dr. Chakkrit Tantithamthavorn (Main)
Professor John Grundy (Associate)

A thesis submitted for the degree of Doctor of Philosophy at
Monash University in 2021
Faculty of Information and Technology

June 3, 2021

Copyright Notice

Acknowledgements

First and foremost I am extremely grateful to my principal supervisor, Dr. Chakkrit Tantithamthavorn, for his valuable guidance both in life and academia, continuous support, and patience. His assistance has helped me overcoming roadblocks in my research and daily life. I would like to express my sincere gratitude to my co-supervisor, Professor John Grundy, for his insightful feedback, plentiful experience, and strong support. I would like to offer my sincere gratitude to my supervisor when I was with the University of Adelaide, Dr. Christoph Treude, for his valuable advice and unwavering support. I would like to extend my gratitude to all of the committee members, Professor Burak Turhan, Professor Dinh Phung, Associate Professor Ron Steinfeld, and Dr. Li Li, for their immense knowledge and valuable feedback during milestone presentations. I am deeply grateful to all of the collaborators, Professor Ahmed Hassan, Associate Professor Hoa Khanh Dam, Associate Professor Katie Walker, and Dr. Patanamon Thongtanunam, for their feedback and contribution to the publications produced during my PhD study. Finally, I am deeply grateful to my family: my parents, my grandmother, and my brother as well as my partner, Saiparn, for their mental support and unwavering belief in me. Without their understanding and encouragement, it would not be possible for me to complete my study.

This research was supported by the Beacon of Enlightenment PhD Scholarship from the University of Adelaide (November 2017 to January 2019) and the FIT Research and International Postgraduate Research Scholarships from Monash University (February 2019 to May 2021).

Thank you.

Jirayus Jiarpakdee
June 3, 2021

Abstract

Software defects are expensive, but hard to detect and prevent. Thus, Software Quality Assurance (SQA) activities (e.g., code review, software testing, and SQA planning) are applied to ensure the highest quality of software systems by detection and prevention. However, such SQA activities are time-consuming and demanding. Defect prediction models have been proposed to help developers prioritise their limited SQA effort on the most risky files. Yet, the adoption of defect prediction models is still limited due to the following reasons. First, practitioners do not understand why a file is predicted as defective. Second, current defect prediction models still fail to uphold the privacy laws (e.g., GDPR), which requires an explanation of any decision made by an algorithm that affects practitioners. Third, explanations are perceived as equally important as their predictions, but the explainability of defect models still remains largely unexplored. These lead us to formulate the following central research question: *How to increase the explainability of defect prediction models to support SQA planning?* Thus, this thesis hypothesised that: *Explainable defect prediction models are needed to support SQA planning. Empirical studies are the way forward to identify the best explainable defect prediction framework to generate the most reliable explanations.*

To address this hypothesis, we first investigated (1) the impact of correlated metrics on the explanations of defect models; (2) the best automated feature selection techniques to mitigate correlated metrics for generating the explanations of defect prediction models; and (3) the best model-agnostic techniques to explain the predictions of defect prediction models and generate actionable guidance to support SQA planning. Through a series of empirical studies, we derive the following suggestion: To develop an explainable defect prediction model, correlated metrics must be mitigated to derive the most reliable and stable explanations by using our proposed AutoSpearman automated feature selection technique and apply LIME model-agnostic techniques to explain the predictions of defect models and apply rule-based model agnostic techniques to generate actionable guidance on what developers should do and should not do to prevent defects in the future.

Publications during Enrolment

The following publications have been published and are included in this thesis.
2021:

(1) (Full paper CORE A, Acceptance Rate 26%) Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2021). Practitioners' Perceptions of the Goals and Visual Explanations of Defect Prediction Models. In *Proceedings of the International Conference on Mining SoftwareRepositories (MSR).*

2020:

(1) (Journal Impact Factor 6.112) Jiarpakdee, J., Tantithamthavorn, C., Dam, H. K., and Grundy, J. (2020). An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models. *Transactions on Software Engineering (TSE).*

(2) (Journal Impact Factor 3.156) Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2020). The Impact of Automated Feature Selection Techniques on the Interpretation of Defect Models. *Empirical Software Engineering (EMSE).*

2019:

(1) (Journal Impact Factor 6.112) Jiarpakdee, J., Tantithamthavorn, C., and Hassan, A. E. (2019). The Impact of Correlated Metrics on the Interpretation of Defect Models. *Transactions on Software Engineering (TSE).*

(2) (Short paper CORE A*) Jiarpakdee, J. (2019). Towards a More Reliable Interpretation of Defect Models. In *Proceedings of the International Conference on Software Engineering (ICSE): Companion Proceedings*, pages 210–213.

2018:

(1) (Full paper CORE A, Acceptance Rate 26%) Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2018). Autospearman: Automatically Mitigating Correlated Metrics for Interpreting Defect Models. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 92–103.

(2) (Short paper CORE A) Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2018). Artefact: An R Implementation of the AutoSpearman Function. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 711–711.

The following publications are not included in this thesis, but were produced in parallel to the research described in this thesis.

(1) (Online Interactive Book) Tantithamthavorn, C., and Jiarpakdee, J. (2021). Explainable AI for Software Engineering. *Available at http://xai4se.github.io/*.

(2) (Journal Impact Factor 6.112) Rajapaksha, D., Tantithamthavorn, C., Jiarpakdee, J., Bergmeir, C., Grundy, J., and Buntine, W. (2021). SQAPlanner: Generating Data-informed Software Quality Improvement Plans. *Transactions on Software Engineering (TSE)*.

(3) (Journal Impact Factor 5.799) Walker, K., Jiarpakdee, J., Loupis, A., Tantithamthavorn,C., Joe, K., Ben-Meir, M., Akhlaghi, H., Hutton, J., Wang, W., Stephenson, M., Blecher,G., Buntine, P., Sweeny, A., and Turhan, B. (2021). Predicting Ambulance Patient Wait Times: A Multi-Center Derivation and Validation Study. *Annals of Emergency Medicine.*

(4) (Journal Impact Factor 2.589) Jiarpakdee, J., Tantithamthavorn, C., and Grundy, J. (2021). Actionable Analytics: Stop Telling Me What It is; Please Tell Me What To Do!. *IEEE Software*.

(5) (Journal Impact Factor 2.491) Walker, K., Jiarpakdee, J., Loupis,A., Tantithamthavorn, C., Joe, K., Ben-Meir, M., Akhlaghi, H., Hutton, J., Wang,W., Stephenson, M., Blecher, G., Buntine, P., Sweeny, A., and Turhan, B. (2021). Emergency Medicine Patient Wait Time Multivariable Prediction Models: A Multicentre Derivation and Validation Study. *Emergency Medicine Journal*.

(6) (Full paper CORE A*, Acceptance Rate 20.6%) Yatish, S., Jiarpakdee, J., Thongtanunam, P., and Tantithamthavorn, C. (2019). Mining Software Defects: Should We Consider Affected Releases? In *Proceedings of the International Conference on Software Engineering (ICSE)*.

Thesis including Published Works Declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 4 original publications published in peer reviewed venues. The core theme of the thesis is *Towards Explainable Software Defect Analytics Tools to Support SQA Planning*. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Doctor of Philosophy (0190) under the supervision of Dr. Chakkrit Tantithamthavorn (Main supervisor) and Professor John Grundy (Associate supervisor).

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapters 2, 3, 4, and 5, my contribution to the work involved the following:

Thesis Chapter	**Publication Title**	**Status**	**Student contribution**	**Co-author(s) contribution**
2	Practitioners' Perceptions of the Goals and Visual Explanations of Defect Prediction Models	Published at MSR [92]	60%. Concept, design survey, submit ethics approval, conduct a survey study, and write first draft	(1) Dr. Chakkrit Tantithamthavorn: input into manuscript 25%; (2) Professor John Grundy: input into manuscript 15%
3	The Impact of Correlated Metrics on the Interpretation of Defect Models	Published at TSE [85]	60%. Concept, preprocess data, conduct experiments, and write first draft	(1) Dr. Chakkrit Tantithamthavorn: input into manuscript 25%; (2) Professor Ahmed E Hassan: input into manuscript 15%
4	The Impact of Automated Feature Selection Techniques on the Interpretation of Defect Models	Published at ICSME and EMSE [89, 91]	60%. Concept, preprocess data, conduct experiments, and write first draft	(1) Dr. Chakkrit Tantithamthavorn: input into manuscript 25%; (2) Dr. Christoph Treude: input into manuscript 15%
5	An Empirical Study of Model-Agnostic Techniques for Defect Prediction Model	Published at TSE [81]	60%. Concept, preprocess data, conduct experiments, design survey, submit ethics approval, conduct a survey study, and write first draft	(1) Dr. Chakkrit Tantithamthavorn: input into manuscript 20%; (2) Associate Professor Hoa Khan Dam: input into manuscript 10%; (3) Professor John Grundy: input into manuscript 10%

I have renumbered sections of the published publications in order to generate a consistent presentation within the thesis.

Student signature: [redacted] **Date:** 23 November 2021

The undersigned hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Principal supervisor signature: [redacted] **Date:** 23 November 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Nature of Software Defects

Software defects are conditions in software systems that do not follow software requirements or users' expectations. Software defects can arise from coding mistakes, wrong software designs, and OS (operating systems) incompatibilities. These variants of software defects in software systems can have varying impact on the functionality of such software systems. For example, a software system with functional software defects may generate outputs that are different from the software specifications. A software system with security software defects may enable potential malicious attacks.

Due to the wide-spread use of software systems in many industries, software defects can be critically harmful to human health and expensive. For example, as high as 200 deaths per year are caused by unnecessary defects in patient care systems in the UK [215]. The U.S. national annual costs of software defects in software systems are estimated to range from $22.2 to $59.5 billion [169]. Thus, ensuring a high quality of software systems is becoming increasingly important.

1.2 Software Quality Assurance

Software Quality Assurance (SQA) activities (e.g., code review, software testing, and SQA planning) are exercised to routinely check software systems to ensure that the developed software systems meet the requirements and are high quality (defect-free). For example, the source code of software systems must be tested thoroughly before the software releases. Pull requests must be reviewed and approved by members of the development teams before integrating into the main codebase. Figure 1.1 shows a generic process of software quality assurance. The process involves managers and developers. Generally, there are 5 steps:

1. Managers establish a software quality assurance (SQA) plan (e.g., minimum reviewers and minimum test coverage) that developers should follow.

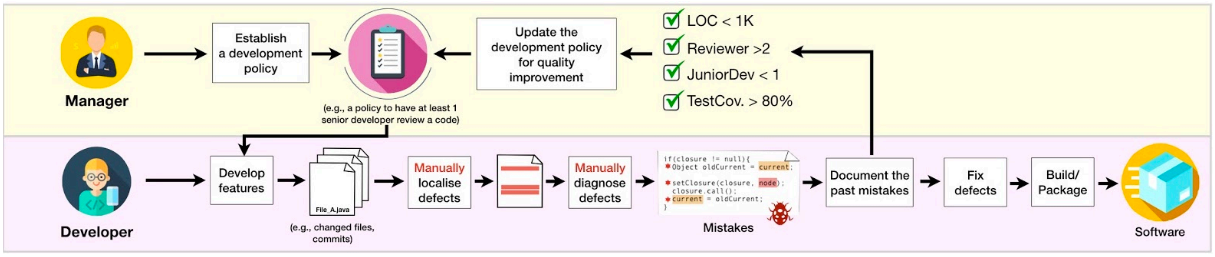

Manager
Establish a development policy
(e.g., a policy to have at least 1 senior developer review a code)
Update the development policy for quality improvement
LOC < 1K
Reviewer >2
JuniorDev < 1
TestCov. > 80%
Developer
Develop features
File_A.java
(e.g., changed files, commits)
Manually localise defects
Manually diagnose defects
Mistakes
Document the past mistakes
Fix defects
Build/ Package
Software

Figure 1.1 A generic process of software quality assurance.

2. Developers implement new features, producing a lot of changed files or commits.
3. Prior to merging to a master branch for release, given thousands of files, developers have to manually localise and diagnose software defects.
4. The documentation of past mistakes will be used to update the SQA plan for quality improvement in future releases.
5. After fixing defects of all features, the software is ready to build and ship to customers.

In this time of fast-paced software development and release cycles, recent software development methodologies embed QA aspects throughout the software development life cycle. Unlike the traditional waterfall development methodology that conducts verification and validation after implementation, Agile testing includes SQA activities into the loop as early as the requirement design and analysis phase. This allows QA engineers to foresee potential issues and develop appropriate measures to address such issues as early as possible to ensure high quality of software systems [149]. Nevertheless, such SQA activities are time-consuming and demanding, especially for large-scale software systems.

To address the increasing effort to ensure the highest quality of large-scale software systems, prior work proposed to use defect prediction models to predict which files [140, 238], methods [71], and lines [223, 228] are likely to be defective in the future. Prior studies developed defect prediction models using a statistical (e.g., regression [17, 153, 238]) or machine learning (e.g., random forests [209]) model for various purposes (e.g., prediction and explanations), which addresses various goals. Figure 1.2 presents an overview process of defect prediction models. To construct defect prediction models, there are four generic steps:

1. select software metrics (e.g., lines of code) that are associated with defect-proneness,
2. construct a defect prediction model using a statistical or machine learning technique (e.g., linear regression and random forests),
3. generate predictions to produce the ranking of the most risky files, and
4. apply model explanation techniques to generate model explanation (i.e., the most important factors that are associated with software defects in the past).

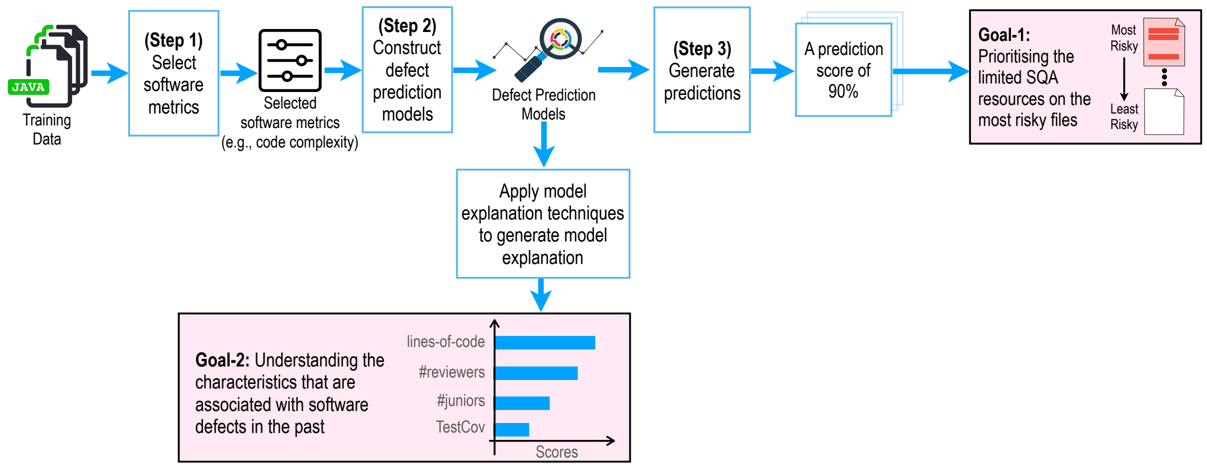

Training Data
JAVA
(Step 1) Select software metrics
Selected software metrics (e.g., code complexity)
(Step 2) Construct defect prediction models
Defect Prediction Models
(Step 3) Generate predictions
A prediction score of 90%
Goal-1: Prioritising the limited SQA resources on the most risky files
Most Risky
Least Risky
Apply model explanation techniques to generate model explanation
Goal-2: Understanding the characteristics that are associated with software defects in the past
lines-of-code
#reviewers
#juniors
TestCov
Scores

Figure 1.2 An overview process of defect prediction models.

Below, we discuss the use of defect prediction models to address various goals.

Use defect prediction models to prioritise the limited SQA resources on the most risky files: Software defects are prevalent in many large-scale software systems (e.g., 47K+ for Eclipse, and 168K+ for Mozilla) [112]. Developers have to exhaustively review and test each file in order to identify and localize software defects. However, given thousands of files in a software system, exhaustively performing SQA activities are likely infeasible due to limited SQA resources (e.g., in a rapid release setting). Thus, prior studies use defect prediction models to predict the likelihood of a file being defective in the future. Such predictions can be used to produce a ranking of the most risky files that require SQA activities. This ranking of the most risky files helps developers save inspection effort and prioritise their SQA resources on the most risky files.

Use defect prediction models to understand the characteristics that are associated with software defects in the past: Numerous characteristics are associated with software quality. For example, the static and dynamic characteristics of source code [65, 238], software development practices (e.g., the amount of added lines) [96, 147], organizational structures (e.g., social networks) [153], and human factors (e.g., the number of software developers, code ownership) [17, 216, 217]. Yet, different systems often have different quality-impacting characteristics. Thus, prior studies used defect prediction models to better understand such characteristics that are associated with software defects in the past. This understanding is used to build empirical theories that are related to software quality to help managers chart appropriate SQA plans. For example, software defects discovered in previous versions of a software project are associated with code complexity. With this understanding derived from defect prediction models, managers can chart SQA plans related to code complexity to ensure high quality of the software system.

1.3 Thesis Motivation

Despite being proposed for decades, the *adoption of defect prediction models in practice* is still very limited due to the following key reasons:

First, practitioners do not understand why a file is predicted as defective. Recent work raises a concern about a lack of explainability of software analytics in software engineering [32]. Practitioners also share similar concerns that analytical models in software engineering must be explainable and actionable [32, 115]. For example, Google [115] argued that defect prediction models should be more actionable to help software engineers debug their programs. Miller [145] also argued that human aspects should be taken into consideration when developing AI/ML-based systems.

Second, current defect prediction models still fail to uphold the privacy laws (e.g., GDPR), which requires an explanation of any decision made by an algorithm that affects practitioners. Article 22 of the European Union's General Data Protection Regulation (GDPR) [179] states that the use of data in decision-making that affects an individual or group *requires an explanation for any decision made by an algorithm*. Yet, current defect prediction models can predict whether a file will likely be defective in the future but cannot explain why models make such a prediction. For example, a model may predict one file to likely be defective with a defective probability value of 80%, while predicting another file to not likely be defective with a defective probability value of 15%. According to GDPR, these predictions need to be better justified, particularly for those predicted as defective, since they may affect the owners of such files.

Third, explanations are perceived as equally important as their predictions, but the explainability of defect models still remains largely unexplored. To better understand the current state of defect prediction research and practitioners' perceptions of defect prediction models, we first conducted a literature analysis and a qualitative survey (Chapter 2). Particularly, we analysed defect prediction studies published in TSE, EMSE, and ICSE during 2015-2020. From this, we identified what are the goals of developing defect prediction models in prior studies. Then, we conducted a qualitative survey on practitioners' perceptions of the goals of developing defect prediction models.

We found that most of the defect prediction studies published at the top SE venues during 2015-2020 (91%) focus on the *prediction* of defect prediction models. Despite receiving little attention from the research community, the survey results show that 82% of the respondents perceived that the *explanation* of defect prediction models are useful. These findings highlight the need to explore the explainability of defect prediction models to support SQA planning.

Based on the 3 aforementioned reasons, we formulated the following overarching research question:

Overarching Research Question (RQ)*–*How can we increase the explainability of defect prediction models to better support SQA planning?

1.4 Thesis Statement and Research Questions

The goal of this thesis is to increase the explainability of defect prediction models to support SQA planning. To address this goal, we formulate the following thesis statement:

Explainable defect prediction models are needed to support SQA planning. Empirical studies are the way forward to identify the best explainable defect prediction framework to generate the most reliable explanations.

However, little is known about the best defect prediction framework to generate the most reliable explanations. Thus, we formulate the following 3 central research questions for this work:

Motivation for RQ1 – The explanation of defect prediction models heavily relies on the studied software metrics used to construct them. However, software metrics are often correlated [54, 72, 73, 86, 207, 211, 235]. For example, Herraiz *et al.* [73], and Gil *et al.* [54] point out that code complexity (`CC`) is often correlated with code size (`size`). Zhang *et al.* [235] point out that many metric aggregation schemes (e.g., averaging or summing of McCabe's cyclomatic complexity values at the function level to derive file-level metrics) often produce correlated metrics.

Recent studies raised concerns that correlated metrics may impact the interpretation of defect models [211, 235]. This is a critical concern since such explanations of defect prediction models constructed using correlated metrics may be used to chart misleading SQA plans. Unfortunately, a literature survey by Emad [193] found that 63% of defect prediction studies that are published during 2000-2011 do not consider correlation analysis prior to constructing defect prediction models. Thus, we formulate the following research question:

(RQ1) How do correlated metrics impact the explanation of defect prediction models?

Motivation for RQ2 – Feature selection techniques are applied in defect prediction studies to select an optimal subset of software metrics that are relevant to defect-proneness [5, 34, 40, 97, 140, 160, 198]. However, none of the prior studies investigated whether such feature selection techniques mitigate correlated metrics for generating the explanation of defect prediction models. Thus, we formulate the following research question:

(RQ2) Which feature selection techniques should be used to mitigate correlated metrics for generating the most reliable explanation of defect prediction models?

Motivation for RQ3 – Commonly-used model explanation techniques (e.g., ANOVA analysis and variable importance) used by prior work [17, 56, 96, 147, 153, 177, 216, 217, 238] to explain defect prediction models only explain the relationship at the project level. However, these model explanations cannot explain the underlying reasoning behind each prediction.

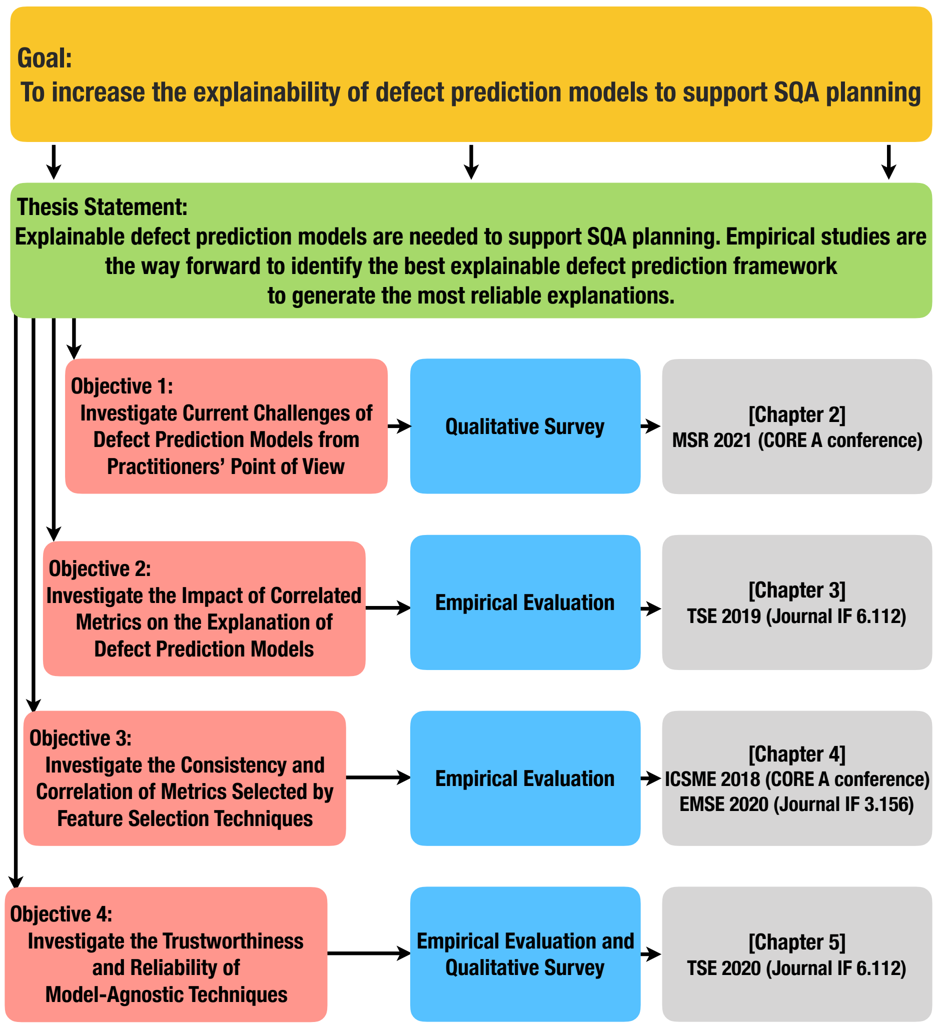

Goal:
To increase the explainability of defect prediction models to support SQA planning
Thesis Statement:
Explainable defect prediction models are needed to support SQA planning. Empirical studies are the way forward to identify the best explainable defect prediction framework to generate the most reliable explanations.
Objective 1: Investigate Current Challenges of Defect Prediction Models from Practitioners' Point of View
Qualitative Survey
[Chapter 2] MSR 2021 (CORE A conference)
Objective 2: Investigate the Impact of Correlated Metrics on the Explanation of Defect Prediction Models
Empirical Evaluation
[Chapter 3] TSE 2019 (Journal IF 6.112)
Objective 3: Investigate the Consistency and Correlation of Metrics Selected by Feature Selection Techniques
Empirical Evaluation
[Chapter 4] ICSME 2018 (CORE A conference) EMSE 2020 (Journal IF 3.156)
Objective 4: Investigate the Trustworthiness and Reliability of Model-Agnostic Techniques
Empirical Evaluation and Qualitative Survey
[Chapter 5] TSE 2020 (Journal IF 6.112)

Figure 1.3 An overview of thesis structure.

Recent work (e.g., LIME [182] and BreakDown [201]) introduced model-agnostic techniques that explain an individual prediction by diagnosing a prediction model. Yet, no defect prediction studies to date have investigated whether such model-agnostic techniques should be used to explain the predictions of defect prediction models. Thus, we formulate the following research question:

(RQ3) Should model-agnostic techniques be used to explain the predictions of defect prediction models?

1.5 Thesis Overview

To address these central research questions, we structure this thesis using the following structure (Figure 1.3). We describe each chapter below.

[Chapter 2] Current Challenges of Defect Prediction Models from the Practitioners' Point of View

To situate the thesis, in this chapter, we first conducted a literature analysis to understand the current state of research in defect prediction studies. Then, we conducted a qualitative survey to investigate practitioners' perceptions of defect prediction models. We asked respondents to assess the perceived usefulness, their willingness to adopt, and the challenges of defect prediction models developed from current practice. The results of the literature analysis show that most recent defect prediction studies (91%) focus on the prediction of defect prediction models. Little research has been done on understanding defect prediction models and their predictions to support SQA planning. Despite receiving little attention from the research community, the results of the qualitative survey show that 82%-84% of the respondents perceived understanding defect prediction models and their predictions as useful and 74%-78% of the respondents are willing to adopt them. These findings motivated us to further explore the explanation of defect prediction models to support SQA planning.

[Chapter 3] The Impact of Correlated Metrics on the Explanation of Defect Prediction Models

In Chapter 2 work we found that there is a need to investigate the explanation of defect prediction models to support SQA planning. Towards eXplainable Defect Prediction models, we first started with software metrics that are used to construct defect prediction models. Prior studies showed that software metrics are often correlated [54, 72, 73, 86, 207, 211, 235]. However, little is known whether such correlated metrics impact the explanation of defect prediction models. Thus, in this chapter, we investigated the impact of correlated metrics on the explanation of defect prediction models. Particularly, we investigated (1) the prevalence of correlated metrics in defect datasets, (2) the impact of the number of correlated metrics on the explanation of defect prediction models, (3) the impact of the ordering of correlated metrics in a model specification on the explanation of defect prediction models, (4) the impact of correlated metrics on the consistency and the level of discrepancy of the explanation of defect prediction models, (5) the consistency of the explanation of defect prediction models after removing correlated metrics, and (6) the impact on the model performance and stability after removing correlated metrics. We found that correlated metrics impact the consistency, the level of discrepancy, and the direction of the importance ranking of metrics, especially for ANOVA techniques. On the other hand, we found that removing all correlated metrics improves the consistency of the produced importance ranking of metrics among the studied model explanation techniques while impacting the model performance by less than 5 percentage points. These findings suggested that correlated metrics must be mitigated when the goal is to derive sound explanation from defect prediction models.

[Chapter 4] Automated Feature Selection Techniques to Mitigate Correlated Metrics for Generating the Explanation of Defect Prediction Models

Chapter 3 work found that correlated metrics must be mitigated to derive sound explanation from defect prediction models. Prior defect prediction studies [5, 34, 40, 97, 140, 160, 198] applied feature selection techniques to find an optimal subset of software metrics that are relevant to defect-proneness. However, little is known whether these feature selection techniques mitigate strong correlation among software metrics. Therefore, in this chapter, we investigated the consistency and correlation of subsets of metrics produced by feature selection techniques. Particularly, we investigated (1) the consistency of the produced subsets of metrics, (2) the correlation of the produced subset of metrics, (3) the performance, (4) the computational cost, and (5) the impact on the explanation of defect prediction models. We found that the subsets of metrics produced by commonly-used feature selection metrics are inconsistent and correlated. To address this concern, we proposed an automated feature selection technique, *AutoSpearman*, that mitigates correlated metrics better and produce more consistent subsets of metrics than other commonly-used feature selection techniques. These findings suggested that *AutoSpearman* should be used in future studies which aim at ensuring high consistency and the automated mitigation of correlated metrics. Furthermore, this work also opened up new research avenues in the automated selection of features for defect prediction models to optimise for explainability.

[Chapter 5] Model-agnostic Techniques to Explain the Predictions of Defect Prediction Models

Chapter 4 proposed *AutoSpearman* that automatically mitigates correlated metrics that are shown in Chapter 3 to have an impact on the explanation of defect prediction models. Nevertheless, commonly-used model explanation techniques (e.g., ANOVA analysis and variable importance) used by prior work [17, 56, 96, 147, 153, 177, 216, 217, 238] to explain defect prediction models only explain the relationship at the project level. These model explanations cannot explain the underlying reasoning behind each prediction. Recent work (e.g., LIME [182] and BreakDown [201]) introduced model-agnostic techniques that explain an individual prediction by diagnosing a prediction model. Yet, no defect prediction studies to date have investigated whether such model-agnostic techniques should be used to explain the predictions of defect prediction models. Thus, in this chapter, we investigated whether model-agnostic techniques should be used to explain the predictions of defect prediction models. To do so, we investigated (1) the variation of explanations generated by model-agnostic techniques, (2) the trustworthiness, (3) reliability, and (4) practicality of model-agnostic techniques. We also conducted a post-evaluation as a qualitative survey to investigate practitioners' perceptions of the explanations generated by model-agnostic techniques. We found that model-agnostic techniques are needed to explain individual prediction of defect prediction models. We also found that more than half of the respondents perceived such explanations as necessary (55%) and useful (65%). Since the implementation of the studied model-agnostic techniques (i.e., LIME and BreakDown) are available in both Python and R, we recommend model-agnostic techniques be used to explain the predictions of defect models.

1.6 Key Contributions

The main contributions of this thesis are as follows:

(1) We conducted a qualitative survey on practitioners' perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction models (Chapter 2).

(2) We investigated the key factors that impact practitioners' perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction model (Chapter 2).

(3) We identified a key set of implications for researchers including open questions for future research on designing and developing the next-generation defect prediction models (Chapter 2).

(4) We investigated the prevalence of correlated metrics in publicly-available defect datasets (Chapter 3).

(5) We investigated the impact of the number and the ordering of correlated metrics on the explanation of defect models (Chapter 3).

(6) We investigated the impact of correlated metrics on the consistency and the level of discrepancy of the produced rankings by the model explanation techniques (Chapter 3).

(7) We investigated the consistency of the importance ranking of metrics after removing all correlated metrics (Chapter 3).

(8) We investigated the impact of removing all correlated metrics on the performance and stability of defect prediction models (Chapter 3).

(9) We investigated the consistency and correlation of subsets of metrics produced by feature selection techniques (Chapter 4).

(10) We investigated the impact of feature selection techniques on the performance of defect prediction models (Chapter 4).

(11) We investigated the computational cost and the correlation threshold values of feature selection techniques (Chapter 4).

(12) We described an introduction to the explainability in software engineering from a perspective of psychological science (Chapter 5).

(13) We described an introduction to model-agnostic techniques (i.e., BreakDown and LIME) for explaining the predictions of defect models (Chapter 5).

(14) We investigated the trustworthiness, reliability, and practicality of model-agnostic techniques (Chapter 5).

Chapter 2

Practitioners' Perceptions of Explainable Defect Models

An earlier version of the work in this chapter appears in the International Conference on Mining Software Repositories (MSR) [92].

2.1 Introduction

Software defects are prevalent, but hard to predict [140] and to prevent [153, 154]. Thus, prior studies developed defect prediction models from historical software data using a statistical or machine learning model for various purposes (e.g., prediction and explanations), which addresses various goals. First is to *predict* the likelihood of a file being defective in the future [65, 140]. Second is to *understand* the characteristics that are associated with software defects in the past [153, 154]. Third is to *explain* their prediction about why a particular file is predicted as defective [81, 109, 165, 166].

Prior studies hypothesized that the predictions could help practitioners prioritize the limited inspection effort on the most risky files [71, 140, 223, 228, 238], while the insights derived from defect prediction models could help managers chart appropriate quality improvement plans [153, 154, 199]. Recently, Wan *et al.* [222] conducted a survey study with practitioners to investigate their perceptions of defect prediction models. However, Wan *et al.* [222] only focused on the prediction goal, while the other two goals (i.e., understanding models and explaining the predictions) have not been investigated. A better understanding of the practitioners' perceptions will help the research community to better understand practitioners' needs, allowing researchers to orient appropriate efforts for the design and the development of the next generation of defect prediction models.

Prior studies used various model-agnostic techniques–techniques that generate explanations of defect prediction models and their predictions–(e.g., ANOVA and LIME) to generate visual explanations to help practitioners understand defect prediction models and their predictions [17, 81, 153, 216, 217]. In fact, different model-agnostic techniques generate different key information, e.g., importance scores and relationship. However, none of the prior studies investigates which model-agnostic techniques are considered as the most preferred by practitioners to generate visual explanations. A better understanding of practitioners' perceptions of the visual explanations of defect prediction models is needed to guide researchers to devise novel visualization techniques that suit practitioners' needs.

In this study, we conducted a qualitative survey to understand the practitioners' perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction models. The analysis of related work led us to focus on three goals of defect prediction models: (1) prioritizing the limited SQA resources on the most risky files; (2) understanding the characteristics that are associated with software defects in the past; and (3) understanding the most important characteristics that contributed to a prediction of a file. We asked respondents to assess the perceived usefulness and their willingness to adopt defect prediction models. Then, we asked the respondents to describe the positive and negative impacts if these defect prediction models were adopted.

Guided by the analysis of related work, we focused on 8 model-agnostic techniques used for generating visual explanations – ANOVA, Variable Importance, Partial Dependence Plots, Decision Tree, LIME [182], BreakDown [55], SHAP [125], and Anchor [183]. We asked our survey respondents to assess each visual explanation generated by these techniques along three dimensions, i.e., information usefulness, information insightfulness, and information quality. We then asked the respondents to describe the positive and negative feedback of each visual explanation of defect prediction models. Through a qualitative analysis of open-ended and closed-ended questions of 50 software practitioners we addressed the following research questions:

(RQ1) Which goals of defect prediction models that practitioners considered the most useful?
82%-84% of the respondents perceived that the three goals of defect prediction models are useful and 74%-78% of them are willing to adopt them. This was especially true for respondents who use Java, have little years of experience, and work in a large team size (more than 100 people). This finding highlights that not only defect predictions but also the other two goals (i.e., understanding defect prediction models and their predictions) receive similar perceptions of usefulness and willingness to adopt with no statistically significant difference.

(RQ2) Which model-agnostic techniques are the most preferred by practitioners to understand defect prediction models and their predictions?
LIME is the most preferred technique for understanding the most important characteristics that contributed to a prediction of a file with an agreement percentage of 66%-78% along three dimensions. ANOVA/VarImp is the second most preferred technique for understanding the characteristics that are associated with software defects in the past with an agreement percentage of 58%-70% along three dimensions.

Based on these findings, future research (1) should put more effort into investigating how to improve the understanding of defect prediction models and their predictions; and (2) should adopt ANOVA/VarImp and LIME model-agnostic techniques to understand defect prediction models and their predictions. We also discuss key lessons learned and open questions for developing the next-generation of defect prediction models, e.g., how to develop the highly scalable human-in-the-loop defect prediction models at the lowest implementation cost, while maintaining its explainability.

The main contributions of this study are as follows:

- We conducted a qualitative survey on practitioners' perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction models. We also provided a detailed replication package as a Jupyter notebook in the online supplementary materials [84].
- We investigated the key factors that impact practitioners' perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction models.

- We identified a key set of implications for researchers including open questions for future research on designing and developing the next-generation defect prediction models.

The body of knowledge provided by the findings of this survey will allow the research community to prioritise the practitioners' needs and to orient future efforts for the design and development of better explainable defect prediction.

Chapter Organisation. Section 2.2 summarizes key related work to identify (1) the key goals of developing defect prediction models, and (2) the model-agnostic techniques used to generate visual explanations; and motivate the research questions based on the analysis of related work. Section 2.3 describes the design of this study. Section 2.4 presents the results of RQ1 and RQ2. Section 2.5 discusses threats to the validity. Finally, Section 2.6 concludes the chapter.

2.2 Related Work & Research Questions

We first summarize key related work to identify (i) the key goals of developing defect prediction models, and (ii) the model-agnostic techniques that are used to generate visual explanations. We then motivate our research questions based on the analysis of related work.

2.2.1 Related Work

In identifying defect prediction studies used in this study, we conduct the following selection steps. First, We collected the titles of full research track publications that were published in the top SE venues (i.e., TSE, ICSE, EMSE, FSE, and MSR) during 2015-2020 from IEEE Xplore, Springer, and ACM Digital Library (as of 11 January 2021). These venues are premier publication venues in the software engineering research community. Second, we used the “defect", “fault", “bug", “predict", and “quality" keywords to search for papers about defect prediction models. This led us to a collection of 2,890 studies. Third, since studies may use several keywords that match with our search queries and appear consistently across the search results, we first identified and excluded duplicate studies. We found that 1,485 studies are duplicated and thus are excluded (1,405 unique studies). Fourth, we manually read the titles and abstracts of these papers to identify whether they are related to defect prediction models. For each paper, we manually downloaded each paper as a pdf file from IEEE Xplore, Springer, and ACM Digital Library. We identified 131 studies that are relevant to software defects prediction. Fifth, of the 131 studies, we excluded 7 studies that are not primary studies of defect prediction models (e.g., secondary studies [78, 240]). Sixth, we excluded 28 studies that are not full-paper and peer-reviewed defect prediction studies (i.e., short, journal first, extended abstract papers). Finally, we selected a total of 96 primary full-paper and peer-reviewed defect prediction studies.

Table 2.1 A summary of type, granularity, and key information of the model-agnostics techniques that are used in defect prediction studies [57, 82], and an example of positive and negative feedback from practitioners (RQ2).

Goal	Technique	Types/Granularity	Information	Positive feedback	Negative feedback
(Goal 2) Understanding the characteristics that are associated with software defects in the past	**Anova/ VarImp**	A Post-hoc Explainer for Model Explanation	(1) The importance scores of each feature	R50 (*"Explains risk values of several factors."*)	R34 (*"It is a very basic plot which could even have been a table."*)
	Partial Dependence Plot (PDP)	A Post-hoc Explainer for Model Explanation	(1) The relationship of each feature on the outcome	R5 (*"I like that I can see a trending pattern to read with ease."*)	R27 (*"Easy to visualize but hard to process the info."*)
	Decision Tree	An Interpretable Model for Model Explanation	(1) The decision rule of each feature	R47 (*"Very analytical, strong with engineers, flow chart."*)	R34 (*"Very difficult to read especially when there are multiple parameters and the tree gets un-manageable."*)
(Goal 3) Understanding the most important characteristics that contributed to a prediction of a file	**LIME**	A Post-hoc Explainer for Outcome Explanation	(1) The prediction of the local models (2) The decision point of each feature (3) The supporting scores of each feature (4) The contradicting scores of each feature (5) The actual feature value of that instance	*R1 ("Risk scores are visually oriented and users will understand it faster.") R50 ("Easy to understand, we just want to know what's already ok and what's need to improve.")*	*R9 ("... But it also could be time consuming to review depending on if there are many different files to review.") R50 ("... too large and too tired to read 10 charts for 10 files.")*
	SHAP	A Post-hoc Explainer for Outcome Explanation	(1) The prediction of the local models (2) % of contribution for each feature to the final probability (3) The actual feature value of that instance	*R5 ("... It does highlight in the explanation of what everything is, which is nice. It gives insightful data on what to look out for.") R14 ("It describes the involvement of each attribute more clearly.")*	*R9 ("This has a lot of information which can be quite useful but lacks a clean readability. It can be quite time-consuming to read this graph.") R27 ("Takes a bit to figure out what is going on.")*
	BreakDown	A Post-hoc Explainer for Outcome Explanation	(1) The prediction of the local models (2) % of contribution for each feature to the final probability (3) The actual feature value of that instance		
	Anchor	A Post-hoc Explainer for Outcome Explanation	(1) The decision rule of each feature	R50 (*"Exactly what I need, just small and short information on what to improve."*)	R36 (*"... doesn't provide a visual aid to put the numbers into perspective."*)

We read each of them to identify their goals of developing defect prediction models and identify the model-agnostic techniques that are used to generate visual explanations. We then further group the goals of developing defect prediction models using *the Open Card Sorting approach*. First, we list all the goals of developing defect prediction models and categorize these goals based on how such models are used in each study. Then, we discuss the inconsistency among the authors to reach the final set of categories. Based on the selected 96 studies, we identify the 3 goals of developing defect prediction models and 8 model-agnostic techniques used to generate visual explanations. Guided by the Guidotti *et al.*'s Taxonomy [57], we classify each technique according to types of model-agnostic techniques (i.e., interpretable models vs. post-hoc explanations) and the granularity levels of explanations (i.e., model explanation and outcome explanation [82]).

Goal 1—Prioritizing the limited SQA resources on the most risky files

Software defects are prevalent in many large-scale software systems (e.g., 47K+ for Eclipse, and 168K+ for Mozilla) [112]. Developers have to exhaustively review and test each file in order to identify and localize software defects. However, given thousands of files in a software system, exhaustively performing SQA activities are likely infeasible due to limited SQA resources (e.g., in a rapid release setting). Thus, prior studies use defect prediction models to predict the likelihood of a file being defective in the future. Such predictions can be used to produce a ranking of the most risky files that require SQA activities. Prior studies leveraged several Machine Learning approaches to develop defect prediction models to predict which files [140, 238], methods [71], lines [228] are likely to be defective in the future. For example, regression models [17, 153, 216, 217, 238], random forests [209], and deep learning [225].

Goal 2—Understanding the characteristics that are associated with software defects in the past

Numerous characteristics are associated with software quality. For example, the static and dynamic characteristics of source code [65, 238], software development practices (e.g., the amount of added lines) [96, 147], organizational structures (e.g., social networks) [153], and human factors (e.g., the number of software developers, code ownership) [17, 216, 217]. Yet, different systems often have different quality-impacting characteristics. Thus, prior studies used defect prediction models to better understand such characteristics that are associated with software defects in the past. This understanding could help managers chart appropriate quality improvement plans. Below, we summarize the four model-agnostic techniques that are used in prior studies to understand the characteristics that are associated with software defects in the past.

Analysis of Variance (ANOVA) is a model-agnostic technique for regression analysis to generate the importance scores of factors that are associated with software defects. ANOVA measures the importance of features by calculating the improvement of the Residual Sum of Squares (RSS) made when sequentially adding a feature to the model. *Variable Importance (VarImp)* is a model-agnostic technique for random forests classification techniques to generate the importance scores of factors that are associated with software defects. The VarImp technique measures the importance of features by measuring the errors made when the values of such features are randomly permuted (i.e., permutation importance). Random forests also provides other variants of importance score calculations (e.g., Gini importance). In this study, we choose the permutation importance technique to generate an example of VarImp visual explanation since we find that permutation importance is more robust to the collinearity issues [85]. We note that the ANOVA and VarImp plots only indicate the importance of each feature, not the directions of the relationship of each feature i.e., positive or negative.

Partial Dependence Plot (PDP) [48] is a model-agnostic technique to generate model explanations for any classification models. Unlike visual explanations generated by ANOVA and VarImp that show only the importance of all features, visual explanations generated by PDP illustrate the marginal effect that one or two features have on the predicted outcome of a classification model.

Decision Tree or Decision Rule is a technique to generate tree-based model explanations. A decision tree is constructed in a top-down direction from a root node. Then, a decision tree partitions the data into subsets of similar instances (homogeneous). Typically, an entropy or an information gain score are used to calculate the homogeneity among instances. Finally, the constructed decision tree can be converted into a set of if-then-else decision rules.

Goal 3—Understanding the most important characteristics that contributed to a prediction of a file

In my previous work, we *et al.* [82] argued that a lack of explainability of defect prediction models could hinder the adoption of defect prediction models in practice (i.e., developers do not understand why a file is predicted as defective). To address this challenge, we proposed to use model-agnostic techniques to generate explanations of the predictions of defect prediction models (i.e., what are the most important characteristics that contributed to a prediction of a file?). Below, we summarize the four state-of-the-art model-agnostic techniques that were used in prior studies to understand the most important characteristics that contributed to a prediction of a file (i.e., LIME, BreakDown, SHAP, and Anchor).

Local Interpretability Model-agnostic Explanations (LIME) [182] is a model-agnostic technique to generate the importance score of the decision rule of each factor for any classification models. The decision rule of each factor is discretized based on a decision tree. LIME aims to generate supporting and contradicting scores which indicate the positive and negative importance of each feature for an instance. For example, a LIME explanation for the LOC feature with an importance score of 40% and a decision rule LOC > 100 => BUG indicates that the condition of the file size that is larger than 100 LOCs would have 40% contribution to the prediction that a file is defective.

BreakDown [55] is a model-agnostic technique for generating probability-based explanations for each model prediction [201]. BreakDown uses the greedy strategy to sequentially measure the contributions of each feature towards the expected predictions. For example, a BreakDown explanation for the LOC feature with an importance score of 40% indicates that the actual feature value of 200 LOCs of the file would have 40% contributions to the final prediction of this particular file as being defective.

SHapley Additive exPlanations (SHAP) [125] is a model-agnostic technique for generating probability-based explanations for each model prediction based on a game theory approach. SHAP uses game theory to calculate the Shapley values (contributions) of each feature based on the decision-making process of prediction models.

Anchor [183] is an extension of LIME [182] that uses decision rules to generate rule-based explanations for each model prediction. The key idea of Anchor is to select if-then rules – so-called anchors – that have high confidence, in a way that features that are not included in the rules do not affect the prediction outcome if their feature values are changed. In particular, Anchor selects only rules with a minimum confidence of 95%, and then selects the rule with the highest coverage if multiple rules have the same confidence value.

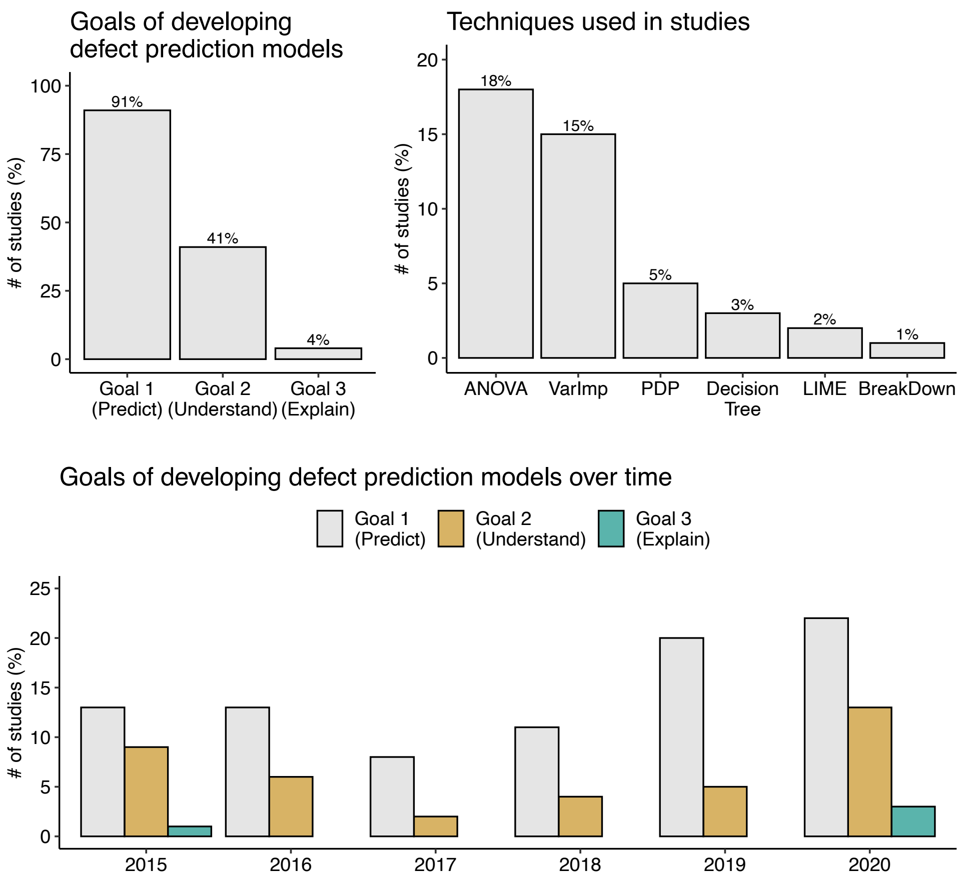


Figure 2.1 The proportion of the goals of developing defect prediction models and the proportion of the model-agnostic techniques used in prior studies. We note that the summation of these percentage values does not add up to 100% since a study may have multiple goals and may use multiple model-agnostic techniques.

2.2.2 Research Questions

As shown in Figure 2.1, we found that most recent defect prediction studies focus on prioritizing the limited SQA resources. This led us to hypothesize that the prediction goal is perceived as more useful than the other two goals, i.e., understanding defect prediction models and their predictions. However, it remains unclear how practitioners perceive the three goals of defect prediction models. Thus, we formulate the following research question:

(RQ1) Which goals of defect prediction models that practitioners considered the most useful?

According to our analysis of related work, prior defect prediction studies also used model-agnostic techniques to generate visual explanations to help practitioners understand (1) the most important characteristics that are associated with software defects in the past; and (2) the most important characteristics that contributed to a prediction of a file. Surprisingly, there exist numerous model-agnostic techniques to generate visual explanations (e.g., ANOVA and LIME) that have been used in the literature. Particularly, we found that 18% used ANOVA,

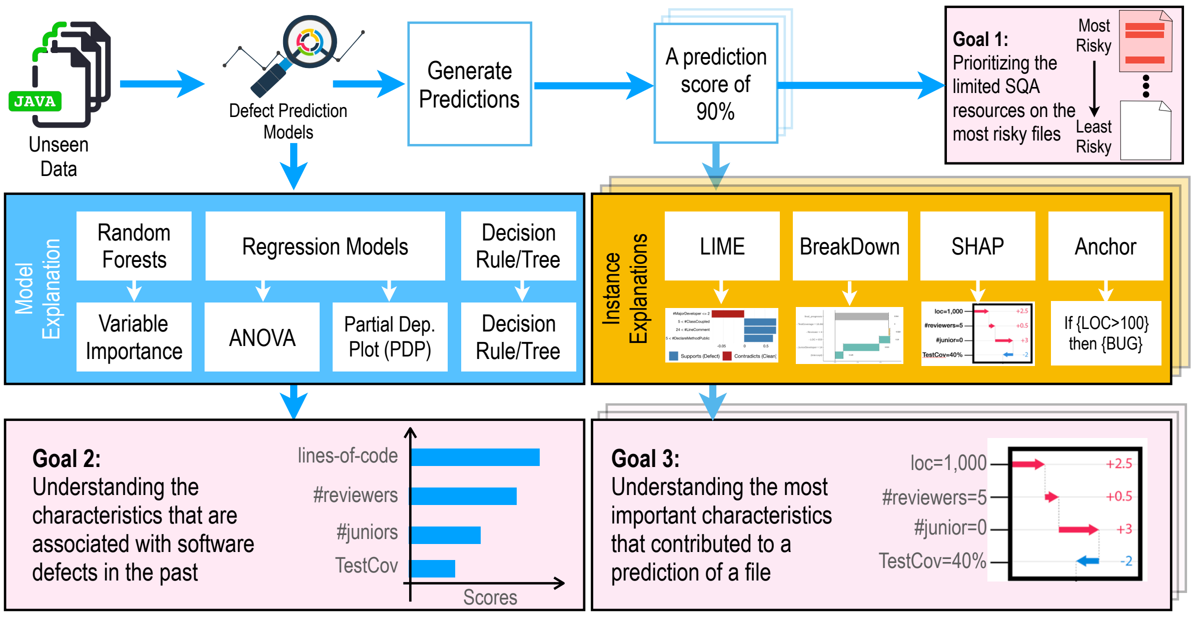


Figure 2.2 An illustrative overview of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction models.

15% used Variable Importance, 5% used Partial Dependence Plot, 3% used Decision Tree, 2% used LIME, and 1% used BreakDown to generate visual explanations. Recently, Esteves *et al.* [41] also used SHAP [125] to understand the predictions of defect prediction models. Anchor [183] (an extension of LIME [182]) was proposed to present the visual explanations in the form of decision trees/rules.

Based on our analysis of the eight selected model-agnostic techniques (see Table 2.1) that were used in prior studies, we found that visual explanations generated by these techniques produce different key information (e.g., important scores and relationship). It remains unclear about which model-agnostic techniques are considered as the most preferred by practitioners to understand defect prediction models and their predictions. Thus, we formulate the following research question:

(RQ2) Which model-agnostic techniques are the most preferred by practitioners to understand defect prediction models and their predictions?

2.3 Survey Methodology

The goal of this work is to assess practitioners' perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction models. To address our two research questions, we conducted a qualitative survey study to investigate the practitioners' perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction models. We used a survey approach, rather than other qualitative approaches (e.g., interview), since we aim to assess their perceptions of the goals along 2 dimensions (i.e., perceived usefulness and willingness to adopt) and the model-agnostic techniques along 3 dimensions (i.e., overall preference, information usefulness, information insightfulness, and information quality). Unlike an interview approach that is more unstructured, the closed-ended responses of the survey approach can be structured and quantified on a Likert scale which can be further analyzed to produce empirical evidence. The open-ended responses of the survey approach also provide in-depth insights to synthesize and generate discussions. As suggested by Kitchenham and Pfleeger [104], we considered the following steps when conducting our study: (1) Survey Design (designing a survey and developing a survey instrument), (2) An Evaluation of the Survey Instrument (evaluating the survey instrument), (3) Participant Recruitment and Selection (obtaining valid data), (4) Data Verification (verifying the data), and (5) Statistical Analysis (analysing the data). We describe each step below.

2.3.1 Survey Design

Our survey design is a cross-sectional study where participants provide their responses at one fixed point in time. The survey consists of 9 closed-ended questions, 11 open-ended questions, and 1 one-ended question for feedback on our survey. The survey takes approximately 20 minutes to complete and is anonymous. Our survey can be found in the online supplementary materials [84].

To fulfil the objectives of our study, we created three sets of closed-ended and open-ended questions with respect to the demographic information, and the two research questions. For closed-ended questions, we used agreement and evaluation ordinal scales. To mitigate the inconsistency of the interpretation of numeric ordinal scales, we labelled each level of ordinal scales with words as suggested by Krosnick [110]. The format of our survey instrument is an online questionnaire. We used Google Forms to implement this online survey. When accessing the survey, each participant was provided with an explanatory statement which describes the purpose of the study, why the participant is chosen for this study, possible benefits and risks, and confidentiality. Below, we present the rationale for the information that we captured:

Part 1–Demographics

We captured the following information, i.e., Role: engineers, managers, and researchers; Experience in years (decimal value); Current country of residence; Primary programming language; Team Size: 1-10, 11-20, 21-50, 51-100, 100+; Usage of static analysis tools: Yes / No.

The collection of demographic information (i.e., roles, experience, country) about the respondents allows us to (1) filter respondents who may not understand our survey (i.e., respondents with less relevant job roles), (2) breakdown the results by groups (e.g., developers, managers, etc), and (3) understand the impact of the demographics on the results of our study.

Team size may have an impact on SQA practices [155]. For example, small teams might use a light-weight SQA practice (e.g., static analysis), while large teams might use a rigorous SQA practice (e.g., CI/CD and automated software testing).

Primary programming languages may impact SQA practices [13]. For example, some high-level programming languages might be easier to conduct SQA practices (e.g., Python and Ruby languages) than some low-level programming languages (e.g., C language).

The usage of static analysis tools may impact the practitioners' perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations of defect prediction models. For example, practitioners who use static analysis may not perceive the benefits of the prioritization goal of defect prediction models [222]. However, the ranking of the most risky files is not the only goal of defect prediction models.

Part 2–Practitioners' perceptions of the goals of defect prediction models

To understand how practitioners perceive the goals of defect prediction models, we first illustrated the concept of defect prediction models then provided participants with a brief definition of each goal. For each goal of defect prediction models, we assessed the practitioner's perceptions along two dimensions, i.e., perceived usefulness and willingness to adopt. Perceived usefulness refers to the degree to which a person believes that using a particular system would enhance his or her job performance [116] Willingness to adopt refers to the degree to which a person is willing to adopt a particular system [222]. Thus, we asked the participants to rate the perceived usefulness and the willingness to adopt using the following evaluation ordinal scales:

- Perceived Usefulness: Not at all useful, Not useful, Neutral, Useful, and Extremely useful
- Willingness to adopt: Not at all considered, Not considered, Neutral, Considered, and Extremely considered

We then asked participants to describe the positive points and points for improvement about these goals of defect prediction models, and how the use of defect prediction models might impact their organizations when deploying in practice.

Part 3–Practitioners' perceptions of the model-agnostic techniques for generating visual explanations of defect prediction models

We provided participants with examples of visual explanation that are generated from the 6 model-agnostic techniques for defect prediction models (i.e., VarImp, Partial Dependence Plots, Decision Tree, LIME, BreakDown, and Anchor). We combined ANOVA and VarImp since both techniques provide the same information. Similarly, we combined SHAP and BreakDown since both techniques provide the same information. As suggested by Lewis *et al.* [6, 116], we use the PSSUQ (Post-Study System Usability Questionnaire) framework to evaluate the practitioners' perceptions of the model-agnostic techniques for generating visual explanations of defect prediction models. The PSSUQ framework focuses on four dimensions, i.e., information usefulness, information quality, information insightfulness, and the overall preference. Information usefulness, information quality, and information insightfulness refer to the degree to which a person satisfies that using a particular visual explanation is useful, able to comprehend, and insightful to understand the characteristics that are associated with software defects and the characteristics that contributed to a prediction of a file, respectively. For each dimension, we use the following evaluation ordinal scales:

- Extremely low, low, moderate, high, and extremely high.

We then asked participants to describe the strengths and weaknesses of each visual explanation. Finally, we asked an open-question to describe the ideal preferences of visual explanations for developing quality improvement plans.

To generate visual explanations for our survey, we used the release 2.9.0 and 3.0.0 of the Apache Lucene software system from Yatish *et al.* [233]'s corpus. The release 2.9.0 data (1,368 instances, 65 software metrics, and a defective ratio of 20%) was used to construct defect prediction models, while the release 3.0.0 data (1,337 instances, 65 software metrics, and a defective ratio of 12%) was used to evaluate such models to ensure that explanations are derived from accurate models. We also used the release 3.0.0 data to generate visual explanations of LIME, SHAP, BreakDown, and Anchor. To simplify the visual explanation for readability, we selected only five metrics, i.e., `AddedLOC`, `CommentToCodeRatio`, `LOC`, `nCommit`, and `nCoupledClass`. We provided the steps for generating visual explanations in Zenodo [84].

2.3.2 An Evaluation of the Survey Instrument

We carefully evaluated our draft survey by using a pilot study for pre-testing [122], prior to recruiting participants. We evaluated the survey with co-authors and 5 PhD students who have background knowledge in software engineering research but may not restrict to the defect prediction domain. They pointed out that the survey needs more context and details, especially for non-domain experts. Particularly, the draft survey did not provide the definition of software defect prediction, how they are handled in software companies, and how defect prediction models are used to support decision-making. Thus, at the beginning of Sections 3 and 4 of the revised draft survey, we included overview figures and scenario-based explanations to address the concern. We repeatedly refined the survey instrument to identify and fix potential problems (e.g., missing, unnecessary, or ambiguous questions) until reaching a consensus among the pre-testing participants. Finally, the survey has been rigorously reviewed, revised, and approved by the Human Research Ethics Committee of our university.

2.3.3 Participant Recruitment and Selection

The target population of our study is software practitioners. To reach our target population, we used the recruiting service provided by the Amazon Mechanical Turk. Unlike Stack-Overflow or Linkedin, the Amazon Mechanical Turk platform comes with built-in options to filter participants for participant selection and customize a monetary incentive for each participant. Particularly, we applied the participant filter options of "*Employment Industry - Software & IT Services*" and "*Job Function - Information Technology*" to ensure that we reached the target population. Another benefit of using the Amazon Mechanical Turk is that we can recruit filtered target participants relative fast. However, these benefits come with the monetary costs. We paid 6.4 USD as a monetary incentive for each participant [38, 200]. In total, our survey has 9 closed questions (450 responses) + 11 open questions (550 responses) + 1 open question (50 responses) for feedback.

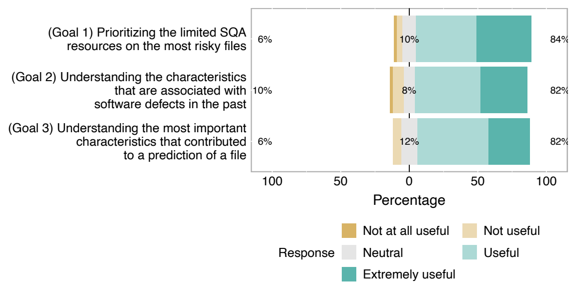
2.3.4 Data Verification

We manually read all of the open-question responses to check the completeness of the responses i.e., whether all questions were appropriately answered. We excluded 68 responses that are missing and are not related to the questions. In the end, we had a set of 982 responses. We summarized and presented the results of closed-ended responses in a Likert scale with stacked bar plots, while we discussed and provided examples of open-ended responses.

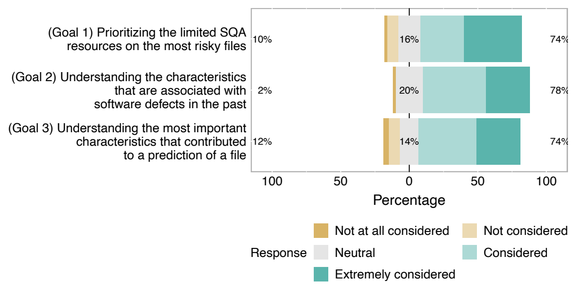
2.3.5 Statistical Analysis

For the closed-end questions with ordinal scales, we converted the ratings into scores. For example, we converted not at all useful, not useful, neutral, useful, and extremely useful to 1, 2, 3, 4 and 5 respectively. Then, we applied the ScottKnott ESD test to clusters of distributions into statistically distinct ranks. We used the implementation of the ScottKnott ESD test as provided by the ScottKnottESD R package [206, 212, 213].

For ratings of statements, we calculated the percentage of respondents who strongly agree or agree with each statement (% strongly agree+% agree) and the percentage of respondents who strongly disagree or disagree with each statement (% strongly disagree+% disagree). As suggested by Wan *et al.* [222], we also computed an agreement factor for each statement. The agreement factor is a measure of agreement between respondents, which is calculated for each statement by the following equation: (% strongly agree + % agree)/(% strongly disagree + % disagree). High values of agreement factors indicate a high agreement of respondents to a statement. The agreement factor of 1 indicates that the



(a) Perceived Usefulness



(b) Willingness to adopt

Figure 2.3 The likert scores of the perceived usefulness and the willingness to adopt from the respondents for each goal of defect prediction models.

numbers of respondents who agree and disagree with a statement are equal. Finally, low values of agreement factors indicate that a high disagreement of respondents to a statement.

2.4 Survey Results

We present the demographics of our survey, and then the results of using survey data to answer our research questions.

2.4.1 Demographics

The top two countries in which the respondents reside are India (58%) and the United States (36%). Among the respondents, they described their job roles as: Developers (50%), Managers (42%), and others (8%). The number of years of professional experience of the respondents varied from less than 5 years (26%), 6–10 years (38%), 11–15 years (22%), 16–20 years (12%), and more than 25 years (2%). They described their team size as: less than 10 people (30%), 11–20 people (30%), 21–50 people (26%), 51–100 people (2%), and more than 100 people (12%). The respondents described their experience in programming languages as: Java (44%), Python (30%), C/C++/C# (28%), and JavaScript (12%). They also answered whether they are using static analysis tools in their organizations as follows: Yes (62%) and No (38%).

These demographics indicate that the responses are collected from practitioners resided in various countries, roles, years of experience, and programming languages, indicating that our findings are likely not bound to specific characteristics of practitioners.

(RQ1) Which goals of defect prediction models that practitioners considered the most useful?

Figure 2.3 presents the Likert scores of the perceived usefulness and the willingness to adopt from the respondents for each goal of defect prediction models. Table 2.2 presents a summary

Table 2.2 (RQ1) A summary of the ScottKnott ESD rank, the agreement percentage, the disagreement percentage, and the agreement factor for the three goals of defect prediction models.

Dimension	Goal	SK Rank	% Agreement	% Disagreement	Agreement Factor
Perceived Usefulness	Goal 1 – Prioritizing the limited SQA resources on the most risky files	1	84%	6%	14.00
	Goal 2 – Understanding the characteristics that are associated with software defects in the past	1	82%	10%	8.20
	Goal 3 – Understanding the most important characteristics that contributed to a prediction of a file	1	82%	6%	13.67
Willingness to Adopt	Goal 1 – Prioritizing the limited SQA resources on the most risky files	1	74%	10%	7.40
	Goal 2 – Understanding the characteristics that are associated with software defects in the past	1	78%	2%	39.00
	Goal 3 – Understanding the most important characteristics that contributed to a prediction of a file	1	74%	12%	6.17

of the ScottKnott ESD rank, the agreement percentage, the disagreement percentage, and the agreement factor for the three goals of defect prediction models.

82%–84% of the respondents perceived that the three goals of defect prediction models are useful and nearly 80% of them are willing to adopt. Figure 2.3 shows that 82%–84% and 72%–78% of respondents rate that the goals of defect prediction models are perceived as useful and considered willing to adopt, respectively. Table 2.2 also confirms that the agreement factors are high across all goals with the values of 8.2–14 and 6–39 for perceived usefulness and willingness to adopt, respectively. The high agreement factors of responses provided by the respondents suggest that most respondents provide positive responses (e.g., useful and extremely useful) when comparing negative responses (e.g., not useful and not at all useful). Respondents provided rationales that if defect prediction models were adopted, they are likely to save developers' effort, e.g., (R6: "*... saves developers a huge amount of effort on reviewing or testing non-defective files ...*"), and improve the efficiency of code inspection (R8: "*Issues can be caught early in development.*", (R24: "*More time will be focused on critical areas. Less time will be wasted on areas without defects.*"). The ScottKnott ESD test also ranks all of the goals at the same rank, confirming that the scores among the goals have negligible effect size difference. This finding highlights that *not only the defect prediction goal but also the other two goals (i.e., understanding defect prediction models and their predictions) receive similar perceptions of usefulness and willingness to adopt with no statistically significant difference.*

Below, we discuss further if the respondents' demographics have any impact on their perceptions.

The use of static analysis tools has no significant impact (with a negligible to small effect size) on their willingness to adopt defect prediction models that are developed from various goals. The emergence of static analysis and defect prediction models is in parallel with different intellectual thoughts: one is driven by algorithms and abstraction over code, while defect prediction models are driven by statistical methods over large defect datasets [222]. Wan *et al.* [222] noted that static analysis shares some overlapping goals with defect prediction models, i.e., improving inspection efficiency, finding minimal, and potentially defective regions in source code. Thus, practitioners who use static analysis may not be willing to adopt defect prediction models. In contrast, we did not observe any significant impact of the use of static analysis tools on their willingness to adopt defect prediction models. This finding is aligned with Rahman *et al.* [175] who found that both static analysis and statistical defect prediction models provide comparable benefits.

We found that **Team Size has the largest influence on their willingness to adopt.** To investigate the impact of various demographic factors on their willingness to adopt, we built a linear regression model by using the `ols` function of the `rms` R package. The independent variables are the years of experience, roles, team size, programming languages, and static analysis, while the dependent variable is the willingness score. After using the optimism-reduced bootstrap validation (i.e., a model validation technique that randomly draws training samples with replacement then tests such models with original samples and the samples used to construct these models), the regression model achieves a goodness-of-fit (R^2) of 0.35. Then, we analyzed the Chi-square statistics of the ANOVA Type-II analysis, then normalized these Chi-square statistics into percentages to better illustrate the relative differences among variables. The ANOVA analysis indicates that Team Size has the largest influence on their willingness to adopt (i.e., 52.80% for TeamSize, 20.98% for useJava, 13.05% for usePython, 7.92% for Year, 5.01% for role). We found that the respondents who use Java, with little years of experience and a large team size (more than 100 people) tend to consider willing to adopt defect prediction models. We speculate that this has to do with a more complex nature of compiled languages (e.g., Java) when comparing to interpreted languages (e.g., Python). Nevertheless, we observe a minimal impact of the roles (developers vs managers) on their perceptions. We provided a detailed analysis of marginal effect size of each demographic factor on the estimated willingness to adopt defect prediction models in Zenodo [84].

(RQ2) Which model-agnostic techniques are the most preferred by practitioners to understand defect prediction models and their predictions?

LIME is the most preferred model-agnostic technique to understand the most important characteristics that contributed to a prediction of a file with an agreement percentage of 66%-78% along three dimensions. As shown in Table 2.3, LIME consistently appears at the top-1 ScottKnott ESD rank for all three dimensions with an agreement percentage of 76% for information usefulness, an agreement percentage of 68% for information insightfulness, and an agreement percentage of 66% for information quality. Respondents found that LIME is very easy to understand, e.g., R1 (*"Risk scores are visually oriented and users will understand it faster."*) and R50 (*"Easy to understand, we just want to know what's already ok and what's need to improve."*). However, some respondents raised concerns

Table 2.3 (RQ2) A summary of the ScottKnott ESD rank, the agreement percentage, the disagreement percentage, and the agreement factor for each model-agnostic technique for generating visual explanations of defect prediction models.

Dimension	Techniques	SK Rank	%Agreement	%Disagreement	Agreement Factor
Usefulness	**LIME**	1	76%	6%	**12.67**
	ANOVA/VarImp	2	60%	14%	4.29
	PDP	2	60%	18%	3.33
	BreakDown/SHAP	2	50%	18%	2.78
	Decision Tree	2	54%	18%	3.00
	Anchor/LORE	2	60%	28%	2.14
Insightfulness	**LIME**	1	**68%**	**8%**	**8.50**
	Decision Tree	1	**52%**	**10%**	**5.20**
	BreakDown/SHAP	2	58%	12%	4.83
	PDP	2	54%	16%	3.38
	ANOVA/VarImp	2	58%	18%	3.22
	Anchor/LORE	3	46%	24%	1.92
Quality	**LIME**	1	**66%**	**4%**	**16.50**
	ANOVA/VarImp	1	**70%**	**8%**	**8.75**
	Decision Tree	1	**70%**	**14%**	**5.00**
	PDP	2	56%	24%	2.33
	BreakDown/SHAP	2	52%	26%	2.00
	Anchor/LORE	2	56%	24%	2.33

that LIME generates too much information (i.e., too many characteristics for many defective files), e.g., R9 (*"... But it also could be time consuming to review depending on if there are many different files to review."*) and R50 (*"... too large and too tired to read 10 charts for 10 files."*).

ANOVA/VarImp is the second most preferred technique to understand the characteristics that are associated with software defects in the past with an agreement percentage of 58%–70% along three dimensions. As shown in Table 2.3, ANOVA/VarImp consistently appears at the second ScottKnott ESD Rank, except for information quality, with an agreement percentage of 60% for information usefulness, an agreement percentage of 58% for information insightfulness, and an agreement percentage of 70% for information quality. Similar to LIME, respondents found that the bar charts of ANOVA/VarImp are very easy to understand, e.g., R50 (*"Explains risk values of several factors."*). This finding indicates that while both LIME and ANOVA/VarImp generate different information, they are complementary to each other. This suggests that while LIME should be used to understand the most important characteristics that contributed to a prediction of a particular file (Goal 3), ANOVA/VarImp is still needed to understand the overview of the general characteristics that are associated with software defects in the past (Goal 2).

2.5 Threats to the Validity

Construct validity: We studied a limited period of publications (i.e., 2015-2020). Thus, the results may be altered if the studied period is changed. Future research should consider expanding our study to a longer publication period.

Hilderbrand *et al.* [74] found that there are statistically gender differences in the cognitive style of developers. Yet, gender is not considered in our survey. Thus, our recommendation may not be generalized to all genders. Future studies should consider gender aspects when collecting the demographics of respondents.

In this study, we design the survey with the assumptions of file-level defect prediction. However, Wan *et al.* [222] found that commit level is the most preferred by practitioners. Thus, our results may not be applicable to other granularity levels of predictions (e.g., commits, methods).

Internal validity: One potential threat is related to the bias in the responses due to the imbalanced nature of the recruited participants. Also, practitioners' perceptions may biased and can change from one person to another or even one organization to another [36]. However, the population of the recruited participants is composed of practitioners of different roles, years of experience, country of residence, and programming languages. To mitigate issues of fatigue bias in our survey study, we conducted a pilot study with co-authors and PhD students to ensure that the survey can be completed within 20 minutes.

External validity: We recruited a limited number of participants. Thus, the results and findings may not generalise to all practitioners. Nevertheless, we described the survey design in details and provided sets of survey questions in the online supplementary materials [84] for future replication.

2.6 Conclusions

We present the findings of the qualitative survey of 50 practitioners on their perceptions of the goals of defect prediction models and the model-agnostic techniques for generating visual explanations for defect prediction models. Through a qualitative analysis of a survey study of 50 practitioners from multi-organisations, we conclude that: (1) Researchers should put more effort into investigating how to improve the understanding of defect prediction models and their predictions, since our analysis of related work found that these two goals are still under research despite receiving similar perceptions of usefulness and willingness to adopt with no statistically significant difference; and (2) Practitioners can use LIME and ANOVA/VarImp to better understand defect prediction models and their predictions. Finally, we discuss many open questions that are significant, yet remain largely unexplored (e.g., developers' privacy and fairness when deploying defect prediction models in practice, and human-in-the-loop defect prediction models).

2.6.1 Chapter Remarks

In this chapter, we introduce a concept of explainable defect models and conduct a survey study of 50 software practitioners from multi-organisations. The results show that, despite receiving little attention from the research community, most of the recruited software practitioners perceived understanding defect prediction models and their predictions as useful and are willing to adopt explainable defect models in practice. Thus, these findings motivated us to further explore the explanation of defect prediction models to support SQA planning.

Chapter 3

Investigate the Impact of Correlated Metrics on the Explanation of Defect Models

An earlier version of the work in this chapter appears in the Transaction on Software Engineering (TSE) [85].

3.1 Introduction

Defect models are constructed using historical software project data to identify defective modules and explore the impact of various phenomena (i.e., software metrics) on software quality. The interpretation of such models is used to build empirical theories that are related to software quality (i.e., what software metrics share the strongest association with software quality?). These empirical theories are essential for project managers to chart software quality improvement plans to mitigate the risk of introducing defects in future releases (e.g., a policy to maintain code as simple as possible).

Plenty of prior studies investigate the impact of many phenomena on code quality using software metrics, for example, code size [236], code complexity [70, 128, 197], change complexity [102, 153, 156, 197, 239], antipatterns [100], developer activity [197], developer experience [173], developer expertise [17], developer and reviewer knowledge [216], design [9, 24, 25, 30, 37], reviewer participation [131, 217], code smells [99], and mutation testing [20]. To perform such studies, there are five common steps: (1) formulating of hypotheses that pertain to the phenomena that one wishes to study; (2) designing appropriate metrics to operationalize the intention behind the phenomena under study; (3) defining a model specification (e.g., the ordering of metrics) to be used when constructing an analytical model; (4) constructing an analytical model using, for example, regression models [17, 153, 216, 217, 238] or random forest models [56, 96, 147, 177]; and (5) examining the ranking of metrics using a model interpretation technique (e.g., ANOVA Type-I, one of the most commonly-used interpretation techniques since it is the default built-in function for logistic regression (`glm`) models in R) in order to test the hypotheses.

For example, to study whether complex code increases project risk, one might use the number of reported bugs (`bugs`) to capture risk, and the McCabe's cyclomatic complexity (`CC`) to capture code complexity, while controlling for code size (`size`). We note that one needs to use control metrics to ensure that findings are not due to confounding factors (e.g., large modules are more likely to have more bugs). Then, one must construct an analytical model with a model specification of `bugs~size+CC`. One would then use an interpretation technique (e.g. ANOVA Type-I) to determine the ranking of metrics (i.e., which metrics have a strong relationship with bugs).

Metrics of prior studies are often correlated [54, 72, 73, 86, 88, 207, 211, 235]. For example, Herraiz *et al.* [73], and Gil *et al.* [54] point out that code complexity (`CC`) is often correlated with code size (`size`). Zhang *et al.* [235] point out that many metric aggregation schemes (e.g., averaging or summing of McCabe's cyclomatic complexity values at the function level to derive file-level metrics) often produce correlated metrics.

Recent studies raise concerns that correlated metrics may impact the interpretation of defect models [211, 235]. Our preliminary analysis also shows that simply rearranging the ordering of correlated metrics in the model specification (e.g., from `bugs~size+CC` to `bugs~CC+size`) would lead to a different ranking of metrics—i.e., the importance scores are sensitive to the ordering of correlated metrics in a model specification. Thus, if one wants to show that code complexity is strongly associated with risk in a project, one simply needs to put code complexity (`CC`) as the first metric in their models (i.e., `bugs~CC+size`), even though a more careful analysis would show that `CC` is not associated with `bugs` at all. The sensitivity of the model specification when correlated metrics are included in a model is a critical problem, since the contribution of many prior studies can be altered by simply re-ordering metrics in the model specification if correlated metrics are not properly mitigated. Unfortunately, a literature survey of Shihab [193] shows that as much as 63% of defect studies that are published during 2000-2011 do not mitigate correlated metrics prior to constructing defect models.

In this chapter, we set out to investigate (1) the impact of correlated metrics on the interpretation of defect models. After removing correlated metrics, we investigate (2) the consistency of the interpretation of defect models; and (3) its impact on the performance and stability of defect models. In order to detect and remove correlated metrics, we apply the variable clustering (VarClus) and the variance inflation factor (VIF) techniques. We construct logistic regression and random forest models using mitigated (i.e., no correlated metrics) and non-mitigated datasets (i.e., not treated). Finally, we apply 9 model interpretation techniques, i.e., ANOVA Type-I, 4 test statistics of ANOVA Type-II (i.e., Wald, Likelihood Ratio, F, and Chi-square), scaled and non-scaled Gini Importance, and scaled and non-scaled Permutation Importance. We then compare the performance and interpretation of defect models that are constructed using mitigated and non-mitigated datasets. Through a study of 14 publicly-available defect datasets of systems that span both proprietary and open source domains, we address the following four research questions:

(RQ1) How do correlated metrics impact the interpretation of defect models?
ANOVA Type-I and Type-II often produce the lowest consistency and the highest level of discrepancy of the top-ranked metric, and have the highest impact on the direction of the ranking of metrics between mitigated and non-mitigated models when compared to Gini and Permutation Importance. This finding highlights the risks of not mitigating correlated metrics in the ANOVA analyses of prior studies.

(RQ2) After removing all correlated metrics, how consistent is the interpretation of defect models among different model specifications?
After removing all correlated metrics, the top-ranked metric according to ANOVA Type-II, Gini Importance, and Permutation Importance are consistent. However, the top-ranked metric according to ANOVA Type-I is inconsistent, since the ranking of metrics is impacted by its order in the model specification when analyzed using ANOVA Type-I (which is the default analysis for the `glm` model in R and is commonly-used in prior studies). This finding suggests that ANOVA Type-I must be avoided even if all correlated metrics are removed.

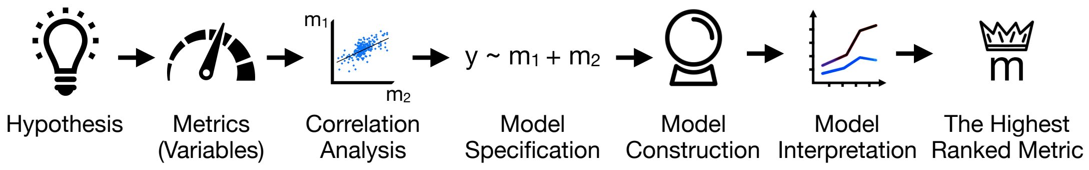


Figure 3.1 An overview of the analytical modelling process.

(RQ3) After removing all correlated metrics, how consistent is the interpretation of defect models among the studied interpretation techniques?
After removing all correlated metrics, we find that the consistency of the ranking of metrics among the studied interpretation techniques is improved by 15%- 64% for the top-ranked metric and 21%-71% for the top-3 ranked metrics, respectively, highlighting the benefits of removing all correlated metrics on the interpretation of defect models, i.e., the conclusions of studies that rely on one interpretation technique may not pose a threat after mitigating correlated metrics.

(RQ4) Does removing all correlated metrics impact the performance and stability of defect models?
Removing all correlated metrics impacts the AUC, F-measure, and MCC performance of defect models by less than 5 percentage points, suggesting that researchers and practitioners should remove correlated metrics with care especially for safety-critical software domains.

Based on our findings, we suggest that: *When the goal is to derive sound interpretation from defect models, our results suggest that future studies must (1) mitigate correlated metrics prior to constructing a defect model, especially for ANOVA analyses; and (2) avoid using ANOVA Type-I even if all correlated metrics are removed, but instead opt to use ANOVA Type-II and Type-III for additive and interaction models, respectively. Due to the variety of the built-in interpretation techniques and their settings, our paper highlights the essential need for future studies to report the exact specification (i.e., model formula) of their models and settings (e.g., the calculation methods of the importance score) of the used interpretation techniques.*

3.1.1 Chapter Organisation

Section 3.2 provides background and motivation. Section 3.3 describes the design and the setup of our study, while Section 3.4 presents the results with respect to the four investigations. Section 3.5 draws practical guidelines based on the experimental results. Section 3.6 elaborates on the threats of the study. Finally, Section 3.7 draws conclusions.

3.2 Background and Motivation

3.2.1 Analytical Modelling Process

Figure 3.1 provides an overview of the commonly-used analytical modelling process. First, one must formulate a set of hypotheses pertaining to phenomena of interest (e.g., whether the size of a module increases the risk associated with that module). Second, one must determine a set of metrics which operationalize the hypothesis of interest (e.g., the total lines of code for size, and the number of field reported bugs to capture the risk that is associated with a module). Third, one must perform a correlation analysis to remove correlated metrics. Forth, one must define a model specification (e.g., the ordering of metrics) to be used when constructing an analytical model. Fifth, one is then ready to construct an analytical model using a machine learning technique (e.g., a random forest model) or a statistical learning technique (e.g., a regression model). Finally, one analyzes the ranking of the metrics using model interpretation techniques (e.g., ANOVA or Breiman's Variable Importance) in order to test the hypotheses of interest. The importance ranking of the metrics is essential for project managers to chart appropriate software quality improvement plans to mitigate the risk of introducing defects in future releases. For example, if code complexity is identified as the top-ranked metric, project managers then can suggest developers to reduce the complexity of their code to reduce the risk of introducing defects.

3.2.2 Correlated Metrics and Concerns in the Literature

Correlated metrics are metrics (i.e., independent variables) that share a strong linear correlation among themselves. In this paper, we focus on two types of correlation among metrics, i.e., collinearity and multicollinearity. Collinearity is a phenomenon in which one metric can be linearly predicted by another metric. On the other hand, multicollinearity is a phenomenon in which one metric can be linearly predicted by a combination of two or more metrics.

Prior work points out that software metrics are often correlated [54, 72, 73, 86, 88, 207, 211, 235]. However, little is known about the prevalence of correlated metrics in the publicly-available defect datasets. Thus, we set out to investigate how many defect datasets of which metrics that share a strong relationship with defect-proneness are correlated. Unfortunately, the results of our recent work [86] show that correlated metrics that share a strong relationship with defect-proneness are prevalent in 83 of the 101 (82%) publicly available defect datasets.

In addition, prior work raises concerns that correlated metrics may impact the interpretation of defect models [211, 235]. To better understand how correlated metrics impact the interpretation of defect models, we set out to investigate (1) the impact of the number of correlated metrics on the importance scores of metrics, and (2) the impact of the ordering of correlated metrics in a model specification on the importance ranking metrics. The

Table 3.1 A summary of the studied correlation analysis techniques, the two studied analytical learners, and the 9 studied interpretation techniques.

Correlation Analysis	Analytical Learner	Interpretation Tech-nique	Test Statistic	R function
Variable Clustering [150, 216, 217, 219]	Logistic Regression (`glm` and `lrm`) [17, 18, 153, 154, 238]	Type-I	Deviance	`stats::anova(glm.model)`
		Type-II	Wald	`car::Anova(glm.model, type=2, test.statistic='Wald')`
			Likelihood Ratio (LR)	`car::Anova(glm.model, type=2, test.statistic='LR')`
Variance Inflation Factor [11, 36, 131, 194, 195]			F	`car::Anova(glm.model, type=2, test.statistic='F')`
			Chi-square	`rms::anova(lrm.model, test='Chisq')`
	Random Forest [56, 59, 96, 147, 177]	Scaled Gini	MeanDecreaseGini	`randomForest::importance(model, type = 2, scale = TRUE)`
Redundancy Analysis [7, 95, 150, 196, 211]		Non-scaled Gini	MeanDecreaseGini	`randomForest::importance(model, type = 2, scale = FALSE)`
		Scaled Permutation	MeanDecreaseAccuracy	`randomForest::importance(model, type = 1, scale = TRUE)`
		Non-scaled Permutation	MeanDecreaseAccuracy	`randomForest::importance(model, type = 1, scale = FALSE)`

results of our preliminary analyses show that the importance scores of metrics substantially decrease when there are correlated metrics in the models for both ANOVA analyses of logistic regression and Variable Importance analyses (i.e., Gini and Permutation) of random forest. The importance scores of metrics are also sensitive to the ordering of correlated metrics (except for ANOVA Type-II).

3.2.3 Techniques for Mitigating Correlated Metrics

There is a plethora of techniques that have been used to mitigate irrelevant and correlated metrics in the domain of defect prediction, e.g., dimensionality reduction [124, 153, 235], feature selection [5, 140], and correlation analysis [36, 194, 217].

Dimensionality reduction transforms an initial set of metrics into a set of transformed metrics that is representative to the initial set of metrics. Prior work has adopted dimensionality reduction techniques (e.g., Principal Component Analysis) to mitigate correlated metrics and improve the performance of defect models [124, 153, 235]. Since the set of transformed metrics does not hold the assumption of the initial set of metrics, and is not sensible for model interpretation and statistical inference [207], we exclude dimensionality reduction techniques from this study.

Feature selection produces an optimal subset of metrics that are relevant and non-correlated. One of the most commonly-used feature selection techniques is the correlation-based feature selection technique (CFS) [64] which searches for the best subset of metrics that share the highest correlation with the outcome (e.g., defect-proneness) while having the lowest correlation among each other. To better understand whether feature selection techniques mitigate correlated metrics, we set out to perform a correlation analysis on the metrics that are selected by feature selection techniques. We focus on the two commonly-used techniques in the domain of defect prediction, i.e., Information Gain and correlation-based feature selection techniques. The results of our preliminary analysis show that the metrics that are selected by the two studied feature selection techniques are correlated (with a Spearman correlation coefficient up to 0.98), suggesting that the commonly-used feature selection techniques do not mitigate correlated metrics.

Correlation analysis is used to measure the correlation among metrics given a threshold. Prior work applies correlation analysis techniques to identify and mitigate correlated metrics [36, 194, 217, 219]. Based on a literature survey of Hall *et al.* [65] and Shihab [193], we select the commonly-used correlation analysis techniques: Variable Clustering analysis (VarClus), and Variance Inflation Factor (VIF).

Variable Clustering (VarClus) is a hierarchical clustering view of the correlation between metrics [190]. We use the implementation of the variable clustering analysis as provided by the `varclus` function of the `Hmisc` R package [67], which is made up of 2 steps.

(Step 1) Compute the correlations between metrics. We use the Spearman rank correlation test (ρ) to assess the correlation between metrics. We choose the Spearman test instead of other types of correlation (e.g., Pearson) because the Spearman test is resilient to non-normality in a dataset as commonly present in software engineering and defect datasets, in particular.

(Step 2) Select one metric from each of the sub-hierarchies for inclusion in a model. Once, a hierarchical overview of the correlation among metrics is constructed, we use the interpretation of correlation coefficients ($|\rho|$) as provided by Kraemer *et al.* [107], i.e., a correlation coefficient of above 0.7 is considered a strong correlation. Thus, for each sub-hierarchy of software metrics with a correlation $|\rho| > 0.7$, we select only one metric from the sub-hierarchy for inclusion in our models. As suggested by prior studies [130, 132, 217], we select the simplest metric to calculate (or interpret) for each sub-hierarchy.

While the variable clustering analysis (VarClus) technique reduces collinearity among metrics, it does not detect all of the inter-correlated metrics (a.k.a. *multi-collinearity*), i.e., a metric that can be predicted from the other metrics in the model with a certain degree of accuracy.

Variance Inflation Factor (VIF) measures the magnitude of multi-collinearity [44]. We use the implementation of the Variance Inflation Factor analysis as provided by the `vif` function of the `rms` R package [69]. Broadly speaking, VIF is made up of 3 steps.

(Step 1) Construct a regression model for each metric. For each metric, we construct a model using the other metrics to predict that particular metric.

(Step 2) Compute a VIF score for each metric. The VIF score for each metric is computed using the following formula: $\text{VIF} = \frac{1}{1-R^2}$, where R^2 is the explanatory power of the regression model from Step 1. A high VIF score of a metric indicates that a given metric can be accurately predicted by the other metrics. Thus, that given metric is considered redundant and should be removed from our model.

(Step 3) Remove metrics with a VIF score that is higher than a given threshold. We remove metrics with a VIF score that is higher than a given threshold. We use a VIF threshold of 5 to determine the magnitude of multi-collinearity, as it is suggested by Fox [43] and is commonly used in prior work [11, 131, 194, 195]. Similar to the variable clustering analysis, we repeat the above three steps until the VIF scores of all remaining metrics are lower than the threshold.

3.2.4 Techniques for Explaining Defect Models

Since there are many analytical learners that can be used to investigate the impact of correlated metrics on defect models, the aforementioned surveys guide our selection of the two commonly-used analytical learners: logistic regression [17, 18, 36, 103, 136, 153, 154, 178, 238] and random forest [56, 59, 96, 147, 177]. These techniques are two of the most commonly-used analytical learners for defect models and they have built-in techniques for model interpretation (i.e., ANOVA for logistic regression and Breiman's Variable Importance for random forest). Finally, we select 9 model interpretation techniques, ANOVA Type-I, ANOVA Type-II with 4 test statistics (i.e., Wald, Likelihood Ratio, F, and Chi-square), scaled and non-scaled Gini Importance, and scaled and non-scaled Permutation Importance.

Analysis of Variance (a.k.a. multi-way ANOVA) is a statistical test that examines the importance of multiple independent variables (e.g., two or more software metrics) on the outcome (e.g., defect-proneness) [42]. The significance of each metric in a regression model is estimated from the calculation of the Sum of Squares (SS)—i.e., the explained variance of the observations with respect to their mean value. A high SS value of a metric indicates that the metric is highly important. There are two commonly-used approaches to calculate the Sum of Squares for ANOVA, namely, Type-I and Type-II. We provide a description of the two types of ANOVA below.

Type-I, *one of the most commonly-used interpretation techniques and the default interpretation technique for a logistic regression (`glm`) model in R*, examines the importance of each metric in a sequential order [28, 43]. In other words, Type-I measures the improvement of the Residual Sum of Squares (RSS) (i.e., the unexplained variance) when each metric is sequentially added into the model. Hence, Type-I attributes as much variance as it can to the first metric before attributing residual variance to the second metric in the model specification. Thus, the importance (i.e., produced ranking) of metrics is dependent on the ordering of metrics in the model specification.

The calculation starts from the RSS of the preliminary model ($y \sim 1$), i.e., a null model that is fitted without any software metrics. We then compute the RSS of the first metric by fitting a regression model with the first metric ($y \sim m_1$). Thus, the importance of the first metric (m_1) is the improvement between the unexplained variances (RSS) of the preliminary model and the model that is constructed by the first metric.

$$\mathrm{SS}(m_1) = \mathrm{RSS}(\mathrm{Model_{null}}) - \mathrm{RSS}(m_1) \tag{3.1}$$

Similar to the computation of the importance of the first metric, the importance of the remaining metrics is computed using the following equation.

$$\mathrm{SS}(m_i) = \mathrm{RSS}(\mathrm{m}_1 + ... + \mathrm{m}_{\mathrm{i}-1}) - \mathrm{RSS}(m_1 + ... + m_i) \tag{3.2}$$

Type II, an enhancement to the ANOVA Type-I, examines the importance of each metric in a hierarchical nature, i.e., the ordering of metrics is rearranged for each examination [28, 43]. The importance of metrics (Type-II) measures the improvement of the Residual Sum of Squares (RSS) (i.e., the unexplained variance) when adding a metric under examination to the model after the other metrics. In other words, the importance of metrics (Type-II) is equivalent to a Type-I where a metric under examination appears at the last position of the model. The intuition is that the Type-II is evaluated after all of the other metrics have been accounted for. The importance of each metric (i.e., SS(m_e)) measures the improvement of the RSS of the model that is constructed by adding only the other metrics except for the metric under examination, and the RSS of the model that is constructed by adding the other metrics where the metric under examination appears at the last position of the model. For example, given a set of M metrics, and $e, i, j \in [1, M]$, the importance of each metric m_e can be explained as follows:

$$\mathrm{SS}(m_e) = \mathrm{RSS}(m_i + \ldots + m_j) - \mathrm{RSS}(m_i + \ldots + m_j + m_e) \tag{3.3}$$

where m_e is the metric under examination and $m_i + \ldots + m_j$ is a set of the other metrics except the metric under examination.

In this study, we consider different variants of test statistics for ANOVA Type-II (i.e., Wald, Likelihood Ratio (LR), F, and Chi-square).

Variable importance (a.k.a. VarImp) is an approach to examine the importance of software metrics for random forest models. There are two commonly-used calculation approaches of variable importance scores, namely, Gini Importance and Permutation Importance, which we describe below.

Gini Importance (a.k.a. MeanDecreaseGini) determines the importance of metrics from the decrease of the Gini Index, i.e., the distinguishing power for the defective class due to a given metric [21, 22]. We start from a random forest model that is constructed using the original dataset with multiple trees, where each tree is constructed using a bootstrap sample. For each tree, a parent node (i.e., G_{Parent}) is split by the best cut-point into two descendent nodes (i.e., $G_{\mathrm{Desc.1}}$ and $G_{\mathrm{Desc.2}}$). The calculation of the Gini Importance of each metric is made up of 2 steps:

(Step 1) Compute the DecreaseGini for all of the trees in the random forest model. The DecreaseGini is the improvement of the ability to distinguish between two classes across parent and its descendent nodes. We compute the DecreaseGini using the following equation:

$$\mathrm{DecreaseGini}(m_i) = I_{m_i} = G_{\mathrm{Parent}} - G_{\mathrm{Desc.1}} - G_{\mathrm{Desc.2}} \tag{3.4}$$

where G is the Gini Index, i.e., the distinguishing power of defective class for a given metric. The Gini Index is computed using the following equation: $G = \sum_{i=1}^{N_{\mathrm{Class}}} p_i(1 - p_i)$, where N_{Class} is the number of classes and p_i is the proportion of Class$_i$.

(Step 2) Compute the MeanDecreaseGini measure. Finally, the importance of each metric (i.e., MeanDecreaseGini) is the average of the DecreaseGini values from all of the splits of

that metric across all the trees in the random forest model. A high MeanDecreaseGini value of a metric indicates that the metric is highly important.

In this study, we consider both the scaled and non-scaled importance scores for the Gini Importance.

Permutation Importance (a.k.a. MeanDecreaseAccuracy) determines the importance of metrics from the decrease of the accuracy (i.e., the misclassification rate) when the values of a given metric are randomly permuted [21, 22]. Similar to MeanDecreaseGini, we start from a random forest model that is constructed using an original dataset with multiple trees, where each tree is constructed using an out-of-sample bootstrap. The calculation of the Permutation Importance is made up of 2 steps:

(Step 1) Compute the DecreaseAccuracy of each tree in the random forest model. The DecreaseAccuracy is the decrease of the accuracy (i.e., misclassification rate) between a model that is tested using the original out-of-bag testing samples and a model that is tested using permuted out-of-bag testing samples, i.e., a dataset with one metric permuted, while all other metrics are unchanged).

(Step 2) Compute the MeanDecreaseAccuracy measure. Finally, the importance of each metric (i.e., MeanDecreaseAccuracy) is the average of the DecreaseAccuracy values across all of the trees in the random forest model. A high MeanDecreaseAccuracy value of a metric indicates that the metric is highly important.

Similar to Gini Importance, in this study, we consider both the scaled and non-scaled importance scores for the Permutation Importance.

We provide the detailed explanation of the studied correlation analysis techniques, analytical learners, and interpretation techniques in Table 3.1.

3.3 Experimental Design and Setup

3.3.1 Studied Datasets

In selecting the studied datasets, we identify four important criteria that need to be satisfied:

Criterion 1—Publicly-available defect datasets. Prior work raises concerns about the replicability of software engineering studies [184]. In order to foster future replication of our work, we focus on publicly-available defect datasets.

Criterion 2—Datasets that are reliable and high quality. Defect models rely greatly on the quality of the datasets that are used to construct them [209]. Shepperd *et al.* [192] raise concerns related to data quality in the NASA datasets. Furthermore, Petrić *et al.* [168] show that problematic data remain in the cleaned NASA datasets. Thus, the quality of the NASA datasets is questionable. To ensure that the studied datasets are reliable and high quality, we exclude the NASA datasets from our study.

Criterion 3—Datasets with correlated metrics that have a strong relationship with defect-proneness. Correlated metrics that have a weak relationship with defect-proneness

Table 3.2 A statistical summary of the studied datasets.

Project	Dataset	Modules	Metrics	Correlated Metrics	EPV	AUC_{LR}	AUC_{RF}
Apache	Lucene 2.4	340	20	9	10	0.74	0.77
	POI 2.5	385	20	11	12	0.80	0.90
	POI 3.0	442	20	10	14	0.79	0.88
	Xalan 2.6	885	20	8	21	0.79	0.85
	Xerces 1.4	588	20	11	22	0.91	0.95
Eclipse	Debug 3.4	1,065	17	9	15	0.72	0.81
	JDT	997	15	10	14	0.81	0.82
	Mylyn	1,862	15	10	16	0.78	0.74
	PDE	1,497	15	9	14	0.72	0.72
	Platform 2.0	6,729	32	24	30	0.82	0.84
	Platform 3.0	10,593	32	24	49	0.79	0.81
	SWT 3.4	1,485	17	7	38	0.87	0.97
Proprietary	Prop 1	18,471	20	10	137	0.75	0.79
	Prop 4	8,718	20	11	42	0.74	0.72

may not be as important as metrics that have a strong relationship with defect-proneness. To ensure that the studied metrics are of importance to practitioners when interpreting defect models, we only focus on the correlated metrics that share a strong relationship with defect-proneness.

Criterion 4—Datasets where we can accurately derive interpretations. Analysts would only consider models (i.e., logistic regression and random forest) that fit the data well (i.e., AUC > 0.7) and are stable (i.e., EPV > 10) [212]. Hence, we only focus on datasets that produce such accurate and stable models.

To satisfy criterion 1, similar to prior work [210], we begin our study using a collection of the 101 publicly-available defect datasets that are collected from 5 different corpora, i.e., 76 datasets from the Tera-PROMISE Repository, 12 clean NASA datasets as provided by Shepperd *et al.* [192], 5 datasets as provided by Kim *et al.* [101, 230], 5 datasets as provided by D'Ambros *et al.* [33, 34], and 3 datasets as provided by Zimmermann *et al.* [238]. To satisfy criterion 2, we exclude 12 datasets where their data quality are questionable. To satisfy criterion 3, we exclude 14 datasets where their correlated metrics do not share a strong relationship with defect-proneness. To satisfy criterion 4, we exclude 58 datasets with an EPV value below 10 and 3 datasets on which models that are constructed produce an AUC value below 0.7. Hence, we focus on 14 datasets of systems that span across proprietary and open-source systems.

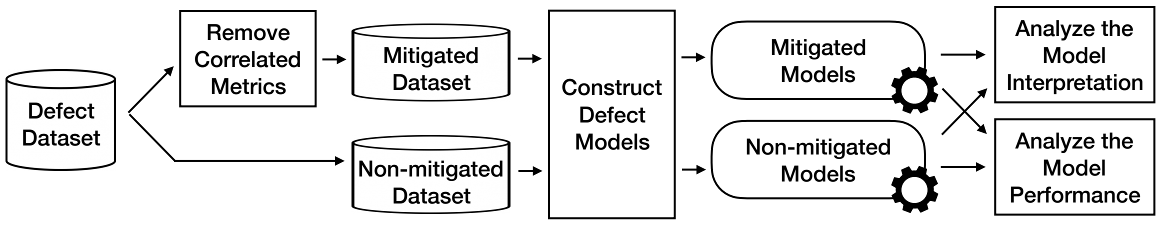


Figure 3.2 An overview diagram of the design of our study.

Table 3.2 shows a statistical summary of the studied datasets, while Figure 3.2 provides an overview of the design of our study. Below, we discuss the design of the study that we perform in order to address our four research questions.

3.3.2 Remove Correlated Metrics

To investigate the impact of correlated metrics on the performance and interpretation of defect models and address our four research questions, we start by removing highly-correlated metrics in order to produce mitigated datasets, i.e., datasets where correlated metrics are removed. To do so, we apply variable clustering analysis (VarClus) and variable influence factor analysis (VIF). We use the interpretation of Spearman correlation coefficients ($|\rho|$) as provided by Kraemer *et al.* [107] to identify correlated metrics, i.e., a Spearman correlation coefficient of above 0.7 is considered a strong correlation. We use a VIF threshold of 5 to identify inter-correlated metrics, as it is suggested by Fox [43] and is commonly used in prior work [11, 131, 194, 195]. We use the implementation of the variable clustering analysis as provided by the `varclus` function of the `Hmisc` R package [67]. We use the implementation of the VIF analysis as provided by the `vif` function of the `rms` R package [69].

3.3.3 Construct Defect Models

To examine the impact of correlated metrics on the performance and interpretation of defect models, we construct our models using the *non-mitigated datasets* (i.e., datasets where correlated metrics are not removed) and *mitigated datasets* (i.e., datasets where correlated metrics are removed). To construct defect models, we perform the following steps:

(CM1) Generate bootstrap samples. To ensure that our conclusions are statistically sound and robust, we use the out-of-sample bootstrap validation technique, which leverages aspects of statistical inference [39, 47, 68, 205, 212]. We first generate bootstrap sample of sizes N with replacement from the mitigated and non-mitigated datasets. The generated sample is also of size N. We construct models using the bootstrap samples, while we measure the performance of the models using the samples that do not appear in the bootstrap samples. On average, 36.8% of the original dataset will not appear in the bootstrap samples, since the samples are drawn with replacement [39]. We repeat the out-of-sample bootstrap process for 100 times and report their average performance.

(CM2) Construct defect models. For each bootstrap sample, we construct logistic regression and random forest models. We use the implementation of logistic regression as provided by the `glm` function of the `stats` R package [214] and the `lrm` function of the `rms` R package [69] with the default parameter setting. We use the implementation of random forest as provided by the `randomForest` function of the `randomForest` R package [23] with the default `ntree` value of 100, since recent studies [210, 213] show that parameters of random forest are insensitive to the performance of defect models. To ensure that the training and testing corpora share similar characteristics and representative to the original dataset, we do not re-balance nor do we re-sample the training data to avoid any impact on the interpretation of defect models [208].

3.3.4 Analyze the Model Interpretation

To address RQ1, RQ2, and RQ3, we analyze the importance ranking of metrics of the models that are constructed using non-mitigated datasets and mitigated datasets. The analysis of model interpretation is made up of 2 steps.

(MI1) Compute the importance score of metrics. We investigate the impact of correlated metrics on the interpretation of defect models using different model interpretation techniques. Thus, we apply the 9 studied model interpretation techniques, i.e., Type-I, Type-II (Wald, LR, F, Chisq), scaled and non-scaled Gini Importance, and scaled and non-scaled Permutation Importance.

(MI2) Identify the importance ranking of metrics. To statistically identify the importance ranking of metrics, we apply the improved Scott-Knott Effect Size Difference (ESD) test (v2.0) [206]. The Scott-Knott ESD test is a mean comparison approach that leverages a hierarchical clustering to partition a set of treatment means (i.e., means of importance scores) into statistically distinct groups *with statistically non-negligible difference*. The Scott-Knott ESD test ranks each metric at only a single rank, however several metrics may appear within one rank. Finally, we identify the importance ranking of metrics for the non-mitigated and mitigated models. Thus, each metric has a rank for each model interpretation technique and for each of the mitigated and non-mitigated models. We use the implementation of Scott-Knott ESD test as provided by the `sk_esd` function of the `ScottKnottESD` R package [206].

3.3.5 Analyze the Model Performance

To address RQ4, we analyze the performance of the models that are constructed using non-mitigated datasets and mitigated datasets.

First, we use the Area Under the receiver operator characteristic Curve (AUC) to measure the discriminatory power of our models, as suggested by recent research [52, 114, 174]. The AUC is a threshold-independent performance measure that evaluates the ability of models in discriminating between defective and clean modules. The values of AUC range between 0 (worst performance), 0.5 (no better than random guessing), and 1 (best performance) [66].

Second, we use the F-measure, i.e, a threshold-dependent measure. F-measure is a harmonic mean (i.e., $\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision}+\text{recall}}$) of precision ($\frac{\text{TP}}{\text{TP+FP}}$) and recall ($\frac{\text{TP}}{\text{TP+FN}}$). Similar to prior studies [5, 237], we use the default probability value of 0.5 as a threshold value for the confusion matrix, i.e., if a module has a predicted probability above 0.5, it is considered defective; otherwise, the module is considered clean.

Third, we use the Matthews Correlation Coefficient (MCC) measure, i.e, a threshold-dependent measure, as suggested by prior studies [127, 191]. MCC is a balanced measure based on true and false positives and negatives that is computed using the following equation: $\frac{\text{TP}\times\text{TN}-\text{FP}\times\text{FN}}{\sqrt{(\text{TP+FP})(\text{TP+FN})(\text{TN+FP})(\text{TN+FN})}}$.

3.4 Experimental Results

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

(RQ1) How do correlated metrics impact the interpretation of defect models?

Motivation. Prior work raises concerns that metrics are often correlated and should be mitigated [54, 72, 73, 86, 207, 211, 235]. For example, Herraiz *et al.* [73], and Gil *et al.* [54] point out that code complexity is often correlated with lines of code. Unfortunately, a literature survey of Shihab [193] shows that as much as 63% of prior defect studies do not mitigate correlated metrics prior to constructing defect models. Yet, little is known about the impact of correlated metrics on the interpretation of defect models.

Approach. To address RQ1, we analyze the impact of correlated metrics on the interpretation of defect models along with three dimensions, i.e., (1) the *consistency* of the top-ranked metric, (2) the *level of discrepancy* of the top-ranked metric, and (3) the *direction* of the ranking of metrics for all non-correlated metrics between mitigated and non-mitigated models.

To do so, we start from mitigated datasets (see Section 3.3.2). We first identify the top-ranked metric for each of the 9 studied interpretation techniques. We use VarClus to select only one of the metrics that is correlated with the top-ranked metric in order to generate non-mitigated datasets. We then append the correlated metric to the first position of the specification of the mitigated models. Thus, the specification for the mitigated models is $y \sim m_{\text{top\_ranked}} + ...$, while the specification for the non-mitigated models is $y \sim m_{\text{correlated}} + m_{\text{top\_ranked}} + ...$, where $m_{\text{correlated}}$ is the metric that is correlated with the top-ranked metric ($m_{\text{top\_ranked}}$). For each of the mitigated and non-mitigated datasets, we construct defect models (see Section 3.3.3) and apply the 9 studied model interpretation techniques (see Section 3.3.4).

To analyze the *consistency* and the *level of discrepancy* of the top-ranked metric, we compute the difference in the ranks of the top-ranked metric between mitigated and non-mitigated models. For example, if a metric $m_{\text{top\_ranked}}$ appears in the 1st rank in both of mitigated

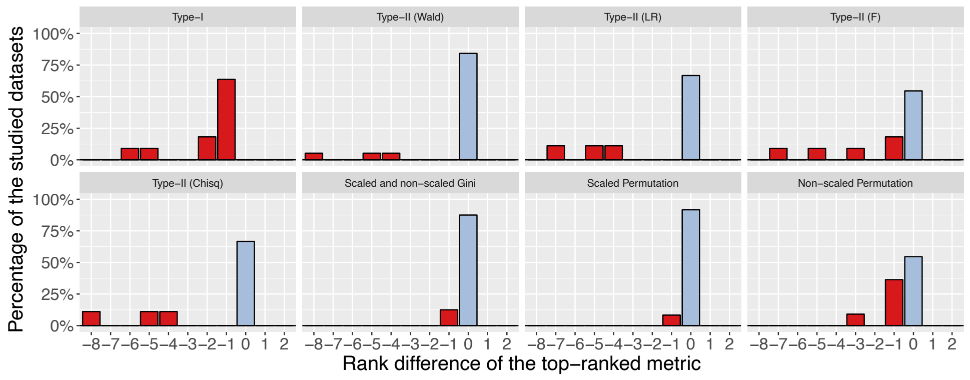


Figure 3.3 The percentage of the studied datasets for each difference in the ranks between the top-ranked metric of the models that are constructed using the mitigated and non-mitigated datasets. The light blue bars represent the consistent rank of the metric between mitigated and non-mitigated models, while the red bars represent the inconsistent rank of the metric between mitigated and non-mitigated models.

and non-mitigated models, then the metric would have a rank difference of 0. However, if $m_{\text{top\_ranked}}$ appears in the 3rd rank of a non-mitigated model and appears in the 1st rank of a mitigated model, then the rank difference of $m_{\text{top\_ranked}}$ would be 2. The consistency of the top-ranked metric measures the percentage of the studied datasets that the top-ranked metric appears at the 1st rank in both mitigated and non-mitigated models. On the other hand, the level of discrepancy of the top-ranked metric measures the highest rank difference of the top-ranked metric between mitigated and non-mitigated models.

To analyze the *direction* of the ranking of metrics for all non-correlated metrics between mitigated and non-mitigated models, we start with the ranking of important metrics that appear in both mitigated and non-mitigated models. We then apply a Spearman rank correlation test (ρ) to compute the correlation between ranks of all non-correlated metrics between mitigated and non-mitigated models. The Spearman correlation coefficient (ρ) of 1 indicates that the ranking of non-correlated metrics between mitigated and non-mitigated models is in the same direction. On the other hand, the Spearman correlation coefficient (ρ) of -1 indicates that the ranking of non-correlated metrics between mitigated and non-mitigated models is in the reverse direction. Since the produced ranking of each model and defect dataset may be different, the use of a weighted Spearman rank correlation test may lead these rankings to be weighted differently. Thus, we select a traditional Spearman rank correlation test for our study.

Results. **ANOVA Type-I produces the lowest consistency of the top-ranked metric between mitigated and non-mitigated models.** We expect that the top-ranked metric in the mitigated model will remain as the top-ranked metric in the non-mitigated model. Unfortunately, Figure 3.3 shows that this expectation does not hold true in any of the studied datasets for ANOVA Type-I. Figure 3.3 shows that, for ANOVA Type-I, none of the top-ranked metric that appears in the 1st rank of mitigated models also appears in the 1st rank of non-mitigated models. On the other hand, we find that the top-ranked metric of mitigated models appears at the top-ranked metric in non-mitigated models for 84%, 67%, 55%, and 67% of the studied datasets for ANOVA Type-II (Wald), Type-II (LR), Type-II (F), Type-II (Chisq), respectively. We suspect that the impact of correlated metrics on the interpretation of Type-I has to do with the sequential nature of the calculation of the Sum of Squares, i.e., Type-I attributes as much variance as it can to the first metric before attributing residual variance to the second metric in the model specification.

ANOVA Type-I and Type-II produce the highest level of discrepancy of the top-ranked metrics between mitigated and non-mitigated models. Figure 3.3 shows that the rank difference for ANOVA Type-I and Type-II can be up to -6 and -8, respectively. In other words, we find that the top-ranked metric in mitigated models appear at the 7th rank and the 9th rank in non-mitigated models for ANOVA Type-I and Type-II, respectively. We suspect that the highest level of discrepancy (i.e., the largest rank difference) for ANOVA Type-I and Type-II has to do with the sharply drop of the importance scores when correlated metrics are included in defect models.

For ANOVA Type-I and Type-II, correlated metrics have the largest impact on the direction of the ranking of metrics of defect models. For ANOVA Type-I, we find that the Spearman correlation coefficients range from -0.1 to 0.84. For ANOVA Type-II, we find that the Spearman correlation coefficients range from 0.1 to 1. A low value of the Spearman correlation coefficients in ANOVA techniques indicates that the direction of the ranking of metrics for all non-correlated metrics is varied in each rank, suggesting that the ranking of non-correlated metrics is inconsistent across mitigated and non-mitigated models.

Gini and Permutation Importance approaches produce the higest consistency and the lowest level of discrepancy of the top-ranked metric, and have the least impact on the direction of the ranking of metrics between mitigated and non-mitigated models. Figure 3.3 shows that the top-ranked metric of mitigated models appears at the top-ranked metric in non-mitigated models for 88%, 92%, and 55% of the studied datasets for Gini Importance, and scaled and non-scaled Permutation Importance, respectively. Figure 3.3 also shows that the rank difference for Gini and Permutation Importance is as low as -1 and -3, respectively. Furthermore, we find that the Spearman correlation coefficients range from 0.9 to 1 for Gini and Permutation Importance. These findings suggest that the lower impact that correlated metrics have on Gini and Permutation Importance than ANOVA techniques have to do with the random process for constructing multiple trees and the calculation of importance scores for a random forest model. For example, the random process of random forest may generate trees that are constructed without correlated metrics. In addition, the averaging of the importance scores from multiple trees may decrease the negative impact of correlated metrics on trees that are constructed with correlated metrics for random forest models.

ANOVA Type-I and Type-II often produce the lowest consistency and the highest level of discrepancy of the top-ranked metric, and have the highest impact on the direction of the ranking of metrics between mitigated and non-mitigated models when compared to Gini and Permutation Importance. This finding highlights the risks of not mitigating correlated metrics in the ANOVA analyses of prior studies.

(RQ2) After removing all correlated metrics, how consistent is the interpretation of defect models among different model specifications?

Motivation. Our motivating analysis and the results of RQ1 confirm that the ranking of the top-ranked metric substantially changes when the ordering of correlated metrics in a model specification is rearranged, suggesting that correlated metrics must be removed. However, after removing correlated metrics, little is known if the interpretation of defect models would become consistent when rearranging the ordering of metrics.

Approach. To address RQ2, we analyze the ranking of the top-ranked metric of the models that are constructed using different ordering of metrics from mitigated datasets. To do so, we start from mitigated datasets that are produced by Section 3.3.2. For each of the datasets, we construct defect models (see Section 3.3.3) and apply the 9 studied model interpretation techniques (see Section 3.3.4) in order to identify the top-ranked metric according to each technique. Then, we regenerate the models where the ordering of metrics is rearranged—the top-ranked metric is at each position from the first to the last for each dataset. Finally, we compute the percentage of datasets where the ranks of the top-ranked metric are inconsistent among the rearranged datasets.

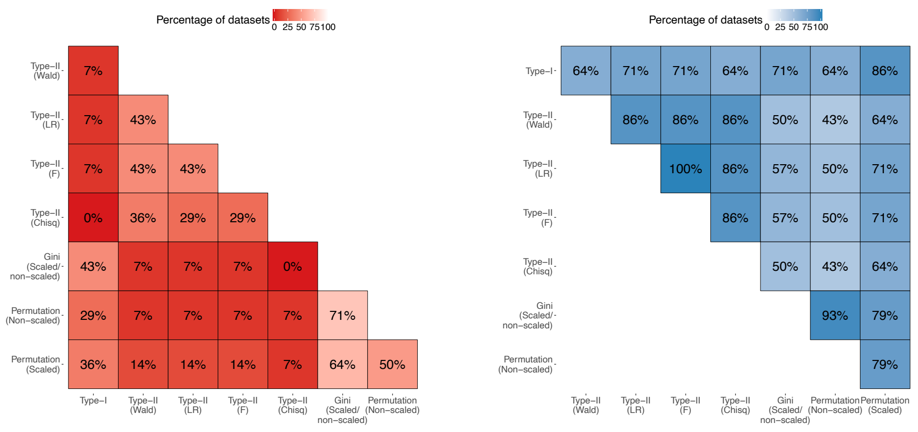
Results. **After removing correlated metrics, the top-ranked metrics according to Type-II, Gini Importance, and Permutation Importance are consistent. However, the top-ranked metric according to Type-I is still inconsistent regardless of the ordering of metrics.** We find that Type-II, Gini Importance, and Permutation Importance produce a stable ranking of the top-ranked metric for all of the studied datasets regardless of the ordering of metrics.

On the other hand, ANOVA Type-I is the only technique that produces an inconsistent ranking of the top-ranked metric. We find that for 71% of the studied datasets, ANOVA Type-I produces an inconsistent ranking of the top-ranked metric when the ordering of metrics is rearranged. We expect that the consistency of the ranking of the top-ranked metrics can be improved by increasing the strictness of the correlation threshold of the variable clustering analysis (VarClus). Thus, we repeat the analysis using stricter thresholds of the variable clustering analysis (VarClus). We use Spearman correlation coefficient ($|\rho|$) threshold values of 0.5 and 0.6. Unfortunately, even if we increase the strictness of the correlation threshold value, Type-I produces the inconsistent ranking of the top-ranked metric for 43% and 50% of the studied datasets, for the threshold of 0.5 and 0.6, respectively.

The inconsistent ranking of the top-ranked metric according to Type-I has to do with the sequential nature of the calculation of the Sum of Squares. In other words, Type-I attributes the importance scores as much as it can to the first metric before attributing the scores to the second metric in the model specification. Thus, Type-I is sensitive to the ordering of metrics.

After removing all correlated metrics, the top-ranked metric according to ANOVA Type-II, Gini Importance, and Permutation Importance are consistent. However, the top-ranked metric according to ANOVA Type-I is inconsistent, since the ranking of metrics is impacted by its order in the model specification when analyzed using ANOVA Type-I (which is the default analysis for the `glm` model in R and is commonly-used in prior studies). This finding suggests that ANOVA Type-I must be avoided even if all correlated metrics are removed.

(RQ3) After removing all correlated metrics, how consistent is the interpretation of defect models among the studied interpretation techniques?



(a) The top-ranked metric for non-mitigated models.

(b) The top-ranked metric for mitigated models.

Figure 3.4 The percentage of datasets where the top-ranked metric is consistent between the two studied model interpretation techniques. While the lower-left side of the matrix (i.e., red shades) shows the percentage before removing correlated metrics, the upper-right side of the matrix (i.e., blue shades) shows the percentage after removing correlated metrics.

Motivation. The findings of prior work often rely heavily on one model interpretation technique [56, 95, 147, 150, 196, 211]. Therefore, the findings of prior work may pose a threat to construct validity, i.e., the findings may not hold true if one uses another interpretation technique. Thus, we set out to investigate the consistency of the top-ranked and top-3 ranked metrics after removing correlated metrics.

Approach. To address RQ3, we start from mitigated datasets that are produced by Section 3.3.2 and non-mitigated datasets (i.e., the original datasets). We compare the two rankings that are produced from mitigated and non-mitigated models using the 9 interpretation techniques for each of the studied datasets. Then, we compute the percentage of datasets where the top-ranked metric is consistent among the studied model interpretation techniques. Moreover, we also compute the percentage of datasets where at least one of the top-3 ranked metrics is consistent among the studied model interpretation techniques. Finally, we present the results using a heatmap (as shown in Figure 3.4) where each cell indicates the percentage of datasets which the top-ranked metric is consistent among the two studied model interpretation techniques.

Results. **Before removing all correlated metrics, we find that the studied model interpretation techniques do not tend to produce the same top-ranked metric.** We observe that the consistency of the ranking of metrics across learning techniques (i.e., logistic regression and random forest) is as low as 0%-43% for the top-ranked metric (Figure 3.4a). Furthermore, according to the lower-left side of the matrix of the Figure 3.4a, we find that, before removing correlated metrics, the top-ranked metric of Type-II (Chisq) is inconsistent with the top-ranked metrics of Type-I and Gini Importance for all of the studied datasets.

After removing all correlated metrics, we find that the consistency of the ranking of metrics among the studied interpretation techniques is improved by 15%-64% for the top-ranked metric and 21%-71% for the top-3 ranked metrics, respectively. In particular, we observe that the consistency of the ranking of metrics across learning techniques is improved by 28%-50% for the top-1 ranked metrics and 43%-71% for the top-3 ranked metrics, respectively. Most importantly, we find that scaled Permutation Importance achieves the highest consistency of the ranking of metrics across learning techniques. This finding highlights the benefits of removing correlated metrics on the interpretation of defect models—the conclusions of studies that rely on one interpretation technique may not pose a threat after mitigating correlated metrics.

After removing all correlated metrics, we find that the consistency of the ranking of metrics among the studied interpretation techniques is improved by 15%- 64% for the top-ranked metric and 21%-71% for the top-3 ranked metrics, respectively, highlighting the benefits of removing all correlated metrics on the interpretation of defect models, i.e., the conclusions of studies that rely on one interpretation technique may not pose a threat after mitigating correlated metrics.

(RQ4) Does removing all correlated metrics impact the performance and stability of defect models?

Motivation. The results of RQ1 show that correlated metrics have a negative impact on the interpretation of defect prediction models, while the results of RQ2 and RQ3 show the benefits of removing correlated metrics on the interpretation of defect models. Thus, removing correlated metrics is highly recommended. However, removing correlated metrics

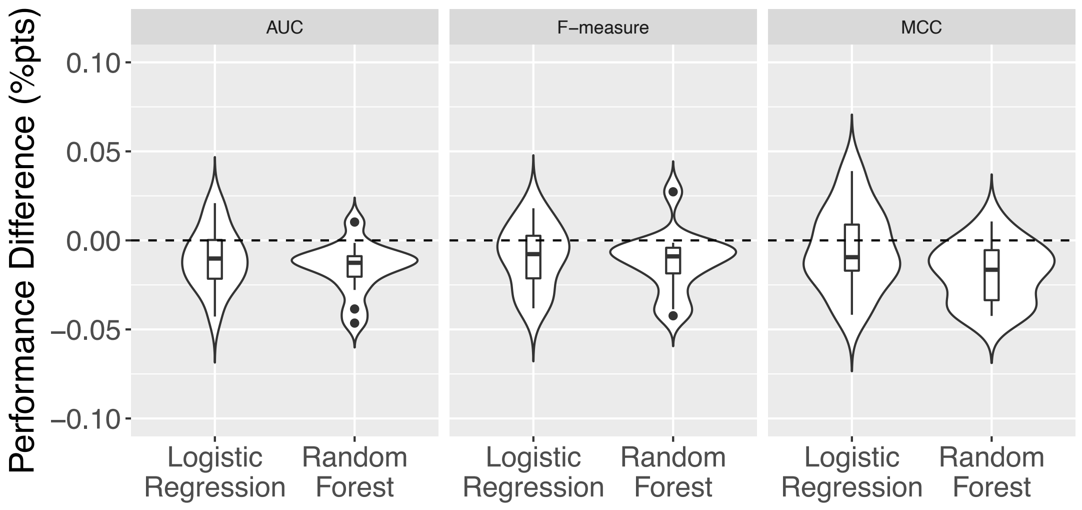


Figure 3.5 The distributions of the performance difference (% pts) between non-mitigated and mitigated models for each of the studied datasets.

may pose a risk to the performance and stability of defect models. Yet, little is known if removing such correlated metrics impacts the performance and stability of defect models.
Approach. To address RQ4, we first start from the AUC, F-measure and MCC performance estimates and their performance stability of the non-mitigated and mitigated models. The performance stability is measured by a standard deviation of the performance estimates as produced by 100 iterations of the out-of-sample bootstrap for each model. We then quantify the impact of removing all correlated metrics by measuring the performance difference (i.e., the arithmetic difference between the performance of the non-mitigated and mitigated models) and the stability ratio (i.e., the ratio of the *S.D.* of performance estimates of non-mitigated to mitigated models, $\frac{S.D.\text{ non-mitigated models}}{S.D.\text{ mitigated models}}$). Furthermore, in order to measure the effect size of the impact, we measure Cliff's $|\delta|$ effect size for the performance difference and the stability ratio across the non-mitigated and mitigated models.
Results. **Removing all correlated metrics impacts the AUC, F-measure, and MCC performance of defect models by less than 5 percentage points.** Figure 3.5 shows that the distributions of the performance difference between the models that are constructed using non-mitigated and mitigated datasets are centered at zero. In addition, our Cliff's $|\delta|$ effect size test shows that the differences between the models that are constructed using mitigated and non-mitigated datasets are negligible to small for the AUC, F-measure, and MCC measures. However, the performance difference of 5 percentage points may be very important for safety-critical software domains. Thus, researchers and practitioners should remove correlated metrics with care.

Removing all correlated metrics yields a negligible difference (Cliff's $|\delta|$) for the stability of the performance of defect models. Figure 3.6 shows that the distributions of the stability ratio of the models that are constructed using non-mitigated and mitigated datasets are centered at one (i.e., there is little difference in model stability after removing all

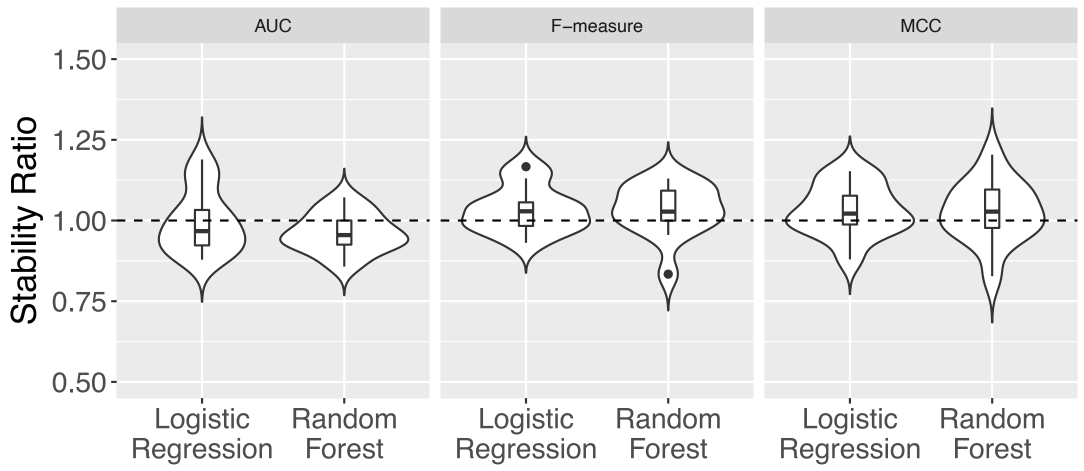


Figure 3.6 The distributions of the stability ratio of non-mitigated to mitigated models for each of the studied datasets.

correlated metrics). Moreover, our Cliff's $|\delta|$ effect size test shows that the difference of the stability ratio between the models that are constructed using mitigated and non-mitigated datasets is negligible.

Removing all correlated metrics impacts the AUC, F-measure, and MCC performance of defect models by less than 5 percentage points, suggesting that researchers and practitioners should remove correlated metrics with care especially for safety-critical software domains.

3.5 Practical Guidelines

In this section, we offer practical guidelines for future studies. When the goal is to derive sound interpretation from defect models:

(1) **Ones must mitigate correlated metrics prior to constructing a defect model, especially for ANOVA analyses**, since RQ1 shows that (1) ANOVA Type-I and Type-II often produce the lowest consistency and the highest level of discrepancy of the top-ranked metric, and have the highest impact on the direction of the ranking of metrics between mitigated and non-mitigated models. On the other hand, the results of RQ2 and RQ3 show that removing all correlated metrics (2) improves the consistency of the top-ranked metric regardless of the ordering of metrics; and (3) improves the consistency of the ranking of metrics among to the studied interpretation techniques, suggesting that correlated metrics must be mitigated. However, the results of RQ4 show that the removal of such correlated metrics impacts the model performance by less than 5 percentage points, suggesting that researchers and practitioners should remove correlated metrics with care especially for safety-critical software domains.

(2) **Ones must avoid using ANOVA Type-I even if all correlated metrics are removed**, since RQ2 shows that Type-I produces an inconsistent ranking of the top-ranked metric when the orders of metrics are rearranged, indicating that Type-I is sensitive to the ordering of metrics even when removing all correlated metrics. Instead, researchers should opt to **use ANOVA Type-II and Type-III for additive and interaction logistic regression models, respectively**. Furthermore, the scaled Permutation Importance approach is recommended for random forest since RQ3 shows that such approach achieves the highest consistency across learning techniques.

We would like to emphasize that mitigating correlated metrics is not necessary for 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 correlated metrics must be mitigated when the goal is to derive sound interpretation from defect models that are trained with correlated metrics (especially for ANOVA Type-I). On the other hand, if the goal of the study is to produce highly-accurate prediction models, one might prioritize their resources on improving the model performance rather than mitigating correlated metrics. Thus, feature selection and dimensionality reduction techniques can be considered to mitigate irrelevant and correlated metrics, and improve model performance. Finally, similar to Zeller *et al.* [234], we also would like to emphasize that correlations do not imply causations. To show causation, one needs to show that changing the cause (e.g., code complexity) also changes the effect (e.g., code quality). One also needs a theory grounded from domain knowledge to explain causation.

3.6 Threats to Validity

Construct Validity. In this work, we only construct regression models in an additive fashion ($y \sim m_1 + ... + m_n$), since metric interactions (i.e., the relationship between each of the two interacting metrics depends on the value of the other metrics) (1) are rarely explored in software engineering (except in [191]); (2) must be statistically insignificant (e.g., absence) for ANOVA Type-II test [28, 43]; and (3) are not compatible with random forest [21] which is one of the most commonly-used analytical learners in software engineering. On the other hand, the importance score of the metric produced by ANOVA Type-III is evaluated after all of the other metrics and all metric interactions of the metric under examination have been accounted for. Thus, if metric interactions are significantly present, one should use ANOVA Type-III and avoid using ANOVA Type-II. Due to the same way in which the importance scores of metrics according to ANOVA Type-II and Type-III are calculated in a hierarchical nature for an additive model, we would like to note that the importance scores of metrics according to ANOVA Type-II and Type-III are the same for such additive models.

Plenty of prior work show that the parameters of classification techniques have an impact on the performance of defect models [49, 106, 133, 134, 210, 213]. While we use a default `ntree` value of 100 for random forest models, recent studies [210, 213] show that the parameters of random forest are insensitive to the performance of defect models. Thus, the parameters of random forest models do not pose a threat to validity of our study.

Recent work point out that the selection [98, 210] and the quality [233] of datasets dataset selection might impact conclusions of a study. Thus, our conclusions may alter when changing a set of the studied datasets. Prior work [226] raise concerns related to the imbalance nature of data in software systems that should be handled with care, i.e., only a small percentage of software modules are defective. While we carefully mitigate correlations among software metrics, we neither re-balance nor transform such studied datasets to preserve the original distributions of software metrics. Finally, Tantithamthavorn *et al.* [212] point out that randomness may introduce bias to the conclusions of a study. To mitigate this threat and ensure that our results are reproducible, we set a random seed in every step in our experiment design.

Internal Validity. Recent research uses ridge regression to construct defect models on the dataset that contains correlated metrics [176]. However, our additional analyses show that ridge regression improves the performance of defect model when comparing to logistic regression, yet produces a misleading importance ranking of metrics. We observe that metrics that are highly correlated appear at different ranks. This observation highlights the importance of mitigating correlated metrics when interpreting defect models.

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

External Validity. The analyzed datasets are part of several corpora (e.g., NASA and PROMISE) of systems that span both proprietary and open source domains. However, we studied a limited number of defect datasets, particularly for proprietary datasets. Thus, the results may not generalize to other datasets and domains. Additional replication studies are needed.

In our study, we exclude (1) datasets that are not representative of common practice or (2) datasets that would not realistically benefit from our analysis (e.g., datasets that most of the software modules are defective) with the selection criteria of the studied datasets. Nevertheless, our proposed approaches are applicable to any dataset. Practitioners are encouraged to explore our approaches on their own datasets with their own peculiarities.

The conclusions of our study rely on one defect prediction scenario (i.e., within-project defect models). However, there are a variety of defect prediction scenarios in the literature (e.g., cross-project defect prediction [27, 237], just-in-time defect prediction [96], heterogenous defect prediction [157]). Therefore, the practical guidelines may differ for other scenarios. Thus, future research should revisit our study in other scenarios of defect models.

3.7 Conclusions

In this chapter, we set out to investigate (1) the impact of correlated metrics on the interpretation of defect models. After removing correlated metrics, we investigate (2) the consistency of the interpretation of defect models; and (3) its impact on the performance and stability of defect models. Through a study of 14 publicly-available defect datasets of systems that span both proprietary and open source domains, we conclude that (1) correlated metrics have the largest impact on the consistency, the level of discrepancy, and the direction of the ranking of metrics, especially for ANOVA techniques. On the other hand, we find that removing all correlated metrics (2) improves the consistency of the produced rankings regardless of the ordering of metrics (except for ANOVA Type-I); (3) improves the consistency of ranking of metrics among the studied interpretation techniques; (4) impacts the model performance by less than 5 percentage points.

Based on our findings, we offer practical guidelines for future studies. When the goal is to derive sound interpretation from defect models:

1. Ones must mitigate correlated metrics prior to constructing a defect model, especially for ANOVA analyses.
2. Ones must avoid using ANOVA Type-I even if all correlated metrics are removed.

Due to the variety of the built-in interpretation techniques and their settings, our paper highlights the essential need for future research to report the exact specification of their models and settings of the used interpretation techniques.

3.7.1 Chapter Remarks

In this chapter, we investigate the impact of correlated metrics on the explanation of defect models. The experimental results show that correlated metrics, indeed, have an impact on the explanation of defect models. The experimental results also lead us to suggest that, after mitigating correlated metrics, ones should use the ANOVA Type-II technique to explain logistic regression models, while using the scaled Permutation Importance technique to explain random forests models. However, to mitigate correlated metrics, commonly-used correlation analysis techniques involve manual processes. For example, for VarClus, the technique requires manual inspection to select a representative metric from each group of correlated metrics. Thus, in the next chapter, we investigate techniques that automatically mitigate correlated metrics when explaining defect models.

Chapter 4

Automatically Mitigate Correlated Metrics when Explaining Defect Models

An earlier version of the work in this chapter appears in the International Conference on Software Maintenance and Evolution (ICSME 2018) [88], while its extension appears in the Springer Journal of Empirical Software Engineering (EMSE) [91].

4.1 Introduction

Defect models are statistical or machine learning models that are used to investigate the impact of software metrics (e.g., lines of code) on defect-proneness and to identify defect-prone software modules. The explanation of defect models is used to validate hypotheses to develop empirical theories related to software quality, which are essential to chart quality improvement and plans.

The conclusions of defect models heavily rely on the studied software metrics. However, software metrics often have strong correlation among themselves [54, 86, 211, 235] and some metrics are irrelevant to defect models [137, 192]. For example, my previous work *et al.* [86] find that many metrics in defect datasets are *correlated* (e.g., the `branch_count` metric is linearly proportional to the `decision_count` metric in some NASA datasets). Shepperd *et al.* [192] find that many metrics in the NASA datasets are *irrelevant to defect models* (e.g., constant metrics).

To address these concerns, feature selection techniques are often applied in the defect prediction domain [5, 34, 40, 97, 140, 160, 198]. However, prior work [5, 34, 40, 97, 140] often relies on one feature selection technique which may pose a threat to the construct validity, i.e., conclusions may not hold true if another feature selection technique is applied. In practice, feature selection techniques should be applied only on training samples to avoid producing optimistically biased performance estimates and explanation [117]. Yet, the conclusions (e.g., the most important metric) that are derived from defect models may be *unreliable* (i.e., produce misleading importance rankings of metrics) if the produced subsets of metrics are (1) inconsistent among feature selection techniques, (2) inconsistent among different training samples, (3) inconsistent among repetitions, and (4) inconsistent among different model specifications.

In this chapter, we investigate 11 commonly-used feature selection techniques and our own contribution AutoSpearman along 5 dimensions: (1) the consistency of the produced subsets of metrics; (2) the correlation of the produced subsets of metrics; (3) the performance; (4) the computational cost; and (5) the impact on the explanation.

Through an empirical investigation of 13 publicly-available defect datasets of systems that span both proprietary and open source domains, we address the following eight research questions:

(RQ1) Do feature selection techniques consistently produce the same subset of metrics when applied on the same training sample?
Feature selection techniques produce inconsistent subsets of metrics. As many as 94-100% of the metrics are inconsistently selected among the studied feature selection techniques, suggesting that the conclusions of prior work could be altered when another feature selection technique is applied.

(RQ2) Do feature selection techniques consistently produce the same subset of metrics when applied across different training samples?

Surprisingly, when applying commonly-used feature selection techniques to different training samples from the same dataset, we find that 31-94% of the selected metrics are inconsistent, suggesting that post-hoc multiple comparison tests (e.g., a Scott-Knott test) should not be applied to identify the most important metrics when explaining defect models. On the other hand, we observe that AutoSpearman yields the highest consistency and leads to absolute improvements of consistency up to 86% compared to other studied feature selection techniques.

(RQ3) **Do feature selection techniques consistently produce the same subset of metrics when they are applied repeatedly?**
The Recursive Feature Elimination technique is the only technique that produces inconsistent subsets of metrics when repeated with different random seeds. Such inconsistency amounts to 62% and 13% of the subsets of metrics for logistic regression and random forest, respectively. This finding indicates that even if Recursive Feature Elimination is applied on the same training sample, simply altering a random seed can lead the technique to produce different subset of metrics.

(RQ4) **Do feature selection techniques consistently produce the same subset of metrics when reordering the model specification of a defect model?**
Regardless of the search directions, Stepwise Regression is the only studied feature selection technique that produces inconsistent subsets of metrics when applied across model specifications.

(RQ5) **Do feature selection techniques mitigate correlated metrics?**
AutoSpearman is the only studied feature selection technique that can mitigate correlated metrics, i.e., collinearity and multicollinearity. We observe that other studied feature selection techniques produce up to 100% of subsets of metrics with collinearity and multicollinearity, suggesting that the explanation of defect models constructed using the subsets of metrics that are produced by all of the studied feature selection techniques (except for AutoSpearman) may be misleading.

(RQ6) **What is the impact of feature selection techniques on the performance of defect models?**
The studied feature selection techniques impact the performance of defect models by up to 5%pts. The results of the Mann-Whitney U test and the Kruskal-Wallis H test suggest that all of the performance differences across feature selection techniques are not statistically significant with the p-values of above 0.05. The results of the Cliff's $|\delta|$ effect size test also confirm that such performances are negligible to small for the AUC, F-measure, and MCC measures.

(RQ7) What is the computational cost of applying feature selection techniques?
The computational cost of filter-based feature selection techniques and AutoSpearman is cheap, while such cost is expensive for wrapper-based feature selection techniques and the consistency-based feature selection technique. The computational cost of wrapper-based feature selection techniques is as high as 7 hours and 30 minutes for RFE-RF to find the best subset of metrics from one training sample of the Proprietary 2 defect dataset. Such expensive computation cost makes the application of wrapper-based feature selection techniques undesirable, particularly, when validating with model validation techniques that require several repetitions (e.g., bootstrap validation technique).

(RQ8) Do correlation threshold values have an impact on the explanation of defect models?
No. Indeed, 0–62% of the studied defect datasets produces different most important metrics among correlation threshold values. However, such different most important metrics are highly correlated with the spearman correlation of 0.84–1, suggesting that correlation threshold values do not impact the explanation of defect models.

Our results lead us to conclude that the subsets of metrics produced by the commonly-used automated feature selection techniques are (1) inconsistent among automated feature selection techniques; (2) inconsistent among training samples; (3) inconsistent among repetitions; and (4) highly-correlated. Since we find that the subsets of metrics produced by the commonly-used feature selection techniques (except for AutoSpearman) are often inconsistent and correlated, these techniques should be avoided when explaining defect models.

4.1.1 Chapter Organisation

Section 4.2 presents the research questions which are formulated from related work. Section 4.3 describes the design and the setup of our study, while Section 4.4 presents the results with respect to the eight research questions. Section 4.5 discusses the trends of the commonly-selected correlated metrics. Section 4.6 elaborates on the threats to the validity of our study. Finally, Section 4.7 draws conclusions.

4.2 Related Work & Research Questions

Feature selection has been widely used in software engineering to remove *irrelevant* metrics (i.e., metrics that do not share a strong relationship with the outcome) [137, 192] and *correlated* metrics (i.e., metrics that share a strong correlation with one or more metrics) [54, 86, 211, 235]. While a variety of feature selection techniques has been proposed in literature, little is known about the best feature selection techniques for model explanation.

Many ML research efforts propose feature selection techniques and investigate the impact of such techniques on the performance of prediction models [35, 64, 105]. For example, Dash *et al.* [35] introduce the consistency-based feature selection technique and investigated the impact of 5 search strategies of such technique on 10 benchmark datasets as provided by the UC Irwin Machine Learning repository [19] using C4.5 and back-propagation neural network. Mark Hall Hall and Smith [64] proposes the correlation-based feature selection technique and evaluate the proposed technique comparing with a wrapper-based feature selection technique on 15 benchmark datasets as provided by the UC Irwin Machine Learning repository [19] using 3 classification techniques (i.e., IB1, Naive Bayes, and C4.5). Kohavi *et al.* Kohavi and John [105] propose the wrapper-based feature selection and investigate the model performance produced by the technique comparing with 4 other filter-based feature selection techniques on 14 benchmark datasets as provided by the UC Irwin Machine Learning repository [19] using 5 classification techniques (i.e., ID3, Naive Bayes, and C4.5). Nevertheless, little is known whether these findings hold true for the context of software engineering.

Prior studies investigate the impact of feature selection techniques on the performance of defect models [53, 124, 161, 185, 231] (see Table 4.1). For example, Ghotra *et al.* Ghotra et al. [53] investigate the impact of 29 feature selection techniques (as provided by Weka [50]) on 18 PROMISE [139] and NASA [192] datasets using 21 classification techniques. Lu *et al.* Lu et al. [124] investigate the impact of 4 feature selection techniques (i.e., Information Gain, Correlation-based, Forward selection, and Backward selection) on 3 releases of Eclipse datasets as provided by Zimmermann *et al.* Zimmermann et al. [238] using random forests. Osman *et al.* Osman et al. [161] investigate the impact of correlation-based and wrapper-based feature selection techniques on 5 defect datasets as provided by D'Ambros *et al.* D'Ambros et al. [34] using 5 classification techniques. Rodriguez *et al.* Rodríguez et al. [185] investigate the impact of 5 feature selection techniques on 5 PROMISE datasets [139] using naive bayes and C4.5. Xu *et al.* Xu et al. [231] investigate the impact of 19 feature selection techniques on 11 NASA [192] and 4 AEEEM [34] datasets using random forests. Yet, no prior work has investigated the impact of feature selection techniques when explaining defect models. Thus, in this study, we investigate 11 commonly-used feature selection techniques and our own contribution AutoSpearman along five dimensions: (1) the consistency of the produced subsets of metrics; (2) the correlation of the produced subsets of metrics; (3) the performance; (4) the computational cost; and (5) the impact on the explanation.

The conclusions of prior work often rely on one feature selection technique [5, 34, 40, 97, 140] which may pose a threat to the construct validity, i.e., conclusions may not hold true if another feature selection technique is applied. Nevertheless, little is known about whether feature selection techniques produce the same subset of metrics among training samples and among feature selection techniques. We therefore formulate the following research question:

Table 4.1 An overview comparison of our study with respect to prior work.

Study	#Datasets	#Feature Selection Techniques	#Classification Techniques	Analyses
Ghotra et al. [53]	18	29	21	Performance (AUC)
Lu et al. [124]	3	5	1	Performance (Precision, Recall, Accuracy, and AUC)
Osman et al. [161]	5	2	5	Consistency (Agreement on the selected metrics) and Performance (RMSE)
Rodríguez et al. [185]	5	5	2	Performance (Accuracy and F-measure)
Xu et al. [231]	15	20	1	Performance (AUC)
Our study	13	12	2	Consistency (Agreement on the selected metrics), Correlation (Collinearity and Multicollinearity), Performance (AUC), Computational Cost (Time), and Explanation (Consistency and Correlation of the most important metrics)

(RQ1) Do feature selection techniques consistently produce the same subset of metrics when applied on the same training sample?

In practice, *feature selection techniques should only be applied on training samples* because of the unavailability of defect labels in testing samples. Since training samples are often randomly generated from model validation techniques (e.g., out-of-sample bootstrap validation or 10-folds cross-validation), feature selection techniques may produce different subsets of metrics for each training sample. Different subsets of metrics among training samples may pose a critical threat to the validity when analysing and identifying the most important metrics. For example, prior work often applies a post-hoc multiple comparison test (e.g., a Scott-Knott test) on the distributions of importance scores to identify statistically distinct ranks of the most important metrics [85, 208, 219]. Nevertheless, little is known about whether feature selection techniques produce the same subset of metrics among training samples. We therefore formulate the following research question:

(RQ2) Do feature selection techniques consistently produce the same subset of metrics when applied across different training samples?

Some feature selection techniques involve randomisation in the process of sampling data for validation to produce the best subset of metrics. Such randomisation may lead a feature selection technique to produce different subsets of metrics when the technique is applied repeatedly. Nevertheless, little is known about whether feature selection techniques produce the same subset of metrics when they are applied repeatedly. We therefore formulate the following research question:

(RQ3) Do feature selection techniques consistently produce the same subset of metrics when they are applied repeatedly?

In theory, feature selection techniques should produce the same subset of metrics regardless of the ordering of metrics in a model specification, e.g., $\text{FS}(y \sim x_1 + x_2) = \text{FS}(y \sim x_2 + x_1)$. However, in practice, some feature selection techniques consider whether to include (or exclude) a metric in the output subset of metrics based on the ordering of metrics in a model specification. Such process may lead a feature selection technique to produce different subsets of metrics, if different model specifications are applied. Nevertheless, little is known about whether feature selection techniques produce the same subset of metrics across different model specifications (when reordering the model specification of a defect model). We therefore formulate the following research question:

(RQ4) Do feature selection techniques consistently produce the same subset of metrics when reordering the model specification of a defect model?

The conclusions of prior defect studies rely on the usage of built-in model explanation techniques of classification techniques (e.g., ANOVA for logistic regression, and Breiman's Variable Importance for random forest). However, recent work points out that such explanation techniques are sensitive to correlated metrics [10, 85, 203, 211, 235]. For example, my previous work *et al.* [85] show that the explanation of ANOVA Type-I can be altered by simply rearranging the model specification (e.g., from $y \sim m_1 + m_2$ to $y \sim m_2 + m_1$ if m_1 and m_2 are correlated). Despite posing a threat to the validity of previous work's conclusion, little is known about whether feature selection techniques mitigate correlated metrics. We therefore formulate the following research question:

(RQ5) Do feature selection techniques mitigate correlated metrics?

Prior research effort has shown the benefits of applying feature selection techniques to defect prediction models [53, 124, 231]. For example, Ghotra *et al.* [53], Lu *et al.* [124], and Xu *et al.* [231] investigate the impact of feature selection techniques on the performance of defect models. Nevertheless, their findings on the best feature selection techniques (i.e., lead to the model with the highest model performance) are not always consistent. Prior studies also often use different defect datasets. While our goal is model explanation, practitioners may question the explanation of inaccurate defect models. We set out to investigate the impact of feature selection techniques on the performance of defect models, particularly, on defect datasets from which we can accurately derive explanation through the following research question:

(RQ6) What is the impact of feature selection techniques on the performance of defect models?

Due to the variation in metric selection, feature selection techniques may incur different computational cost. For example, filter-based feature selection techniques search for the best subset of metrics regardless of model construction. Wrapper-based feature selection techniques rely on classification techniques to assess each subset of metrics and find the best subset of metrics. Thus, we set out to investigate the computational cost of feature selection techniques through the following research question:

(RQ7) What is the computational cost of applying feature selection techniques?

Literature has suggested a variety of correlation threshold values to indicate strong correlations among metrics. For example, Kraemer *et al.* [107] suggested the use of Spearman correlation of 0.7 to indicate strong correlations between metrics. Similarly, Hinkle *et al.* [76] suggested that the Spearman correlation of 0.7–0.9 indicates strong correlations between metrics. Mason *et al.* [126], Fox *et al.* [44], and Hair *et al.* [61] suggested the use of VIF threshold values of 3, 5, and 10 to indicate strong correlations between metrics, respectively. Such contradictory suggestion about the correlation threshold may produce different subsets of metrics and explanation of defect models. Particularly, a stricter correlation threshold value identifies a higher number of correlated metrics and produces a smaller subset of metrics. On the other hand, a more relaxed correlation threshold value identifies a less number of correlated metrics and produces a larger subset of metrics. Thus, we set out to investigate the impact of correlation threshold values on the explanation of defect models through the following research question:

(RQ8) Do correlation threshold values have an impact on the explanation of defect models?

4.3 Experimental Design

4.3.1 Studied Datasets

In selecting the studied datasets, we identify four important criteria that need to be satisfied:

Criterion 1—Publicly-available defect datasets. Prior work raises concerns about the replicability of software engineering studies [184]. In order to foster future replication of our work, we focus on publicly-available defect datasets.

Criterion 2—Datasets that are reliable and of high quality. Defect models rely greatly on the quality of the datasets that are used to construct them. [192] raise concerns related to data quality in the NASA datasets. Furthermore, [168] show that problematic data remain in the cleaned NASA datasets. Thus, the quality of the NASA datasets is questionable. To ensure that the studied datasets are reliable and of high quality, we exclude the NASA datasets from our study.

Criterion 3—Datasets that produce non-overly optimistic model performance. Classification techniques that are trained on imbalanced data often favour the majority class. When defective modules are the majority class, defect models are likely to produce overly optimistic performance estimates. Thus, we exclude datasets that have a defective ratio above 50%.

Criterion 4—Datasets from which we can accurately derive explanations. Analysts would only consider models that fit the data well (i.e., AUC > 0.7) and are stable (i.e., EPV > 10) [212]. Hence, we only focus on datasets that produce such accurate and stable models. To identify datasets that produce accurate models, we generate 100 sets of training and testing samples using the out-of-sample bootstrap validation technique. While we use training samples to construct defect models (i.e., logistic regression and random forests), we use testing samples to evaluate such models using the AUC measure. We then compute the average AUC estimation of all models and exclude datasets that produce models with the average AUC estimation of below 0.7. To identify datasets that produce stable models, we compute the EPV measure for each dataset. EPV (Event-Per-Variables) is a measure of the risk of overfitting. EPV is

Table 4.2 A statistical summary of the studied datasets.

Project	Dataset	Modules	Metrics	Defective Ratio	EPV	AUC_{LR}	AUC_{RF}
Apache	Xalan 2.6	885	20	46	21	0.79	0.85
Eclipse	Debug 3.4	1,065	17	25	15	0.72	0.81
	JDT	997	15	21	14	0.81	0.82
	Mylyn	1,862	15	13	16	0.78	0.74
	PDE	1,497	15	14	14	0.72	0.72
	Platform 2	6,729	32	14	30	0.82	0.84
	Platform 2.1	7,888	32	11	27	0.77	0.78
	Platform 3	10,593	32	15	49	0.79	0.81
	SWT 3.4	1,485	17	44	38	0.87	0.97
Proprietary	Prop 1	18,471	20	15	137	0.75	0.79
	Prop 2	23,014	20	11	122	0.71	0.82
	Prop 4	8,718	20	10	42	0.74	0.72
	Prop 5	8,516	20	15	65	0.7	0.71

calculated as a ratio of the number of occurrences of the least frequently occurring class of the dependent variable (i.e., the number of defective modules) to the number of software metrics that are used to train the model. We then exclude datasets that have the EPV values of below 10 since models that are constructed using such datasets have a high risk of overfitting [212].

To satisfy criterion 1, similar to prior work [210], we begin our study using a collection of the 101 publicly-available defect datasets that are collected from 5 different corpora, i.e., 76 datasets from the Tera-PROMISE Repository [139], 12 clean NASA datasets as provided by Shepperd *et al.* [192], 5 datasets as provided by Kim *et al.* [101], 5 datasets as provided by D'Ambros *et al.* [33, 34], and 3 datasets as provided by Zimmermann *et al.* [238]. To satisfy criterion 2, we exclude 12 datasets for which their data quality is questionable. To satisfy criterion 3, we exclude 17 datasets that have a defective ratio above 50%. Finally, to satisfy criterion 4, we exclude 59 datasets which have an EPV value below 10 and produce models with an AUC value below 0.7. Hence, we focus 13 defect datasets from 4 corpora, i.e., 5 datasets as provided by Jureczko *et al.* [94], 3 datasets as provided by D'Ambro *et al.* [33, 34], 3 datasets as provided by Zimmermann *et al.* [238], and 2 datasets as provided by Kim *et al.* [101].

Table 4.2 shows a statistical summary of the 13 studied datasets.

4.3.2 Studied Feature Selection Techniques

Feature selection is a data preprocessing technique for selecting a subset of the best software metrics prior to constructing a defect model. There is a plethora of feature selection techniques that can be applied [60], e.g., filter-based, wrapper-based, and embedded-based families. Since it is impractical to study all of these techniques, we would like to select a manageable set of feature selection techniques for our study. Similar to Ghotra *et al.* [53], we select two commonly-used families of feature selection techniques, i.e., filter-based feature selection techniques and wrapper-based feature selection techniques. Thus, embedded-based feature selection techniques are excluded from our analysis, as they are rarely explored in software engineering.

In this study, we select 12 commonly-used feature selection techniques from three families (i.e., 5 filter-based, 5 wrapper-based feature selection techniques, and one hybrid feature selection techniques), and our own contribution AutoSpearman [89] for evaluation. For filter-based feature selection techniques, we select correlation-based (CFS), information gain (IG), chi-squared-based (χ^2), consistency-based (CON), and findCorrelation. For wrapper-based feature selection techniques, we select Recursive Feature Elimination with two classification techniques (i.e., logistic regression (RFE-LR) and random forest (RFE-RF)), and three directions of Stepwise Regression (i.e., forward (Step-FWD), backward (Step-BWD), and both (Step-BOTH)). For a hybrid feature selection technique, we use a combination of chi-squared based filter-based and wrapper-based using logistic regression as provided by the `HybridFS` R package [162]. We select these feature selection techniques since they are the most commonly used ones in previous work on defect prediction[5, 34, 40, 97, 140, 160, 198]. Table 4.3 provides the summary of the detailed implementation for the twelve studied feature selection techniques. Below, we provide the description of each studied feature selection technique.

Filter-based feature selection techniques

Filter-based feature selection techniques search for the best subset of metrics according to an evaluation criterion regardless of model construction. Since constructing models is not required, the use of filter-based feature selection techniques is considered low cost and widely used in the defect prediction literature [5, 26, 40, 97, 140, 160]. There are many variants of filter-based feature selection techniques, which we describe below.

Correlation-based feature selection [63] is a deterministic feature selection technique that searches for the best subset of metrics that shares the strongest relationship with the outcome, while having a low correlation among themselves.

Information gain feature selection [148] is a deterministic feature selection technique that ranks metrics according to the information gain with respect to the outcome. The information gain is measured by how much information of the outcome is provided by a metric.

Chi-Squared-based feature selection [129] is a deterministic feature selection technique that assesses the importance of metrics with the χ^2 statistic which is a non-parametric statistical test of independence.

Consistency-based feature selection [35] is a deterministic feature selection technique that uses the consistency measure (i.e., inconsistency rate) to evaluate a subset of metrics.

Table 4.3 A summary of the detailed implementation for the twelve studied feature selection techniques.

Type	Technique	R Package	R Function	Abbreviation
Filter-based Feature Selection Techniques	Correlation-based	FSelector [188]	`cfs(class~metrics, dataset)`	CFS
	Information Gain		`information.gain(class~metrics, dataset)`	IG
	Chi-Squared-based (χ^2)		`chi.squared(class~metrics, dataset)`	Chisq
	Consistency-based		`consistency(class~metrics, dataset)`	CON
	findCorrelation	Hmisc [67] caret [111]	`correlation.matrix = rcorr(as.matrix(dataset[, metrics]), type = ‘spearman’)$r`	findCorrelation
			`correlated.metrics = findCorrelation(correlation.matrix, cutoff = 0.7, exact = TRUE)`	
			`metrics[-correlated.metrics]`	
Wrapper-based Feature Selection Techniques	Recursive Feature Elimination (Logistic Regression)	caret [111]	`lrFuncs.AUC = lrFuncs`	RFE-LR
			`lrFuncs.AUC$summary = twoClassSummary`	
			`control = rfeControl(functions = lrFuncs.AUC, method = "boot", number = iterations)`	
			`rfe(x = dataset[, metrics], y = dataset[, class], rfeControl = control, metric = "ROC")`	
	Recursive Feature Elimination (Random Forest)		`rfFuncs.AUC = rfFuncs`	RFE-RF
			`rfFuncs.AUC$summary = twoClassSummary`	
			`control = rfeControl(functions = rfFuncs.AUC, method = "boot", number = iterations)`	
			`rfe(x = dataset[, metrics], y = dataset[, class], rfeControl = control, metric = "ROC")`	
	Stepwise Regression (Forward Direction)	stats [214]	`null.model = glm(class~1, data = dataset, family = binomial())`	Step-FWD
			`full.model = glm(class~metrics, data = dataset, family = binomial())`	
			`step(null.model, scope = list(upper = full.model), data = dataset, direction = "fwd")`	
	Stepwise Regression (Backward Direction)		`full.model = glm(class~metrics, data = dataset, family = binomial())`	Step-BWD
			`step(full.model, data = dataset, direction = "bwd")`	
	Stepwise Regression (Both Directions)		`null.model = glm(class~1, data = dataset, family = binomial())`	Step-BOTH
			`full.model = glm(class~metrics, data = dataset, family = binomial())`	
			`step(null.model, scope = list(upper = full.model), data = dataset, direction = "both")`	
Hybrid (Filter and Wrapper)	HybridFS	HybridFS [162]	`HybridFS(dataset, class)`	Hybrid
	AutoSpearman	Rnalytica [1]	`AutoSpearman(dataset, metrics, spearman.threshold = 0.7, vif.threshold = 5)`	AutoSpearman

The technique finds the optimal subset of metrics whose inconsistency rate approximates the inconsistency rate of all metrics.

findCorrelation is a deterministic feature selection technique that reduces pair-wise correlations among metrics using a correlation matrix. The technique can be applied to any correlation matrix (e.g., Pearson and Spearman). In our paper, we choose the Spearman rank correlation test instead of other correlation tests (e.g., Pearson) since the test is resilient to non-normal distributions as commonly present in defect datasets.

Table 4.3 summarises the detailed implementation of the studied filter-based feature selection techniques.

Wrapper-based Feature Selection Techniques

Wrapper-based feature selection techniques [93, 105] use classification techniques to assess each subset of metrics and find the best subset of metrics according to an evaluation criterion. Wrapper-based feature selection is made up of three steps, which we described below.

(Step 1) Generate a subset of metrics. Since it is impossible to evaluate all possible subsets of metrics, wrapper-based feature selection often uses search techniques (e.g., best first, greedy hill climbing) to generate candidate subsets of metrics for evaluation.

(Step 2) Construct a classifier using a subset of metrics with a predetermined classification technique. Wrapper-based feature selection constructs a classification model using a candidate subset of metrics for a given classification technique (e.g., logistic regression and random forest).

(Step 3) Evaluate the classifier according to a given evaluation criterion. Once the classifier is constructed, wrapper-based feature selection evaluates the classifier using a given evaluation criterion (e.g., Akaike Information Criterion).

For each candidate subset of metrics, wrapper-based feature selection repeats Steps 2 and 3 in order to find the best subset of metrics according to the evaluation criterion. Finally, it provides the best subset of metrics that yields the highest performance according to the evaluation criterion.

In this study, we select two commonly-used variants of wrapper-based feature selection techniques, which we describe below.

Recursive Feature Elimination (RFE) [60] searches for the best subset of metrics by recursively eliminating the least important metric. First, RFE constructs a model using all metrics and ranks metrics according to their importance score (e.g., Breiman's Variable Importance for random forest). In each iteration, RFE excludes the least important metric and reconstruct a model. Finally, RFE provides the subset of metrics which yields the best performance according to an evaluation criterion (e.g., AUC). In our study, we select the AUC measure since it measures the discriminatory power of models, as suggested by recent research [52, 114, 174, 208]. We use the implementation of the recursive feature elimination using the `rfe` function as provided by the `caret` R package [111].

Stepwise Regression [28] finds the best subset of metrics by individually assessing each metric and adding (or removing) a metric if it improves an evaluation criterion (e.g., Akaike

Algorithm 1: AutoSpearman

```
Input  : $M$ is a set of studied metrics,
         $sp.t$ is a threshold value for a Spearman rank
         correlation test,
         $vif.t$ is a threshold value for a Variance
         Inflation Factor analysis.
Output : $M'$ is a set of non-correlated metrics based on a Spearman rank correlation
         test and a Variance Inflation Factor analysis.
1  $M' = M$
2  $S = Spearman(M, M)$
3  $C_S = \{c(m_i, m_j) \in S | abs(c(m_i, m_j)) \geq sp.t\}$
4  $C_S = sort(C_S)$
5  for $c(m_i, m_j)$ in $C_S$ do
6      $selected.metric = min($
         $mean(abs(Spearman(m_i, M - \{m_i, m_j\}))),$
         $mean(abs(Spearman(m_j, M - \{m_i, m_j\}))))$
7      $removed.metric = \{m_i, m_j\} - selected.metric$
8      $C_S = \{c(m_i, m_j) \in C_S |$
         $m_i \neq removed.metric \land m_j \neq removed.metric\}$
9      $M' = M' - removed.metric$
10 end
11 repeat
12     $V = VIF(M')$
13     $C_V = \{v(m_i) \in V | v(m_i) \geq vif.t\}$
14     $removed.metric = \{m_i |$
         $v(m_i) \in C_V \land v(m_i) = max(C_V)$
15     $M' = M' - removed.metric$
16 until $|C_V| = 0$;
17 return $M'$
```

Information Criterion). The process is repeated until there is no improvement from adding or removing a metric. In this study, we investigate three directions (i.e., forward, backward, and both directions) of Stepwise Regression. We use the implementation of Stepwise Regression using the `step` function as provided by the `stats` R package [214].

AutoSpearman

In our recent work [89], we introduce AutoSpearman, an automated metric selection approach based on the Spearman rank correlation test and the VIF (Variance Inflation Factor) analysis for statistical inference. Below, we describe AutoSpearman using Algorithm 1, where S is a set of Spearman coefficients for each pair of metrics, C_S is a set of Spearman coefficients that are above a Spearman threshold value ($sp.t$), V is a set of VIF scores of metrics, C_V is a set of VIF scores of metrics that are above a VIF threshold value ($vif.t$), and M' is a set of non-correlated metrics based on the Spearman rank correlation test and the Variance Inflation Factor analysis. The high-level concept of AutoSpearman can be summarised into 2 parts:

(Part 1) Automatically select non-correlated metrics based on a Spearman rank correlation test. We first measure the correlation of all metrics using the Spearman rank correlation test (ρ) (*cf.* Line 2). We use the interpretation of correlation coefficients ($|\rho|$) as provided by Kraemer *et al.* [107]—i.e., a Spearman correlation coefficient of above or equal to 0.7 is considered a strong correlation. Thus, we only consider the pairs that have an absolute Spearman correlation coefficient of above or equal to the threshold value ($sp.t$) of 0.7 (*cf.* Line 3).

To automatically select non-correlated metrics based on the Spearman rank correlation test, we start from the pair that has the highest Spearman correlation coefficient (*cf.* Line 4). Since the two correlated metrics under examination can be linearly predicted with each other, one of these two metrics must be removed. Thus, we select the metric that has the lowest average values of the absolute Spearman correlation coefficients of the other metrics that are not included in the pair (*cf.* Line 6). That means the removed metric is another metric in the pair that is not selected (*cf.* Line 7). Since the removed metric may be correlated with the other metrics, we remove any pairs of metrics that are correlated with the removed metric (*cf.* Line 8). Finally, we exclude the removed metric from the set of the remaining metrics (M') (*cf.* Line 9). We repeat this process until all pairs of metrics have their Spearman correlation coefficient below a threshold value of 0.7 (*cf.* Line 5).

(Part 2) Automatically select non-correlated metrics based on a Variance Inflation Factor analysis. We first measure the magnitude of multicollinearity of the remaining metrics (M') from `Part 1` using the Variance Inflation Factor analysis (*cf.* Line 12). We use a VIF threshold value ($vif.t$) of 5 to identify the presence of multicollinearity, as suggested by Fox *et al.* [43] and prior work [11, 85, 86, 131] (*cf.* Line 13).

To automatically remove correlated metrics from the Variance Inflation Factor analysis, we identify the removed metric as the metric that has the highest VIF score (*cf.* Line 14). We then exclude the removed metric from the set of the remaining metrics (M') (*cf.* Line 15). We apply the VIF analysis on the remaining metrics until none of the remaining metrics have their VIF scores above or equal to the threshold value (*cf.* Line 16). Finally, AutoSpearman produces a subset of non-correlated metrics based on the Spearman rank correlation test and the VIF analysis (M') (*cf.* Line 17).

Similar to filter-based feature selection techniques, *Part 1* of AutoSpearman measures the correlation of all metrics using the Spearman rank correlation test regardless of model construction. Similar to wrapper-based feature selection techniques, *Part 2* of AutoSpearman constructs linear regression models to measure the magnitude of multicollinearity of metrics. Thus, we consider AutoSpearman as a hybrid feature selection technique (both filter-based and wrapper-based).

4.3.3 Studied Classification Techniques

Logistic regression is a statistical learner which explains the relationship between one binary dependent variable (e.g., defect-proneness) and one or more independent variables (e.g., software metrics).
Random forests is a machine learner that constructs multiple decision trees from bootstrap samples [21]. The final predicted class of a software module is the aggregation of the votes from all of the constructed trees.

4.4 Experimental Results

(RQ1) Do feature selection techniques consistently produce the same subset of metrics when applied on the same training sample?

Approach. To address RQ1, we investigate the consistency of the subsets of metrics that are produced by feature selection techniques. We first generate training samples. Then, we apply feature selection techniques on each training sample. Finally, we analyse the consistency of subsets of metrics that are produced by the studied feature selection techniques. We describe each step below.

(Step 1) Generate training samples. To generate training samples, we use the out-of-sample bootstrap validation technique that (1) leverages aspects of statistical inference [39, 47, 68]; and (2) produces the least bias and variance of performance estimates for defect prediction [212]. We randomly generate a bootstrap sample of size N with replacement from an original dataset, where N is the size of the original dataset. On average, 36.8% of the original dataset will not be selected, since a bootstrap sample is selected with replacement [39]. We repeat the out-of-sample bootstrap process 100 times.

(Step 2) Apply feature selection techniques. We only apply feature selection techniques on *a training sample*, instead of the whole dataset in order to avoid producing optimistically biased performance estimates and explanation [117]. The subsets of metrics that are produced by each studied feature selection technique for all studied datasets are available in the online appendix [90].

(Step 3) Analyse the consistency of subsets of metrics among different feature selection techniques. Ideally, feature selection techniques should consistently produce the same subset of metrics. We measure the consistency as a percentage of the unique metrics that consistently appeared among all of the studied feature selection techniques compared to all of the unique metrics for all studied feature selection techniques (i.e., $\frac{|S_{\mathrm{FS}_1\mathrm{TS}_j} \cap S_{\mathrm{FS}_2\mathrm{TS}_j} ... \cap S_{\mathrm{FS}_9\mathrm{TS}_j}|}{|S_{\mathrm{FS}_1\mathrm{TS}_j} \cup S_{\mathrm{FS}_2\mathrm{TS}_j} ... \cup S_{\mathrm{FS}_9\mathrm{TS}_j}|}$, where $S_{\mathrm{FS}_i\mathrm{TS}_j}$ is a subset of metrics that is produced by a feature selection technique (FS_i) when applied on a training sample (TS_j)). We present the consistency percentage using boxplots in Figure 4.1.
Results. **When applying the studied feature selection techniques to the same training sample, only 0–6% of the metrics are consistently selected.** Figure 4.1 shows the per-

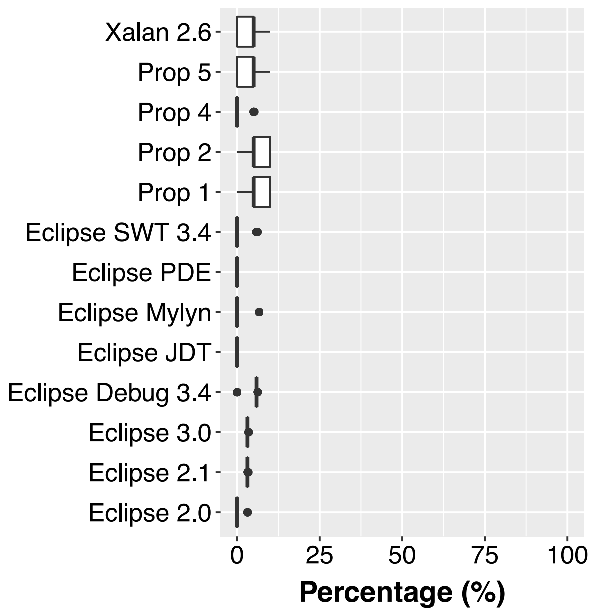


Figure 4.1 The percentage of metrics that are consistently selected when applying feature selection techniques to the same training sample for all defect datasets.

centage of metrics that are consistently selected when applying feature selection techniques on each training sample of each defect dataset. We observe that, at the median, only 0-6% of the metrics are consistently selected among the studied feature selection techniques. In other words, as many as 94-100% of the metrics are inconsistently selected among the studied feature selection techniques, suggesting that feature selection techniques select different metrics even if they are applied on the same training sample. We provide an illustrative example below.

Illustrative Example. We select the Eclipse Platform 2 dataset as the subject of this example, since it is widely used in a large number of defect prediction studies [16, 151, 237]. We first draw a bootstrap training sample (*cf.* Step 1 of RQ1) and apply all studied feature selection techniques (*cf.* Step 2 of RQ1). Unfortunately, we observe that none of the metrics is consistently selected across all studied feature selection techniques. The most commonly-selected metric is `pre`, which is selected by the 10 studied feature selection techniques except for information gain and chi-squared-based. This observation raises concerns related to the construct validity of prior work that their conclusions could be altered when another feature selection technique is applied. We report the produced subsets of metrics for this illustrative example in Table 4.4.

Feature selection techniques produce inconsistent subsets of metrics. As many as 94-100% of the metrics are inconsistently selected among the studied feature selection techniques, suggesting that the conclusions of prior work could be altered when another feature selection technique is applied.

Table 4.4 The subsets of metrics that are produced by all of the twelve studied feature selection techniques on a training sample from the Eclipse Platform 2 dataset. While a ✓ mark indicates that a metric is selected by a feature selection technique, a ✗ mark indicates that a metric is not selected by a feature selection technique.

	CFS	IG	Chisq	CON	findCorrelation	RFE-LR	RFE-RF	Step-FWD	Step-BWD	Step-BOTH	Hybrid	AutoSpearman
ACD	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✗	✓
CC_avg	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
CC_max	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗	✗
CC_sum	✗	✓	✓	✓	✗	✓	✓	✗	✗	✗	✓	✗
FOUT_avg	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗	✗
FOUT_max	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
FOUT_sum	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗
MLOC_avg	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
MLOC_max	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗	✗
MLOC_sum	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗
NBD_avg	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓
NBD_max	✓	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗	✗
NBD_sum	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗	✓	✗
NOF_avg	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗	✗	✓
NOF_max	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗
NOF_sum	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
NOI	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
NOM_avg	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗	✓
NOM_max	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗
NOM_sum	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
NOT	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✓
NSF_avg	✗	✗	✗	✓	✓	✗	✓	✓	✗	✓	✗	✓
NSF_max	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗
NSF_sum	✗	✗	✗	✗	✗	✓	✓	✗	✓	✗	✗	✗
NSM_avg	✓	✗	✗	✓	✓	✗	✓	✓	✗	✓	✗	✗
NSM_max	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗
NSM_sum	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
PAR_avg	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗	✓
PAR_max	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✗
PAR_sum	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
pre	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
TLOC	✗	✓	✓	✓	✗	✗	✓	✗	✗	✗	✓	✗

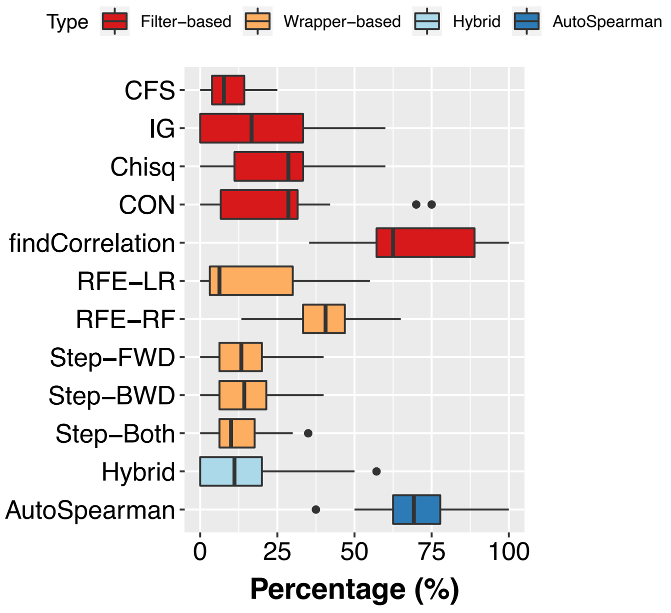


Figure 4.2 The percentage of metrics that are consistently selected when applying feature selection techniques to different training samples from the same dataset.

(RQ2) Do feature selection techniques consistently produce the same subset of metrics when applied across different training samples?

Approach. To address RQ2, we investigate the consistency of subsets of metrics across different training samples. Similar to RQ1, we start from the subsets of metrics that are produced by all studied feature selection techniques for each training sample of each studied dataset from RQ1 (*cf.* Step 2 of RQ1). Ideally, each feature selection technique should produce the same subset of metrics for all of the 100 training samples. We compute the consistency as a percentage of the unique metrics that consistently appeared among all of the 100 training samples compared to all of the unique metrics for all training samples (i.e., $\frac{|S_{\mathrm{FS}_i\mathrm{TS}_1} \cap S_{\mathrm{FS}_i\mathrm{TS}_2} ... \cap S_{\mathrm{FS}_i\mathrm{TS}_{100}}|}{|S_{\mathrm{FS}_i\mathrm{TS}_1} \cup S_{\mathrm{FS}_i\mathrm{TS}_2} ... \cup S_{\mathrm{FS}_i\mathrm{TS}_{100}}|}$, where $S_{\mathrm{FS}_i\mathrm{TS}_j}$ is a subset of metrics that is produced by a feature selection technique (FS_i) when applied on a training sample (TS_j)). Finally, we present the consistency percentage using boxplots in Figure 4.2.

Results. **Surprisingly, when applying the studied feature selection techniques to different training samples from the same dataset, 6-69% of the metrics are consistently selected.** Figure 4.2 shows the percentage of metrics that are consistently selected when applying feature selection techniques to different training samples. We find that, at the median, 8%, 17%, 29%, 29%, 63%, 6%, 41%, 13%, 14%, 10%, 11%, and 69% of the metrics are consistently selected when applying CFS, IG, Chisq, CON, findCorrelation, RFE-LR, RFE-RF, Step-FWD, Step-BWD, Step-BOTH, Hybrid, and AutoSpearman to different training samples, respectively. In other words, the selected metrics are 31-94% inconsistent when applying the studied feature selection techniques to different training samples from the same dataset. We observe that AutoSpearman yields the highest consistency and leads

Table 4.5 The subsets of metrics that are produced by the correlation-based feature selection technique (CFS) when applied on different training samples. While the green texts represent metrics that are consistently selected, the red texts represent inconsistently selected metrics.

Training Sample	The Subset of Metrics that is produced by CFS
Sample 1 (TS_1)	pre, MLOC_sum, NBD_sum, PAR_max, CC_max, NBD_max, NOM_max, NSM_avg
Sample 2 (TS_2)	pre, MLOC_sum, NBD_sum, PAR_max, CC_max, FOUT_sum, NBD_avg, NSM_max, PAR_sum

to absolute improvements of consistency of up to 86% compared to other studied feature selection techniques.

Although the training samples are drawn from the same original dataset, none of the commonly-used feature selection techniques consistently produce the same subset of metrics. We suspect that the inconsistency of subsets of metrics among training samples has to do with the differences in the characteristics of data among such training samples. For example, different training samples may have different correlation among metrics. Thus, the correlation-based feature selection technique may produce different subsets of metrics that share the strongest relationship with the outcome while having a low correlation among themselves for such different training samples.

The inconsistency of subsets of metrics among training samples suggests that the randomisation of training samples could produce defect models that are constructed from different subsets of metrics even if the training samples are drawn from the same dataset. Below, we provide a detailed illustrative example to demonstrate such inconsistency of subsets of metrics in practice.

Illustrative Example. Similar to prior illustrative examples in RQ1, we use the Eclipse Platform 2 dataset and the correlation-based feature selection technique (CFS) for this illustrative example. First, we draw two bootstrap training samples (*cf.* Step 1 of RQ1) and apply CFS on these samples. We observe that only 5 of 12 metrics are consistently selected. This inconsistency in subsets of metrics restricts an application of post-hoc multiple comparison analyses (e.g., a Scott-Knott test) to identify the most important metrics when explaining defect models, since the importance score of the metric may not be available for multiple comparisons in another training sample if the metric is not selected by a feature selection technique. We provide the subsets of metrics that are produced by CFS for this illustrative example in Table 4.5.

Surprisingly, when applying commonly-used feature selection techniques to different training samples from the same dataset, we find that 31-94% of the selected metrics are inconsistent, suggesting that post-hoc multiple comparison tests (e.g., a Scott-Knott test) should not be applied to identify the most important metrics when explaining defect models. On the other hand, we observe that AutoSpearman yields the highest consistency and leads to absolute improvements of consistency up to 86% compared to other studied feature selection techniques.

(RQ3) Do feature selection techniques consistently produce the same subset of metrics when they are applied repeatedly?

Approach. To address RQ3, we investigate the consistency of subsets of metrics when feature selection techniques are applied repeatedly. Unlike Step 1 of RQ1 and RQ2, for each studied defect dataset, we use only one bootstrap training sample. To foster future replication studies, we set random seeds prior to applying feature selection techniques. Thus, we apply feature selection techniques on one training sample for all studied defect datasets with different random seeds. Finally, we analyse the consistency of subsets of metrics that are produced by feature selection techniques. We describe each step below.

(Step 1) Apply feature selection techniques on the training sample with different random seeds. Since the goal of this research question is to measure the consistency across different random seeds, we use 100 random seeds that range from 1 to 100 prior to applying feature selection techniques. We then apply feature selection techniques on the training sample and produce 100 subsets of metrics for each technique and each defect dataset.

(Step 2) Analyse the consistency of subsets of metrics across different random seeds. We start from the 100 subsets of metrics that are produced in Step 1 for each technique and each defect dataset. Ideally, each technique should produce the same subset of metrics for all 100 random seeds. We compute the consistency as a percentage of the unique metrics that consistently appeared among all of the 100 random seeds compared to all of the unique metrics for all random seeds (i.e., $\frac{|S_{\mathrm{FS}_i\mathrm{RS}_1\mathrm{TS}} \cap S_{\mathrm{FS}_i\mathrm{RS}_2\mathrm{TS}} \dots \cap S_{\mathrm{FS}_i\mathrm{RS}_{100}\mathrm{TS}}|}{|S_{\mathrm{FS}_i\mathrm{RS}_1\mathrm{TS}} \cup S_{\mathrm{FS}_i\mathrm{RS}_2\mathrm{TS}} \dots \cup S_{\mathrm{FS}_i\mathrm{RS}_{100}\mathrm{TS}}|}$, where $S_{\mathrm{FS}_i\mathrm{RS}_j\mathrm{TS}}$ is a subset of metrics that is produced by a feature selection technique (FS_i) when applied with a random seed (RS_j) on a training sample (TS)). We present the consistency percentage using boxplots in Figure 4.3.

Results. **The Recursive Feature Elimination technique is the only studied feature selection technique that produces inconsistent subsets of metrics when applied repeatedly with different random seeds regardless of the classification techniques.** Figure 4.3 shows the percentage of metrics that are consistently selected when repeatedly applying feature selection techniques with different random seeds. The results show that the Recursive Feature Elimination technique is the only studied feature selection technique that produces inconsistent subsets of metrics when applied with different random seeds. We observe that, at the median, 38% and 87% of the metrics are consistently selected when

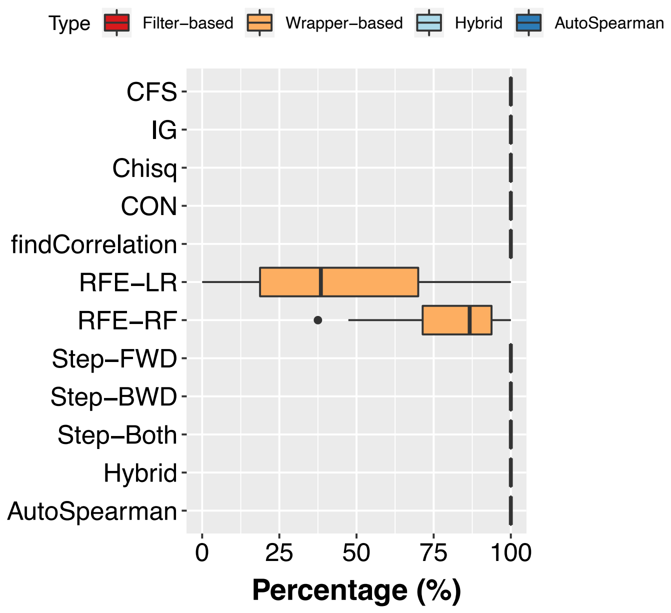


Figure 4.3 The percentage of metrics that are consistently selected when applying feature selection techniques with different random seeds.

repeatedly applying Recursive Feature Elimination with logistic regression and random forest (i.e., RFE-LR and RFE-RF, respectively) with different random seeds. In other words, the selected metrics are 62% and 13% inconsistent for RFE-LR and RFE-RF, respectively. Similar to RQ2, we observe a higher consistency of subsets of metrics when applying Recursive Feature Elimination with random forest compared to logistic regression. We suspect that such a higher consistency of subsets of metrics has to do with the process of constructing and using multiple trees for a random forests model. For examples, unlike logistic regression that constructs only one model, random forests constructs multiple trees and uses the aggregated results of these trees to generate predictions.

The inconsistency of subsets of metrics when applied repeatedly with different random seeds suggests that although these variants of Recursive Feature Elimination are applied on the same sample, simply altering a random seed can lead them to produce different subset of metrics. We suspect that the inconsistency in subsets of metrics has to do with the randomisation in the process of sampling data for validation in Recursive Feature Elimination. We discuss and provide a detailed illustrative example below.

Illustrative Example. Similar to prior illustrative examples in RQ1 and RQ2, we use the Eclipse Platform 2 dataset as the subject of this example. First, we draw a bootstrap training sample (unlike Step 1 of RQ1, we use only one training sample) and apply RFE-LR with two different random seeds, i.e., `set.seed(1)` and `set.seed(2)`. We find that RFE-LR selects 16 metrics when applied with one random seed, while selecting 20 metrics when applied with another random seed. Across these subsets of metrics, 16 metrics are consistently selected. Such inconsistency in subsets of metrics when repeatedly applying Recursive Feature Elimination technique raises concerns related to the reliability and the construct validity.

Table 4.6 The subsets of metrics that are produced by the Recursive Feature Elimination technique with logistic regression (RFE-LR) when applied with different random seeds on the same training sample. While the green texts represent metrics that are consistently selected, the red texts represent inconsistently selected metrics.

Random Seed	The Subset of Metrics that is produced by RFE-LR
RS_1 (`set.seed(1)`)	ACD, CC_max, CC_sum, FOUT_avg, FOUT_sum, MLOC_max, NBD_avg, NBD_max, NBD_sum, NOF_max, NOM_avg, NSF_sum, PAR_avg, PAR_max, PAR_sum, pre
RS_2 (`set.seed(2)`)	ACD, CC_max, CC_sum, FOUT_avg, FOUT_sum, MLOC_max, NBD_avg, NBD_max, NBD_sum, NOF_max, NOM_avg, NSF_sum, PAR_avg, PAR_max, PAR_sum, pre, NOF_avg, NOF_sum, NSM_avg, NSF_max

We provide the subsets of metrics that are produced by RFE-LR when applied with two different random seeds for this illustrative example in Table 4.6.

The Recursive Feature Elimination technique is the only technique that produces inconsistent subsets of metrics when repeated with different random seeds. Such inconsistency amounts to 62% and 13% of the subsets of metrics for logistic regression and random forest, respectively. This finding indicates that even if Recursive Feature Elimination is applied on the same training sample, simply altering a random seed can lead the technique to produce different subset of metrics.

(RQ4) Do feature selection techniques consistently produce the same subset of metrics when reordering the model specification of a defect model?

Approach. To address RQ4, we investigate the consistency of subsets of metrics across different model specifications. Similar to RQ3, for each studied defect dataset, we use only one bootstrap training sample. Then, we apply feature selection techniques on the training sample with different model specifications for all studied defect datasets. Finally, we analyse the consistency of subsets of metrics that are produced by feature selection techniques. We describe each step below.

(Step 1) Apply feature selection techniques on the training sample with different model specifications. Since it is difficult to measure the consistency across all possible model specifications (e.g., 10 metrics would lead to 10! different model specifications), in this study, we generate 100 model specifications by randomly rearranging the ordering of metrics of each studied defect dataset differently for 100 times. For example, $y \sim x_1 + x_2 + ...$, $y \sim x_2 + x_1 + ...$, and

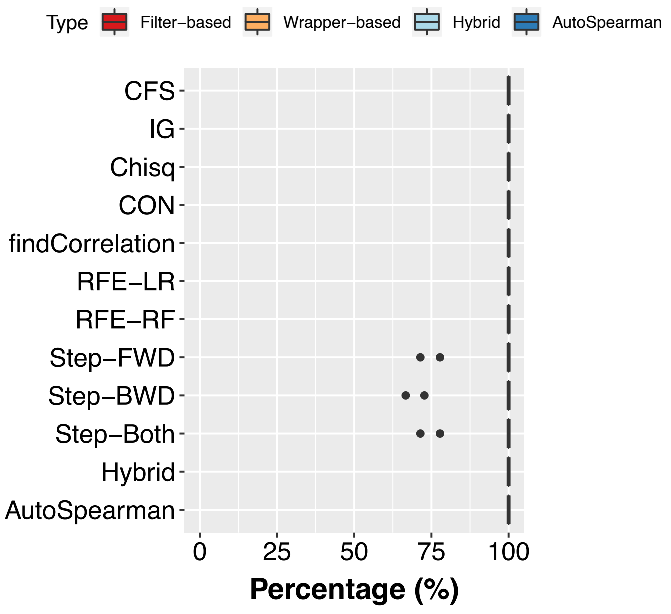


Figure 4.4 The percentage of metrics that are consistently selected when applying feature selection techniques with different model specifications.

$y \sim x_3 + x_1 + ...$ We then apply feature selection techniques on the training sample with 100 model specifications and produce 100 subsets of metrics for each technique and each defect dataset.

(Step 2) Analyse the consistency of subsets of metrics across different model specifications. We start from the subsets of metrics in Step 1 that are produced with 100 variations of model specification on the training sample for each feature selection technique of each studied dataset. Ideally, each feature selection technique should produce the same subset of metrics for all 100 variations of model specification regardless of the ordering of metrics, e.g., $\mathrm{FS}(y \sim x_1 + x_2) = \mathrm{FS}(y \sim x_2 + x_1)$. We compute the consistency as a percentage of the unique metrics that consistently appeared among all of the 100 variations of the model specification compared to all of the unique metrics for all model specifications (i.e., $\frac{|S_{\mathrm{FS}_i\mathrm{MS}_1\mathrm{TS}} \cap S_{\mathrm{FS}_i\mathrm{MS}_2\mathrm{TS}} ... \cap S_{\mathrm{FS}_i\mathrm{MS}_{100}\mathrm{TS}}|}{|S_{\mathrm{FS}_i\mathrm{MS}_1\mathrm{TS}} \cup S_{\mathrm{FS}_i\mathrm{MS}_2\mathrm{TS}} ... \cup S_{\mathrm{FS}_i\mathrm{MS}_{100}\mathrm{TS}}|}$, where $S_{\mathrm{FS}_i\mathrm{MS}_j\mathrm{TS}}$ is a subset of metrics that is produced by a feature selection technique (FS_i) when applied with a model specification (MS_j) on a training sample (TS)). We present the consistency percentage using boxplots in Figure 4.4.

Results. The studied feature selection techniques (except for Stepwise Regression) produce consistent subsets of metrics regardless of the ordering of metrics in model specifications. Figure 4.4 shows the percentage of metrics that are consistently selected when applying feature selection techniques with different model specifications. We find that all of the studied feature selection techniques (except for Stepwise Regression) produce consistent subsets of metrics regardless of the ordering of metrics in model specifications. We observe an inconsistency of subsets of metrics that are produced by Stepwise Regression when reordering the model specification in 2 studied defect datasets, i.e., Eclipse PDE and Eclipse SWT 3.4. We discuss and provide a detailed illustrative example below.

Table 4.7 The subsets of metrics that are produced by the forward direction Stepwise Regression (Step-FWD) when applied with different model specifications on the same training sample. While the green texts represent metrics that are consistently selected, the red texts represent inconsistently selected metrics.

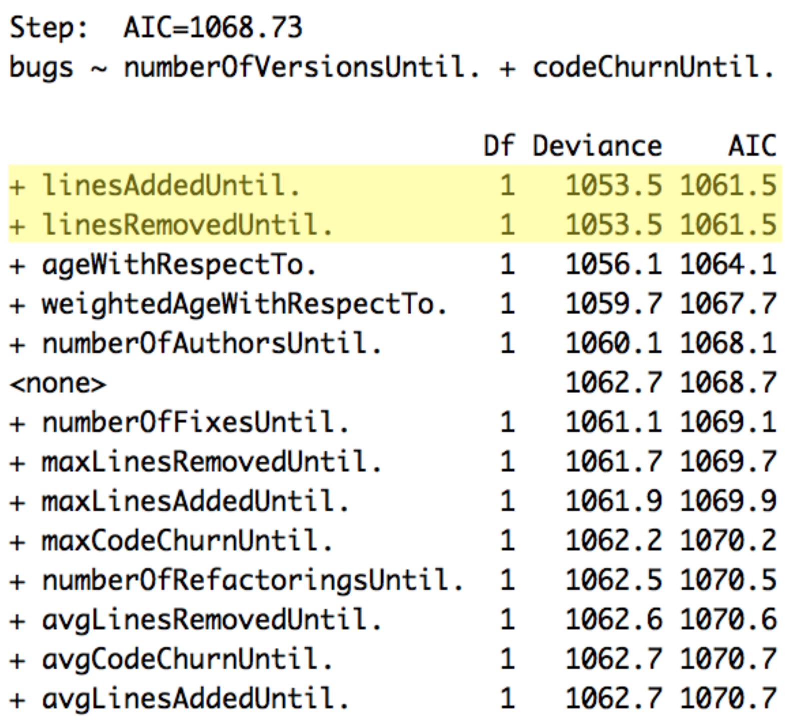
Model Specification	The Subset of Metrics that is produced by Step-FWD
Specification 1 (MS_1)	`ageWithRespectTo.`, `codeChurnUntil.`, `linesAddedUntil.`, `maxLinesRemovedUntil.`, `numberOfRefactoringsUntil.`, `numberOfVersionsUntil.`
Specification 2 (MS_2)	`ageWithRespectTo.`, `codeChurnUntil.`, `linesRemovedUntil.`, `maxLinesRemovedUntil.`, `numberOfRefactoringsUntil.`, `numberOfVersionsUntil.`

Illustrative Example. We select the Eclipse PDE dataset, one of the problematic defect datasets, as the subject of this example. Similar to the prior illustrative example in RQ3, we draw only one bootstrap training sample and apply the forward direction Stepwise Regression with two different model specifications. Table 4.7 shows the subsets of metrics that are produced by the forward direction Stepwise Regression when applied with two different model specifications. We observe that, while 5 metrics are consistently selected, Stepwise Regression either selects `linesAddedUntil` or `linesRemovedUntil` depending on which metric appears first in the ordering of metrics in a model specification. The result of Stepwise Regression shows that including either of these metrics equally improves the AIC value of a regression model. This finding leads us to conclude that, when including (or excluding) either of the two metrics can equally improve the AIC value of a regression model, Stepwise Regression includes (or excludes) the first metric that appears in the model specification. We further investigate the correlation of these problematic metrics and find that their Spearman correlation coefficient is as high as 0.87, suggesting that they are highly-correlated metrics. This finding raises concerns that correlated metrics may introduce the inconsistency of subsets of metrics when applying Stepwise Regression with different model specifications. We provide the subset of metrics for this illustrative example in Table 4.7 and the mentioned part of the results of Stepwise Regression in Figure 4.5.

Regardless of the search directions, Stepwise Regression is the only studied feature selection technique that produces inconsistent subsets of metrics when applied across model specifications.

(RQ5) Do feature selection techniques mitigate correlated metrics?

Approach. To identify correlated metrics, we apply correlation analyses on subsets of metrics that are produced by feature selection techniques. Similar to RQ1 and RQ2, we start from the subsets of metrics that are produced by all studied feature selection techniques for



```
Step:  AIC=1068.73
bugs ~ numberOfVersionsUntil. + codeChurnUntil.

                             Df Deviance    AIC
+ linesAddedUntil.            1   1053.5 1061.5
+ linesRemovedUntil.          1   1053.5 1061.5
+ ageWithRespectTo.           1   1056.1 1064.1
+ weightedAgeWithRespectTo.   1   1059.7 1067.7
+ numberOfAuthorsUntil.       1   1060.1 1068.1
<none>                            1062.7 1068.7
+ numberOfFixesUntil.         1   1061.1 1069.1
+ maxLinesRemovedUntil.       1   1061.7 1069.7
+ maxLinesAddedUntil.         1   1061.9 1069.9
+ maxCodeChurnUntil.          1   1062.2 1070.2
+ numberOfRefactoringsUntil.  1   1062.5 1070.5
+ avgLinesRemovedUntil.       1   1062.6 1070.6
+ avgCodeChurnUntil.          1   1062.7 1070.7
+ avgLinesAddedUntil.         1   1062.7 1070.7
```

Figure 4.5 The results of Stepwise Regression.

each training sample of each studied dataset from RQ1 (*cf.* Step 2 of RQ1). Then, we analyse the correlation among metrics for each subset of metrics that are produced by the studied feature selection techniques. In this study, we focus on two types of correlation among metrics, i.e., collinearity and multicollinearity. Collinearity is a phenomenon in which one metric can be linearly predicted by another metric. On the other hand, multicollinearity is a phenomenon in which one metric can be linearly predicted by a combination of two or more metrics. We describe each step below.

(Step 1) Analyse collinearity. To analyse collinearity, we use a Spearman rank correlation test (ρ) to measure the correlation between metrics. We choose the Spearman test instead of other correlation tests (e.g., Pearson) since the Spearman test is resilient to non-normal distributions as commonly present in defect datasets. We use the interpretation of correlation coefficients ($|\rho|$) as provided by [107], i.e., a Spearman correlation coefficient of above *0.7 is considered as a strong correlation*. Thus, two metrics which have their Spearman correlation coefficient of above 0.7 are considered correlated. We use the implementation of the `rcorr` function as provided by the `Hmisc` R package [67].

(Step 2) Analyse multicollinearity. To analyse multicollinearity, we use the Variance Inflation Factor analysis (VIF) [44]. VIF determines how well a metric can be linearly predicted by a combination of other metrics through a construction of a regression model. A VIF score of a metric under examination is an R^2 goodness-of-fit of the model that is constructed by the other metrics to predict the metric under examination where a VIF score is $\frac{1}{1-R^2}$. We use a VIF threshold value of 5 to identify the presence of multicollinearity, as suggested by [43] and prior work [11, 85, 86, 131]. Thus, metrics that have their VIF score

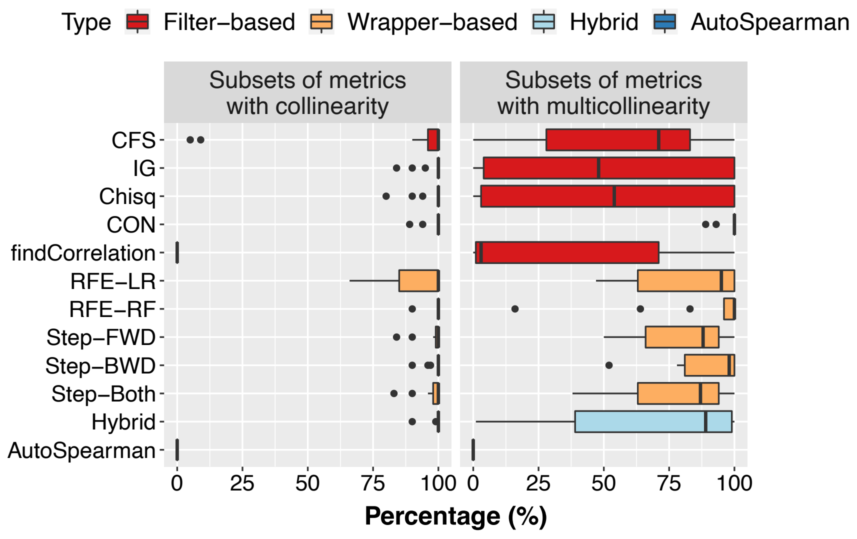


Figure 4.6 The percentage of subsets of metrics that contain correlated metrics for each studied feature selection technique. The left boxplots present the percentage of subsets of metrics with collinearity, while the right boxplots present the percentage of subsets of metrics with multicollinearity.

above 5 are considered correlated. We use the implementation of the Variance Inflation Factor analysis using the `vif` function as provided by the `rms` R package [69]. Finally, we present the results using boxplots in Figure 4.6.

Results. All of the commonly-used feature selection techniques do not mitigate correlated metrics except for AutoSpearman. Figure 4.6 presents the percentage of subsets of metrics that contain correlated metrics for each studied feature selection technique. The studied feature selection techniques produce, at the median, 100% of subsets of metrics with collinearity except for findCorrelation and AutoSpearman. Furthermore, except for AutoSpearman, the studied feature selection technique produce, at the median, 3-100% of subsets of metrics with multicollinearity.

Surprisingly, although CFS searches for the best subset of metrics that share the highest correlation with the outcome (e.g., defect-proneness) while having the lowest correlation among each other, our results show that correlated metrics are prevalent in subsets of metrics that are produced by CFS. We observe that CFS tends to focus on the correlation of each metric and the outcome more than the correlation between metrics. Thus, when CFS is applied, correlated metrics are often selected if these correlated metrics share a strong relationship with the outcome.

Furthermore, while findCorrelation can mitigate collinearity, the technique cannot mitigate multicollinearity as its percentage of subsets of metrics with multicollinearity is as high as 100% in some studied defect datasets. We observe that findCorrelation only reduces pair-wise correlations among metrics using a correlation matrix (e.g., Spearman correlation), leading to the mitigation of collinearity. Unfortunately, the usage of pair-wise correlations among metrics of findCorrelation fails to detect and mitigate multicollinearity

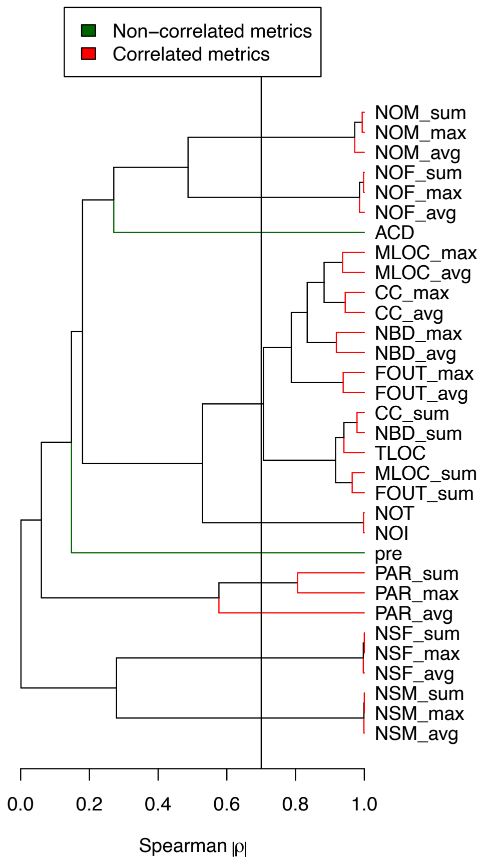
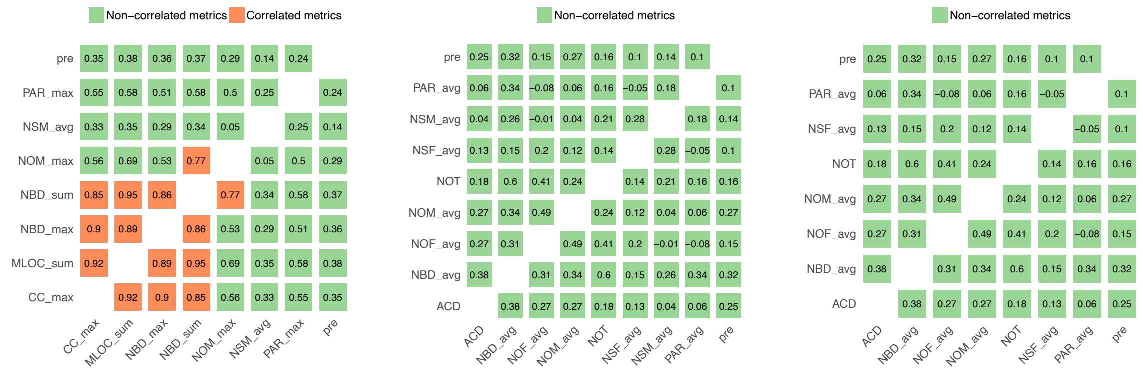


Figure 4.7 The hierarchical cluster view of the correlation analysis of all metrics in the Eclipse Platform 2 dataset. Correlated metrics that can be linearly predicted by another metric and have their Spearman correlation coefficient above 0.7 are highlighted in red, while non-correlated metrics are highlighted in green.

(i.e., a phenomenon in which one metric can be linearly predicted by two or more metrics), suggesting that more advanced techniques (e.g., Variance Inflation Factor analysis) should be applied. Below, we provide detailed illustrative examples.

Illustrative Example. Similar to prior illustrative examples in RQ1, RQ2, and RQ3, we select the Eclipse Platform 2 dataset as the subject of this example. We first draw a bootstrap training sample (*cf.* Step 1 of RQ1) and analyse the correlation among all software metrics, i.e., collinearity and multicollinearity. Then, we apply three feature selection techniques that consider the correlation among metrics when selecting metrics, i.e., a correlation-based feature selection technique (CFS—one of the most commonly-used feature selection technique in the defect prediction domain), findCorrelation, and AutoSpearman. Finally, we analyse the correlation among the metrics in the subsets of metrics that are produced by CFS and findCorrelation.

According to Figure 4.7, among the 32 metrics in the Eclipse Platform 2 dataset, we find that 30 metrics are correlated with a Spearman correlation coefficient of above 0.7. We find that CFS selects 8 metrics that share the highest correlation with defect-proneness while



(a) CFS (b) findCorrelation (c) AutoSpearman

Figure 4.8 The Spearman rank correlation test on the subsets of metrics that are produced by CFS, findCorrelation, and AutoSpearman, respectively. Correlated metrics that can be linearly predicted by another metric and have their Spearman correlation coefficient above 0.7 are highlighted in red, while non-correlated metrics are highlighted in green.

Table 4.8 The Variance Inflation Factor analysis of the subsets of metrics that are produced by the correlation-based feature selection technique (CFS), findCorrelation, and AutoSpearman, respectively. Correlated metrics that can be linearly predicted by a combination of other metrics and have their VIF score of above 5 are highlighted in red.

CFS	
Metric	VIF score
NBD_sum	18.02
MLOC_sum	12.37
NOM_max	5.32
CC_max	3.24
NBD_max	2.18
PAR_max	1.40
pre	1.32
NSM_avg	1.17

findCorrelation	
Metric	VIF score
NSM_avg	7.24
NSF_avg	7.17
NBD_avg	2.10
NOT	1.84
pre	1.29
ACD	1.26
PAR_avg	1.15
NOM_avg	1.12
NOF_avg	1.11

AutoSpearman	
Metric	VIF score
NBD_avg	2.10
NOT	1.83
pre	1.29
ACD	1.26
PAR_avg	1.13
NOM_avg	1.12
NOF_avg	1.11
NSF_avg	1.02

having the lowest correlation among each other; findCorrelation selects 9 metrics through the reduction of pair-wise Spearman correlations among metrics; and AutoSpearman selects 8 metrics based on the results of correlation analyses. Unfortunately, the results of correlation analyses show that CFS and findCorrelation cannot mitigate correlated metrics. According to Figures 4.8a and 4.8b, as many as 5 of the 8 metrics that are selected by CFS can be linearly predicted by another metric and have their Spearman correlation coefficient above 0.7. Furthermore, according to Table 4.8, 3 of the 8 metrics that are selected by CFS and 2 of 9 the metrics that are selected by findCorrelation can be linearly predicted by a combination of other metrics and have their VIF score above 5. These findings suggest that (1) CFS, while considering the correlation among metrics, does not mitigate correlated metrics; and (2) findCorrelation can mitigate collinearity but not multicollinearity.

On the other hand, accroding to Figure 4.8c and Table 4.8, AutoSpearman mitigates correlated metrics for both collinearity and multicollinearity. We observe that the collinearity analysis of AutoSpearman selects one representative metric from each group of correlated metrics. Furthermore, the multicollinearity analysis of AutoSpearman further mitigates correlated metrics that cannot be detected by collinearity analysis (i.e., the analysis of pair-wise correlation among metrics).

AutoSpearman is the only studied feature selection technique that can mitigate correlated metrics, i.e., collinearity and multicollinearity. We observe that other studied feature selection techniques produce up to 100% of subsets of metrics with collinearity and multicollinearity, suggesting that the explanation of defect models constructed using the subsets of metrics that are produced by all of the studied feature selection techniques (except for AutoSpearman) may be misleading.

(RQ6) What is the impact of feature selection techniques on the performance of defect models?

Approach. To address RQ6, we analyse the performance of defect models that are constructed using the subsets of metrics that are produced by the twelve studied feature selection techniques, and a baseline (i.e., all metrics of a defect dataset). We start from the subsets of metrics that are produced by all studied feature selection techniques for each training sample of each studied dataset from RQ1 (*cf.* the Step 2 of RQ1). We then construct defect models using these subsets of metrics and evaluate their performance. Consequently, we analyse the impact of each studied feature selection technique on the model performance. We describe each step below.

(Step 1) Construct defect models. For each training sample, we construct logistic regression and random forest models using subsets of metrics that are produced by the twelve studied feature selection techniques, and all metrics of a defect dataset. We use the implementation of logistic regression as provided by the `glm` function of the `stats` R package [214] with the default parameter setting. We use the implementation of random forest as provided by the

`randomForest` function of the `randomForest` R package [23] with the default `ntree` value of 100, since recent studies [210, 213] show that the performance of random forest models is insensitive to the parameter setting. To ensure that the training and testing corpora share similar characteristics and avoid any potential impact on the explanation of defect models, we do not re-balance nor do we re-sample the training data [208].

(Step 2) Evaluate defect models. In our study, we evaluate defect models using three performance measures. First, we use the Area Under the receiver operator characteristic Curve (AUC) to measure the discriminatory power of our models, as suggested by recent research [52, 114, 174, 207]. The *AUC* is a threshold-independent performance measure that evaluates the ability of classifiers in discriminating between defective and clean modules. The values of AUC range between 0 (worst performance), 0.5 (no better than random guessing), and 1 (best performance) [66]. Second, we use the F-measure, i.e., a threshold-dependent measure. F-measure is a harmonic mean (i.e., $\frac{2\cdot\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$) of precision ($\frac{TP}{TP+FP}$) and recall ($\frac{TP}{TP+FN}$). Similar to prior studies [5, 237], we use the default probability value of 0.5 as a threshold value for the confusion matrix, i.e., if a module has a predicted probability above 0.5, it is considered defective; otherwise, the module is considered clean. Third, we use the Matthews Correlation Coefficient (MCC) measure, i.e, a threshold-dependent measure, as suggested by prior studies [127, 191]. MCC is a balanced measure based on true and false positives and negatives that is computed using the following equation: $\frac{TP\times TN-FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$.

(Step 3) Analyse the impact on the model performance. We analyse the performance difference between defect models that are constructed using subsets of metrics that are produced by feature selection techniques (i.e., the twelve studied feature selection techniques) and all metrics of a defect dataset (P_{FS}–P_{All}). We also set out to investigate whether such performance difference is statistically significant using the Mann-Whitney U test and its extension, the Kruskal and Wallis H test. A p-value of the Mann-Whitney U test of below 0.05 suggests that the difference between two input distributions is statistically significant, while a p-value of the Kruskal and Wallis H test of below 0.05 suggest that the differences across multiple input distributions are statistically significant. Finally, in order to measure the effect size of the performance difference, we use Cliff's $|\delta|$ effect size to analyse the magnitude of the difference between these distributions. Cliff's $|\delta|$ effect size ranges from 0 to +1, where a zero value indicates that two distributions are identical. We use the interpretation of Cliff's $|\delta|$ estimates by Romano *et al.* [187] which maps Cliff's $|\delta|$ to Cohen's significance levels as follows: Negligible: $|\delta| < 0.147$, Small: $0.147 \leq |\delta| < 0.33$, Medium: $0.33 \leq |\delta| < 0.474$, and Large: $0.474 \leq |\delta|$. We present the results using boxplots in Figure 4.9.

Results. The studied feature selection techniques impact the performance of defect models by up to 5%pts. Figure 4.9 shows the performance difference (%pts) between defect models that are constructed using subsets of metrics that are produced by the twelve studied feature selection techniques and all metrics of a defect dataset, i.e., $P_{FS} - P_{All}$. We make three

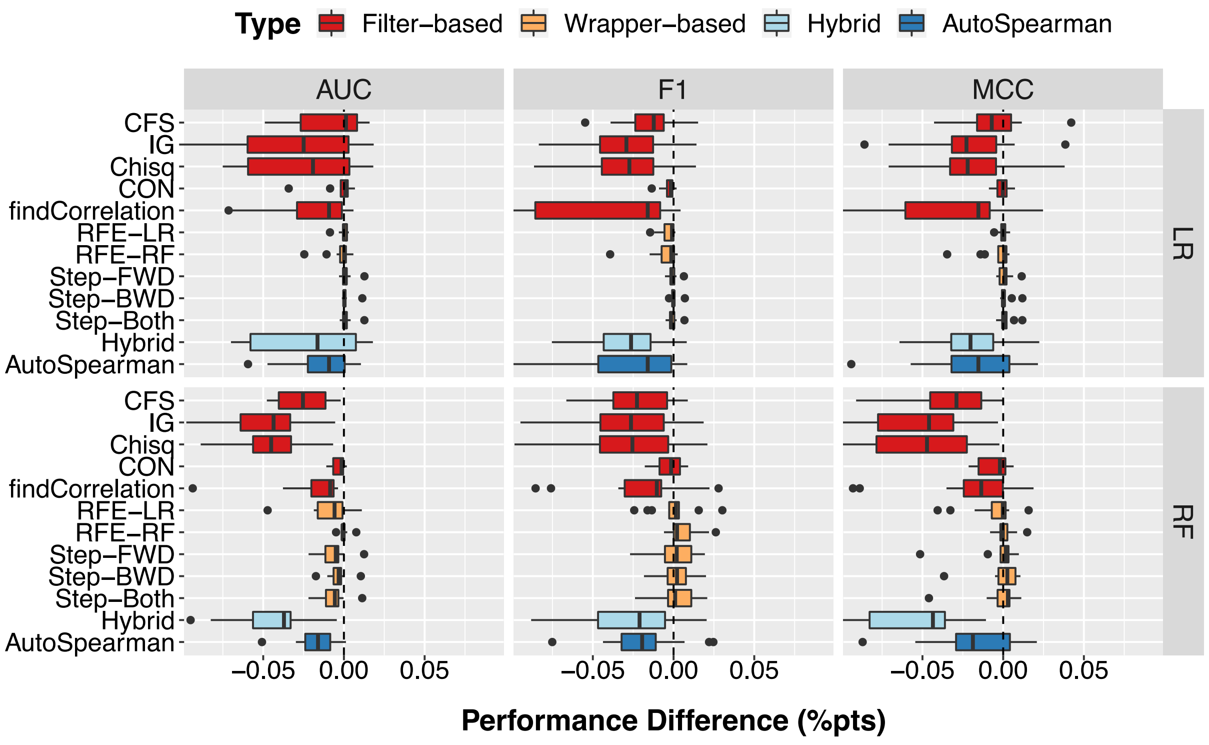


Figure 4.9 The distributions of the performance difference (%pts) between defect models that are constructed using subsets of metrics that are produced by the twelve studied feature selection techniques and all metrics of a defect dataset, i.e., $P_{\text{FS}} - P_{\text{All}}$.

following observations: (1) filter-based feature selection techniques (except for CON) have the highest impact on the performance of defect models by up to 5%pts (at the median); (2) wrapper-based feature selection techniques have the least impact on the performance of defect models by up to 1%pt (at the median); and (3) AutoSpearman impacts the performance of defect models by up to 2%pts (at the median). The results of the Mann-Whitney U test and the Kruskal-Wallis H test suggest that all of the performance differences are not statistically significant with the p-values of above 0.05. Also, our Cliff's $|\delta|$ effect size test shows that such performance differences are negligible to small for the AUC, F-measure, and MCC measures. We suspect that the highest impact on the model performance of filter-based (except for consistency-based) and hybrid feature selection techniques has to do with the removal of metrics that share a strong relationship with the outcome.

We observe that the finding regarding wrapper-based feature selection techniques is consistent with Ghotra *et al.* [53], i.e., the performance of defect models is impacted by at most 2%pts for the AUC measure when applying wrapper-based feature selection techniques. We suspect that such a low impact on the model performance has to do with the low number of irrelevant metrics (e.g., a constant metric) in defect datasets.

The studied feature selection techniques impact the performance of defect models by up to 5%pts. The results of the Mann-Whitney U test and the Kruskal-Wallis H test suggest that all of the performance differences across feature selection techniques are not statistically significant with the p-values of above 0.05. The results of the Cliff's |δ| effect size test also confirm that such performances are negligible to small for the AUC, F-measure, and MCC measures.

(RQ7) What is the computational cost of applying feature selection techniques?

Approach. To address RQ7, we analyse the computational cost of the twelve studied feature selection techniques. For each defect dataset, we use only one bootstrap training sample. Then, we measure the computational cost of applying the twelve studied feature selection techniques on a training sample for each defect dataset. We measure the computational cost using the `microbenchmark` function of the `microbenchmark` R package [144]. Finally, we present the results using boxplots in Figure 4.10. Due to the enormous difference in the computation cost across the studied feature selection techniques, we report the computational cost (x-axis) using a `log10` scale.

Results. **The computational cost of wrapper-based feature selection techniques and the consistency-based feature selection technique is expensive.** Figure 4.10 shows the computational cost analysis of the studied feature selection techniques using a `log10` scale. We observe that, at the median, the computational cost of the studied feature selection techniques is 0.67s, 0.29s, 0.19s, 186s, 0.18s, 135.3s, 8965s (2 hours and 29 minutes), 6.92s, 3.27s, 12.53s, 1.53s, and 0.22s for CFS, IG, Chisq, CON, findCorrelation, RFE-LR, RFE-RF, Step-FWD, Step-BWD, Step-BOTH, Hybrid, and AutoSpearman, respectively. In particular, wrapper-based feature selection techniques and the consistency-based feature selection technique are expensive to compute, and can cost up to 7 hours and 30 minutes for RFE-RF to find the best subset of metrics from one training sample of the Proprietary 2 defect dataset. Such expensive computation cost makes the application of wrapper-based feature selection techniques undesirable, particularly, when validating with model validation techniques that require several repetitions (e.g., bootstrap validation technique).

The computational cost of filter-based feature selection techniques and AutoSpearman is cheap, while such cost is expensive for wrapper-based feature selection techniques and the consistency-based feature selection technique. The computational cost of wrapper-based feature selection techniques is as high as 7 hours and 30 minutes for RFE-RF to find the best subset of metrics from one training sample of the Proprietary 2 defect dataset. Such expensive computation cost makes the application of wrapper-based feature selection techniques undesirable, particularly, when validating with model validation techniques that require several repetitions (e.g., bootstrap validation technique).

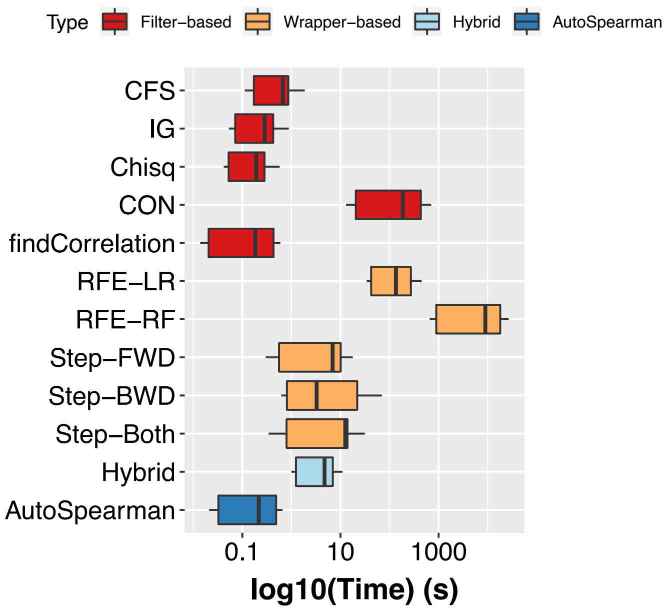


Figure 4.10 The computational cost analysis of the twelve studied feature selection techniques.

(RQ8) Do correlation threshold values have an impact on the explanation of defect models?

Approach. To address RQ8, we analyse the impact of correlation threshold values when producing subsets of metrics with AutoSpearman on the explanation of defect models along 2 dimensions, i.e., consistency and correlation. AutoSpearman relies on two threshold values, i.e., the Spearman correlation threshold and the VIF threshold. While we use the Spearman correlation threshold value of 0.7 as a baseline as suggested by Kraemer *et al.* [107] and the VIF threshold value of 5 as a baseline as suggested by Fox *et al.* [44] to indicate strong correlations between metrics, we perform sensitivity analyses for both threshold values. For the sensitivity analysis of the Spearman correlation threshold, we use the VIF threshold value of 5 and compare the explanation produced by the Spearman correlation threshold values of 0.8 and 0.9 with those produced by the baseline. Similarly, for the sensitivity analysis of the VIF threshold, we use the Spearman correlation threshold value of 0.7 and compare the explanation produced by the VIF threshold values of 3 and 10 with those produced by the baseline. Below, we explain how we generate the explanation of defect models and how we analyse the impact of correlation threshold values on such explanation.

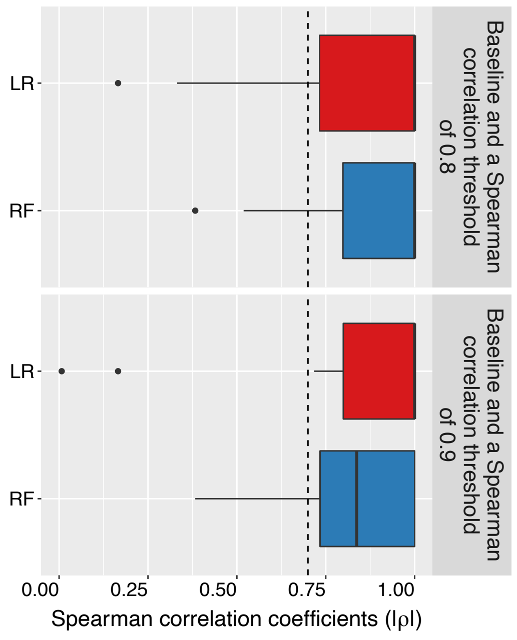
To generate the explanation of defect models, we adopted the defect modelling workflow used in prior studies [85]. First, we mitigate correlated metrics using our proposed feature selection technique, AutoSpearman. Second, we generate training and testing samples using the out-of-sample bootstrap validation technique. Third, we use training samples to construct defect models (i.e., logistic regression and random forests). Fourth, we generate the importance score of metrics using the ANOVA Type-II analysis [43] for logistic regression and the scaled permutation importance analysis [21] for random forests. Finally, we identify

Table 4.9 The percentage of the studied defect datasets in which their most important metrics are consistent among correlation threshold values.

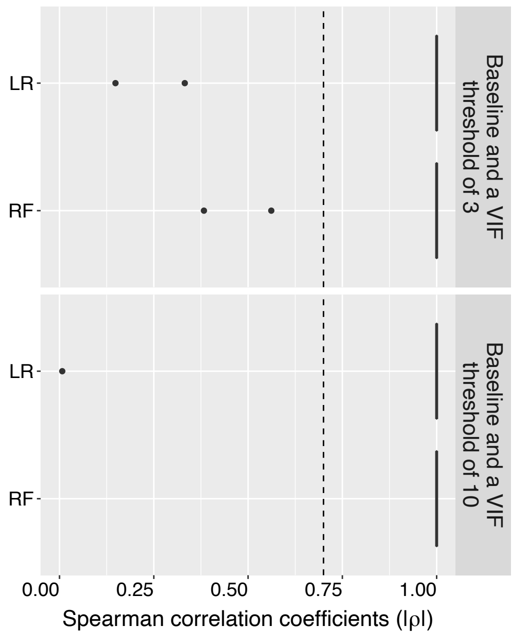
Classification Techniques	Model Explanation Techniques	Spearman Correlation Threshold		VIF Threshold	
		0.8	0.9	3	10
Logistic Regression	ANOVA Type 2	69%	62%	85%	92%
Random Forests	Scaled Permutation Importance	69%	38%	85%	100%

the importance ranking of metrics using the improved Scott-Knott Effect Size Difference (ESD) test [206].

We analyse the consistency and correlation of the most important metrics among correlation threshold values. To do so, we compute the percentage of the studied datasets in which their most important metrics are consistent among correlation threshold values. We also compute the Spearman correlation of the most important metrics produced by different correlation threshold values.



(a) Spearman correlation



(b) VIF

Figure 4.11 The distributions of Spearman correlation coefficients of the most important metrics produced by the baseline and correlation threshold values.

Results. **Different correlation threshold values may produce different most important metrics.** Table 4.9 shows the percentage of the studied defect datasets in which their most important metrics are consistent among correlation threshold values. We find that 69% and 38-62% of the studied defect datasets has their most important metrics produced by the Spearman correlation threshold values of 0.8 and 0.9 consistent with the baseline

Table 4.10 The top-5 most important metrics according to the percentage of the studied defect datasets in which their most important metrics are consistent among correlation threshold values.

Rank	AutoSpearman(0.7)	AutoSpearman(0.9)
1	NOM_avg	NBD_sum
2	NBD_avg	NBD_avg
3	pre	NOM_avg
4	PAR_avg	FOUT_avg
5	NOF_avg	pre

(the Spearman correlation threshold value of 0.7), respectively. We also find that 85% and 92-100% of the studied defect datasets has their most important metrics produced by the VIF threshold values of 3 and 10 consistent with the baseline (the VIF threshold value of 5), respectively. In other words, 31-62% of these studied datasets produces different most important metrics among Spearman correlation threshold values, while such inconsistency among VIF threshold values are 0-15%.

The most important metrics produced by different correlation threshold values are highly correlated. Figure 4.11 shows the distributions of Spearman correlation coefficients of the most important metrics produced by the baseline and correlation threshold values. We find that, at the median, Spearman correlation coefficients of the most important metrics produced by the baseline and the Spearman correlation threshold values of 0.8 and 0.9 are 1 and 0.84-1 for both logistic regression and random forests. Similarly, at the median, Spearman correlation coefficients of the most important metrics produced by the baseline and the VIF threshold values of 3 and 10 are 1 for logistic regression and random forests, respectively. These findings show that while different correlation threshold values may produce different most important metrics, such different most important metrics are highly correlated. The experimental results lead us to conclude that correlation threshold values do not impact the explanation of defect models.

Illustrative Example. Similar to previous research questions, we use the Eclipse Platform 2 dataset as the subject of this example. Following the approach to generate the explanation of defect models as explained above, we identify the importance rankings of metrics produced by AutoSpearman with the correlation threshold values of 0.7 (baseline) and 0.9 as shown in Table 4.10. According to Figure 4.12, we find that NOM_avg (the most important metric according to AutoSpearman(0.7)) and NBD_sum (the most important metric according to AutoSpearman(0.9)) are highly correlated with the Spearman correlation of 0.73. This example further illustrates that the little impact of correlation threshold values on the explanation of defect models.

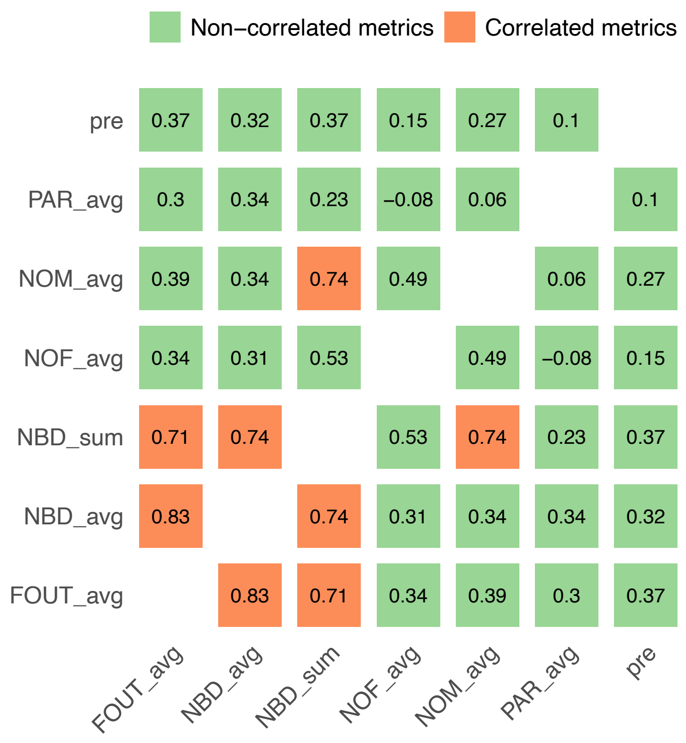


Figure 4.12 The Spearman rank correlation test on the top-5 important metrics according to AutoSpearman using the correlation threshold values of 0.7 and 0.9. Correlated metrics that can be linearly predicted by another metric and have their Spearman correlation coefficient above 0.7 are highlighted in red, while non-correlated metrics are highlighted in green.

No. Indeed, 0–62% of the studied defect datasets produces different most important metrics among correlation threshold values. However, such different most important metrics are highly correlated with the spearman correlation of 0.84–1, suggesting that correlation threshold values do not impact the explanation of defect models.

4.5 Discussions

4.5.1 The trends of correlated metrics that are elected by the commonly-used feature selection techniques

Approach. To investigate the trends of correlated metrics that are selected by the commonly-used feature selection techniques, similar to the illustrative example in RQ1, we first select the Eclipse Platform 2 dataset as the subject of this analysis. We draw a bootstrap training sample and apply all studied feature selection techniques to generate subsets of metrics for each studied technique. Then, we analyse the correlation among metrics for both collinearity and multicollinearity to find correlated metrics in each subset of metrics produced by the studied feature selection techniques (*cf.* Steps 1 and 2 of RQ5). Finally, we identify (1) the number of

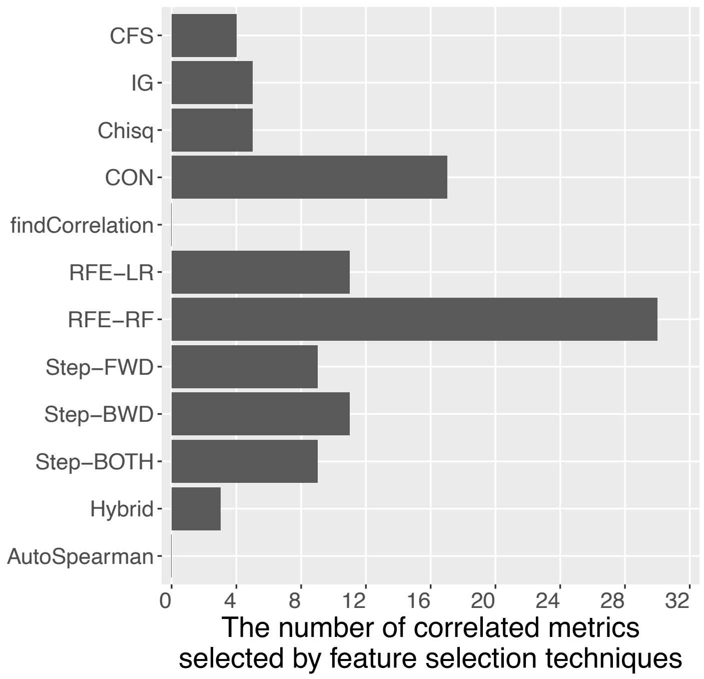


Figure 4.13 The number of correlated metrics selected by each studied feature selection technique for an illustrative analysis of the Eclipse Platform 2 dataset.

correlated metrics selected by each feature selection technique and (2) the number of feature selection techniques that select each correlated metric.

Results. **Most of the correlated metrics selected by commonly-used feature selection techniques are aggregated metrics produced by metric aggregation schemes.** Figure 4.14 shows the number of studied feature selection techniques that select each correlated metric for an illustrative analysis of the Eclipse Platform 2 dataset. We find that correlated metrics are selected by 1–9 studied feature selection techniques. We observe that most of correlated metrics are aggregated metrics—metrics that are extracted at the method level and are summarised to the file level using metric aggregation schemes, e.g., minimum, average, and maximum. This observation is consistent with that of Zhang *et al.* [235] where metric aggregation schemes often produce correlated metrics.

4.6 Threats to Validity

Construct Validity. Plenty of prior work show that the parameters of classification techniques have an impact on the performance of defect models [49, 106, 133, 134, 210]. While we use a default `ntree` value of 100 for random forest models, recent studies [80, 210, 220] show that the performance of random forest models is insensitive to this parameter setting. Thus, we believe that this threat is not a major limitation of our work.

The concept of non-correlated metrics in our paper relies on threshold values of correlation analyses (i.e., 0.7 for a Spearman rank correlation test and 5 for a Variance Inflation

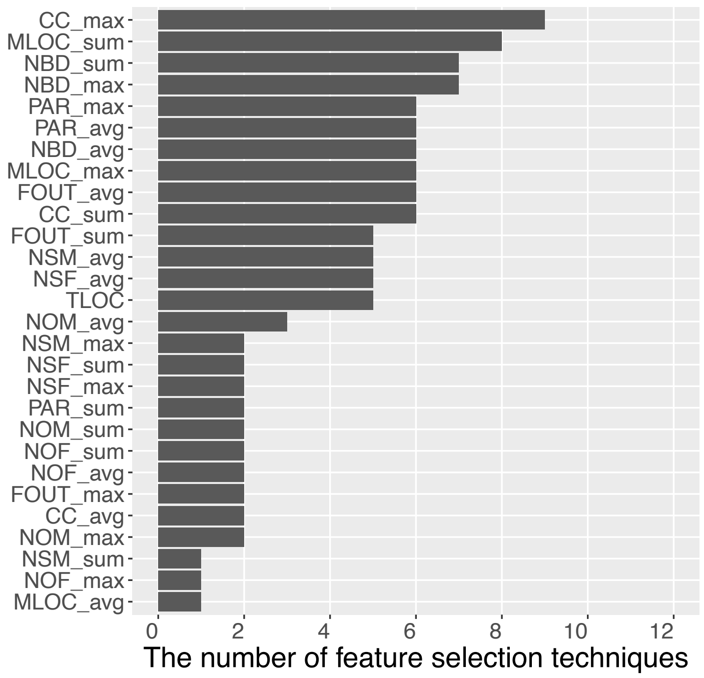


Figure 4.14 The number of studied feature selection techniques that select each correlated metric for an illustrative analysis of the Eclipse Platform 2 dataset.

Factor analysis). To mitigate this threat, we perform an in-depth sensitivity analysis in RQ8 and find that correlation threshold values do not impact the explanation of defect models.

Internal Validity. Prior studies raise concerns related to the replicability [184], the data quality [168, 192], and the risk of overfitting [212]. Nevertheless, we conduct a highly-controlled experiment where we apply 4 dataset selection criteria to mitigate these concerns. Thus, our study focuses 13 defect datasets from 4 corpora, i.e., 5 datasets as provided by [94], 3 datasets as provided by D'Ambros *et al.* [33, 34], 3 datasets as provided by Zimmermann *et al.* [238], and 2 datasets as provided by Kim *et al.* [101] and Wu *et al.* [230].

External Validity. We studied a limited number of feature selection techniques. Thus, our results may not generalise to other feature selection techniques. Nonetheless, other feature selection techniques can be explored in future work. We provide a detailed methodology for others who would like to re-examine our findings using unexplored feature selection techniques.

Recent work [119, 224, 225] used advanced deep learning techniques to automatically extract semantic features for predicting software defects. However, such semantic features may impact distributions of original features and transform them into an unexplainable form of features. Since the ultimate goal of this study is to constructing explainable defect prediction models, we therefore excluded these approaches from our studied techniques.

The studied defect datasets are part of several systems (e.g., Eclipse) that span both proprietary and open source domains. However, we studied a limited number of defect datasets. Thus, the results may not generalise to other datasets and domains. Replication studies are needed. The conclusions of our study rely on one defect prediction scenario (i.e., within-project defect models). However, there are a variety of defect prediction scenarios in the literature (e.g., cross-project defect prediction [27, 237], just-in-time defect prediction [96], heterogenous defect prediction [157]). Therefore, the conclusions may differ for other scenarios. Thus, future research should revisit our study in other scenarios of defect models.

4.7 Conclusions

In this chapter, we investigate 11 commonly-used feature selection techniques and our own contribution AutoSpearman along 5 dimensions: (1) the consistency of the produced subsets of metrics; (2) the correlation of the produced subsets of metrics; (3) the performance; (4) the computational cost; and (5) the impact on the explanation. Through a study of 13 publicly-available defect datasets of systems that span both proprietary and open source domains, we find that the selected metrics of the commonly-used techniques (except for Stepwise Regression) are consistent regardless of the ordering of metrics in model specifications. However, we find that (1) 94-100% of the selected metrics are inconsistent among the studied techniques; (2) 31-94% of the selected metrics are inconsistent among training samples; (3) 0-62% of the selected metrics are inconsistent when the application of the feature selection techniques is repeated; and (4) 3-100% of the produced subsets of metrics contain correlated metrics—suggesting that the commonly-used automated feature selection techniques are often unreliable. We also find that (5) 0–62% of the studied defect datasets produces different most important metrics among correlation threshold values while such different most important metrics are highly-correlated with the Spearman correlation of 0.84–1.

We are the first to provide empirical evidence that commonly-used feature selection techniques in the defect prediction domain are inconsistent in nature. Such inconsistent nature of these feature selection techniques (1) restricts an application of post-hoc multiple comparison analyses (e.g., a Scott-Knott test) to identify the most important metrics when explaining defect models; and (2) has a negative impact on the explanation of defect models due to the presence of correlated metrics. Thus, to mitigate these concerns, the results of our empirical investigations lead us to recommend AutoSpearman be used in future studies, since AutoSpearman yields higher consistency than commonly-used feature selection techniques and automatically mitigates correlated metrics. We also provide the implementation of AutoSpearman as an R package [87]. Nevertheless, future research effort should develop and explore a new automated feature selection technique that mitigates these concerns (e.g., produces subsets of metrics that are consistent and have no correlations among themselves).

Finally, we would like to emphasise that the goal of this work is not to claim the generalisation of our results for every dataset and every model in software engineering. In addition, the best subset of metrics that one should include in studies depends on the goal of the studies. For example, if the goal of the study is prediction (i.e., aiming to achieve the highest predictive performance), one might prioritise resources on improving the model performance regardless of the correlation among metrics. On the other hand, if the goal of the study is model explanation (i.e., aiming to examine the impact of various phenomena on software quality), one should avoid using commonly-used feature selection techniques when explaining defect models. To mitigate the inconsistent nature of automated feature selection techniques, we recommend AutoSpearman be used in future studies when explaining defect models.

4.7.1 Chapter Remarks

In this chapter, we investigate techniques that automatically mitigate correlated metrics when explaining defect models. The experimental results show that, when comparing to commonly-used feature selection techniques, our contribution, AutoSpearman, yields higher consistency and automatically mitigates correlated metrics. Thus, to derive the best defect modelling workflow that produces the most accurate and reliable explanation of defect models, the experimental results lead us to suggest that future research should automatically mitigate correlated metrics with AutoSpearman prior to constructing and explaining defect models.

According to the experimental results in Chapters 3 and 4, one should automatically mitigate correlated metrics with AutoSpearman prior to constructing defect models; and use the ANOVA Type-II technique to explain logistic regression models, while using the scaled Permutation Importance technique to explain random forests models. However, these model explanation techniques cannot justify each individual prediction of the models on testing or unseen data. Thus, in the next chapter, we set out to investigate the best techniques for explaining the predictions of defect models.

Chapter 5

Explain the Predictions of Defect Models

An earlier version of the work in this chapter appears in the Transaction on Software Engineering (TSE) [82].

5.1 Introduction

Software analytics have empowered many software organisations to improve software quality and accelerate software development processes. Such analytics are essential to guide operational decisions and establish quality improvement plans. For example, Microsoft leverages the advances of Artificial Intelligence and Machine Learning (AI/ML) capabilities to predict software defects [154]. In addition, prior studies have proposed techniques to estimate Agile story points [31], estimate software development costs [159], recommend a reviewer [218], recommend a developer to fix a software defect [4].

Despite the recent advances in software analytics, such decision-making based on AI/ML-based systems needs to be better justified and uphold privacy laws. Article 22 of the European Union's General Data Protection Regulation (GDPR) [179] states that the use of data in decision-making that affects an individual or group *requires an explanation for any decision made by an algorithm*. Recent work raises a concern about a lack of explainability of software analytics in software engineering [32]. Practitioners also share similar concerns that analytical models in software engineering must be explainable and actionable [32, 115]. For example, Google [115] argues that defect models should be more actionable to help software engineers debug their programs. Miller [145] also argues that human aspects should be taken into consideration when developing AI/ML-based systems. Thus, *Explainable Software Analytics*—a suite of AI/ML techniques that produce accurate predictions, while being able to explain such predictions—is vitally needed.

Thus, researchers often generated ***global explanations***, which refers to an explanation that summarises the predictions of black-box AI/ML learning algorithms. Such global explanations can be generated by model-specific explanation techniques of machine learning techniques (e.g., an ANOVA analysis for logistic regression and a variable importance analysis for random forests). Prior studies used these model interpretation techniques to understand the relationship between studied variables and an outcome. For example, Menzies *et al.* [140] investigated the impact of code attributes on software quality. Bird *et al.* [17] studied the correlation between human factors and software quality. McIntosh *et al.* [131] and Thongtanunam *et al.* [216] investigated the relationship between code review practices and post-release defects.

However, such global explanations cannot justify each individual prediction of the models on testing or unseen data. For example, an analytical model for software defects may generate a predicted probability of 0.9 for a testing instance, suggesting that a software module will be defective in the future. Such the predicted probability does not provide any explanation from the models as to why the machine learning techniques make that prediction. A lack of explanation of the predictions generated by such analytical models could lead to serious errors in decision- and policy-making, hindering the adoption of software analytics in industrial practices [32].

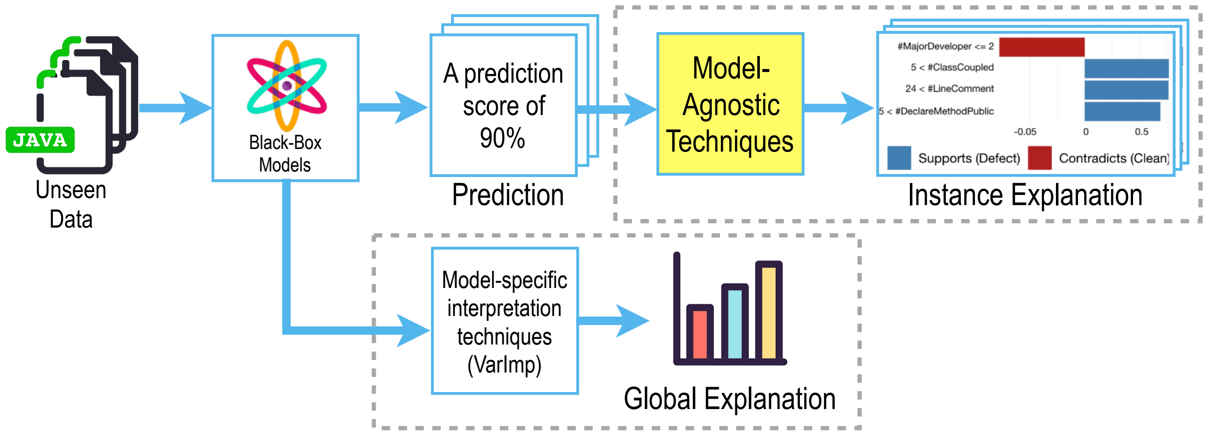


Figure 5.1 An illustration of model-agnostic techniques. Model-agnostic techniques are used to explain the predictions of unseen data, while the global explanation is derived from the trained models from training data. In other words, one model can have only one global explanation, but should have multiple instance explanations.

Recently, ***model-agnostic techniques*** have been proposed to explain the prediction of black-box AI/ML algorithms by identifying the contribution that each metric has on the prediction of an instance according to a trained model. Yet, such techniques have never been formally introduced and empirically evaluated in the context of software engineering. To address this challenge, this study is the first to focus on generating ***instance explanations*** which refers to an explanation of the prediction of defect prediction models (see Figure 5.1), by answering a central question: *Should model-agnostic techniques be used to explain the predictions of defect models?*

In this paper, we empirically evaluate three model-agnostic techniques, i.e., two state-of-the-art Local Interpretability Model-agnostic Explanations (LIME) technique [182] and BreakDown [55, 201] technique, and our improvement of LIME with Hyper Parameter Optimisation (LIME-HPO) using a differential evolution algorithm. *LIME* constructs a local regression model surrounding the instance to be explained to identify the contribution of each metric to the prediction of the instance to be explained. On the other hand, *BreakDown* decomposes the prediction of the instance to be explained into parts that can be attributed to each studied metric as to their contribution to the prediction. We generate explanations of the predictions of defect models that are constructed from six classification techniques (i.e., logistic regression (LR), random forests (RF), C5.0, averaged neural network (AVNNet), gradient boosting machines (GBM), and extreme Gradient Boosting Trees (xGBTree)). Through a study of 32 publicly-available defect datasets of 9 large-scale open-source software systems, we address the following six research questions:

(RQ1) Does LIME with Hyper Parameter Optimisation (LIME-HPO) outperform default LIME in terms of the goodness-of-fit of the local regression models?
LIME-HPO always outperform default LIME in terms of the goodness-of-fit (R^2)

of the local regression models with an average improvement of 8% for all of the studied classification techniques.

(RQ2) **Can model-agnostic techniques explain the predictions of defect models?**
Model-agnostic techniques can explain the predictions of defect models. Given the same defect models, different predictions have different instance explanations. For example, one metric that appears at the top rank for one instance could appear at the rank 21 for another instance. Such high variation indicates that global explanations do not imply instance explanations (and vice versa), highlighting the need for model-agnostic techniques for explaining the predictions of defect models.

(RQ3) **Do instance explanations generated by model-agnostic techniques overlap with the global explanation of defect models?**
Despite the variation of the ranking of the top-10 important metrics for instance explanations (see RQ2), their overall ranking is mostly overlapping (but not exactly the same) with that of the global explanations. We find that, at the median, 7, 10, and 9 of the top-10 summarised important metrics of for instance explanations are overlapping with the top-10 global important metrics for LIME, LIME-HPO, and BreakDown, respectively.

(RQ4) **Do model-agnostic techniques generate the same instance explanation when they are re-generated for the same instance?**
Regardless of the studied classification techniques, LIME-HPO and BreakDown consistently generate the same instance explanation for the same instance. On the other hand, LIME generates different instance explanations when re-generating instance explanations of the same instance with a median rank difference of 7 ranks.

(RQ5) **What is the computational time of model-agnostic techniques for explaining the predictions of defect models?**
The computational time of LIME-HPO, BreakDown, and LIME techniques is less than a minute to generate instance explanations for all of the studied classification techniques, suggesting that all of the studied model-agnostic techniques are practical to be used in real-world deployments in the future.

(RQ6) **How do practitioners perceive the contrastive explanations generated by model-agnostic techniques?**
65% of the practitioners agree that model-agnostic techniques can generate a contrastive explanation within an object over time (Time-contrast) (e.g., why was file *A* not classified as defective in version 1.2, but was subsequently classified as defective in version 1.3?). In particular, 55% and 65% of the participants perceive that such Time-contrast explanations are necessary and useful, respectively.

Since the implementation of the studied model-agnostic techniques is readily available in both Python (LIME [180] and pyBreakDown [14]) and R (LIME [164] and BreakDown [15, 55]), we recommend model-agnostic techniques be used to explain the predictions of defect models.

Novelty & Contributions. The key contributions of this study are as follows:

(1) An introduction to the explainability in software engineering from a perspective of psychological science (Section 5.2).

(2) An introduction to the state-of-the-art model-agnostic techniques (i.e., LIME and BreakDown) for generating instance explanations and our contribution approach – An improvement of LIME using Hyper Parameter Optimisation (LIME-HPO) with a differential evolution algorithm (Section 5.3).

(3) An empirical evidence of an improvement of LIME-HPO over LIME in terms of the goodness-of-fit of the local regression models (RQ1) (Section 5.4).

(4) An empirical study of the need (RQ2), trustworthiness (RQ3), reliability (RQ4), computational time (RQ5), and software practitioners' perception (RQ6) of model-agnostic techniques (Section 5.4).

5.1.1 Chapter Organisation

Section 5.2 introduces the explainability in software engineering. Section 5.3 introduces model-agnostic techniques for generating instance explanation. Section 5.4 presents the design of our study, while Section 5.5 discusses the results with respect to six research questions. Section 5.6 discusses the key differences between model-agnostic techniques and a simple tree-based technique. Section 5.7 discusses related work in order to situate the contributions of our paper with respect to explainable software analytics and analytical models for software defects. Section 5.8 discusses the threats to the validity of our study. Finally, Section 5.9 draws conclusions.

5.2 Explainability in Software Engineering

Software engineering is by nature a collaborative social practice. Collaboration among different stakeholders (e.g., users, developers, and managers) is essential in modern software engineering. As a part of the collaboration, individuals are often expected to explain decisions made throughout software development processes to develop appropriate trust and enable effective communication. Since tool support in software development processes is an integral part of this collaborative process, similar expectations are also applied. Such tools should not only provide insights or generate predictions for recommendation, but also be able to explain such insights and recommendations.

Recent automated and advanced software development tools heavily rely on Artificial Intelligence and Machine Learning (AI/ML) capabilities to predict software defects, estimate development effort, and recommend API choices. However, such AI/ML algorithms are often "black-box", which makes it hard for practitioners to understand how the models arrive at a decision. A lack of explainability of the black-box algorithms leads to a lack of trust in the predictions or recommendations produced by such algorithms.

5.2.1 A Theory of Explainability

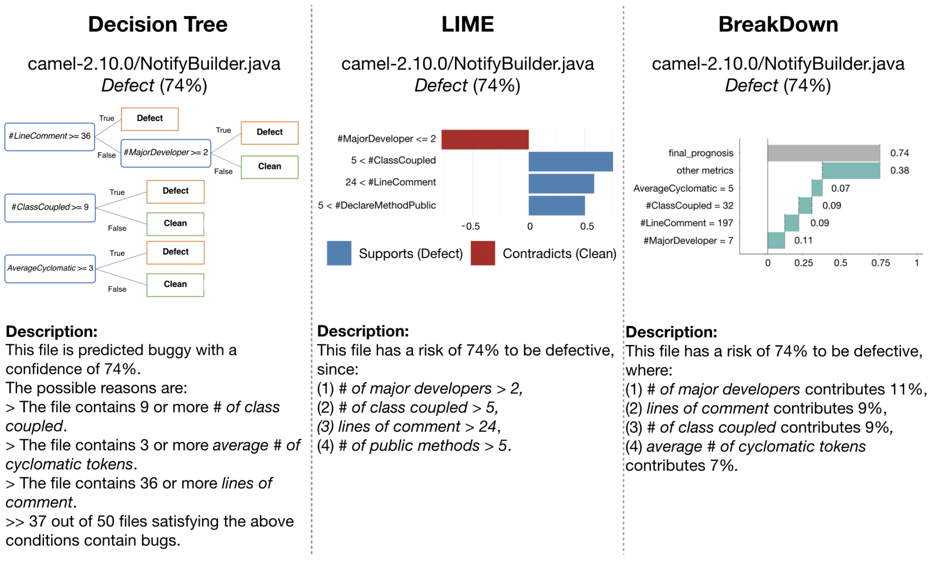
According to philosophy, social science, and psychology theories, a common definition of *explainability or interpretability* is *the degree to which a human can understand the reasons behind a decision or an action* [146]. The explainability of AI/ML algorithms can be achieved by (1) making the entire decision-making process transparent and comprehensible and (2) explicitly providing an explanation for each decision [121] (since an explanation is not likely applicable to all decisions [113]). In order to make the entire decision-making process transparent, prior software engineering studies often use white-box AI/ML algorithms (e.g., decision trees and decision rules). While such white-box AI/ML algorithms can increase the explainability of the decision-making process, their predictions may not be as accurate as complex black-box AI/ML techniques (e.g., random forest, extreme gradient boosting trees). Hence, research has emerged to explore how to explain decisions made by complex, black-box models and how explanations are presented in a form that would be easily understood (and hence, accepted) by humans.

5.2.2 A Theory of Explanations

According to a philosophical and psychological theory of explanations, Salmon *et al.* [189] argue that explanations can be presented as a causal chain of causes that lead to the decision. Causal chains can be classified into five categories [75]: temporal, coincidental, unfolding, opportunity chains and pre-emptive. Each type of causal chain is thus associated with an explanation type. However, identifying the complete causal chain of causes is challenging, since most AI/ML techniques produce only correlations instead of causations.

In contrast, Miller [146] argue that explanations can be presented as answers to why-questions. Similarly, Lipton *et al.* [120] also share a similar view of explanations as being *contrastive*. There are three components of why-questions [8]: (1) the event to be explained, also called the *explanandum* (e.g., file A is defective); (2) a set of similar events that are similar to the explanandum but did not occur (e.g., file A is clean); and (3) a request for information that can distinguish the occurrence of the explanandum from the non-occurrence of the other similar events (e.g., a large number of changes made to file A). Below, we describe four types of why-questions:

- **Plain-fact** is the properties of the object. *"Why does object a have property P?"*
  Example: Why is file *A* defective?



Description:
This file is predicted buggy with a confidence of 74%.
The possible reasons are:
> The file contains 9 or more *# of class coupled*.
> The file contains 3 or more *average # of cyclomatic tokens*.
> The file contains 36 or more *lines of comment*.
>> 37 out of 50 files satisfying the above conditions contain bugs.

Description:
This file has a risk of 74% to be defective, since:
(1) *# of major developers > 2,*
(2) *# of class coupled > 5,*
(3) lines of comment > 24,
(4) *# of public methods > 5.*

Description:
This file has a risk of 74% to be defective, where:
(1) *# of major developers* contributes 11%,
(2) *lines of comment* contributes 9%,
(3) *# of class coupled* contributes 9%,
(4) *average # of cyclomatic tokens* contributes 7%.

Figure 5.2 An example of visual explanations for a decision tree model, and two model-agnostic techniques (i.e., LIME and BreakDown).

- **Property-contrast** is the differences in the Properties within an object. *"Why does object a have property P, rather than property P′?"*
  Example: Why is file *A* defective rather than clean?

- **Object-contrast** is the differences between two Objects. *"Why does object a have property P, while object b has property P′?"*
  Example: Why is file *A* defective, while file *B* is clean?

- **Time-contrast** is the differences within an object over Time. *"Why does object a have property P at time t, but property P′ at time t′?"*
  Example: Why was file *A* not classified as defective in version 1.2, but was subsequently classified as defective in version 1.3?

Answers to the P-contrast, O-contrast and T-contrast why-questions form an explanation. *Contrastive explanations focus on only the differences on* ***Properties within an object*** *(Property-contrast), between* ***two Objects*** *(Object-contrast), and* ***within an object over Time*** *(Time-contrast) [221].* Answering a plain fact question is generally more difficult than generating answers to the contrastive questions [120]. For example, we could answer the Property-contrast question (e.g., "Why is file *A* classified as being defective instead of being clean?") by citing that there are a substantial number of defect-fixing commits that involve with the file. Information about the size, complexity, owner of the file, and so on are not required to answer this question. On the other hand, explaining why file *A* is defective in a non-contrastive manner would require us to use all causes. In addition, humans tend to be cognitively attached to digest contrastive explanations [146]. Thus, contrastive explanations may be more valuable and more intuitive to humans. These important factors from both social and computational perspectives should be considered when providing explainable models or tool support for software engineering.

Explanation is not only a *product*, as discussed above, but also a *process* [123]. In fact, generating explanations is a *cognitive process* which essentially involves four cognitive systems: (1) attention, (2) long-term memory, (3) working memory, and (4) metacognition [77, 113]. Recent work [146] further recognised the importance of considering explanation as being not only a cognitive process but also a *social process*, in which an explainer communicates knowledge to an explainee. Using this view, explanations should be considered as part of a conversation between the explainer and explainee. The theories, models, and processes of how humans explain decisions to one another are important to the work on explainable software analytics and the development of explainable tool support for software engineering in general.

5.3 Techniques for Generating Explanations

Prior studies often leverage *white-box AI/ML techniques*, such as decision trees [46] and decision rules [171]. The transparency of such white-box AI/ML techniques allows us to meaningfully understand the magnitude of the contribution of each metric on the learned outcomes by directly inspecting the model components. For example, the coefficients of each metric in a regression model, paths in a decision tree, or rules of a decision rule model. Figure 5.2 provides an example visual explanation of a white-box model (e.g., decision trees) and model-agnostic techniques.

In contrast, white-box AI/ML techniques are often less accurate than complex black-box AI/ML techniques and often generate generic explanations (e.g., one decision node may cover 100 instances). Recently, model-agnostic techniques (e.g., LIME [182] and BreakDown [55]) have been used to explain the predictions of any black-box AI/ML models at an instance level. Guiding by a theory of explanations in Section 5.2.2, this study focuses on answering the why-questions at an instance level for any predictions made by a black-box model. Below, we present the formal definition of a black-box model, a global explanation, and an instance explanation.

Definition 1.1: A black-box model. A black-box model is a function $b : \mathcal{X}^{(m)} \rightarrow \mathcal{Y}$ where $\mathcal{X}^{(m)}$ is the feature space with m corresponding to the studied metrics (i.e., independent variables), and $\mathcal{Y}$ is the outcome space (e.g., defective or clean). Typically, training data D_{train} is used for training a black-box model $b(D_{\text{train}})$, and testing data D_{test} is used for evaluating the accuracy of a black-box model b.

There are a plethora of techniques for generating explanations from these black-box models where each technique has different definitions and targets of explanations. Below, we provide formal definitions and introduce techniques for explaining a black-box model and techniques for explaining an individual prediction made by a black-box model.

5.3.1 Explaining a black-box model

A **global explanation** (or a model explanation) refers to an explanation of the decisions of a black-box model which summarises the logic of a classification technique based on the conditional relationship between the independent variables (software metrics) and the dependent variable (an outcome) with respect to **a whole training data**. Below, we present the formal definition of global explanation and model-specific explanation techniques as follows:

Definition 1.2: global explanation. A global explanation $e_m = \varepsilon(b, D_{\text{train}})$, if a global explanation e_m is generated from an explanation function ε which summarises the logic of a black-box model b that is learned from a training dataset D_{train}.

Model-specific explanation techniques focus on explaining the entire decision-making process of a specific black-box model. For example, an ANOVA analysis for logistic regression and a variable importance analysis for random forests. However, such global explanations are often derived from black-box models that are constructed from training data, which are not specific enough to explain an individual prediction.

5.3.2 Explaining an individual prediction

An **instance explanation** (or a local explanation) refers to an explanation of the decision of a black-box model with respect to **a testing instance**. Below, we present the formal definition of instance explanation and model-agnostic techniques as follows:

Definition 1.3: instance explanation An instance explanation $e_o = \varepsilon(b, x)$, if an instance explanation e_o is generated from an explanation function ε for a prediction $b(x)$ of an instance $x \in D_{\text{test}}$ in a testing dataset D_{test}.

Model-agnostic techniques (i.e., local explanation techniques) focus on explaining an individual prediction by diagnosing a black-box model. Unlike model-specific explanation techniques discussed above, the great advantage of model-agnostic techniques is their flexibility. Such model-agnostic techniques can (1) interpret any classification techniques (e.g., regression, random forest, and neural networks); (2) are not limited to a certain form of explanations (e.g., feature importance or rules); and (3) are able to process any input data (e.g., features, words, and images [181]). According to the literature survey of model-agnostic techniques [58], we selected and discussed two state-of-the-art model-agnostic techniques (i.e., LIME and BreakDown). We also propose LIME-HPO—an improvement of LIME using Hyper Parameter Optimisation based on the differential evolution technique.

LIME: Explaining a local model that mimics the global model

LIME (i.e., Local Interpretable Model-agnostic Explanations) [182] is a model-agnostic technique that mimics the behaviour of the black-box model to generate the explanations of the predictions of the black-box model. Given a black-box model and an instance to explain, LIME performs 4 key steps to generate an instance explanation as follows:

Algorithm 2: LIME's algorithm [182]

```
Input  : $b$ is a black-box model,
         $x$ is an instance to explain,
         $n$ is a number of randomly generated
         instances, and
         $k$ is a length of explanation
Output : $e$ is a set of contributions of metrics on the prediction of the instance $x$.
1 $D = \varnothing$
2 for $i$ in $\{1, ..., n\}$ do
3     $d_i$ = sample_around($x$)
4     $y'_i$ = predict($b, d_i$)
5     $D = D \cup \langle d_i, y'_i \rangle$
6 end
7 $l$ = K − Lasso($D, k$)
8 $e$ = get_coefficients($l$)
9 return $e$
```

- First, LIME randomly generates instances surrounding the instance of interest (*cf.* Line 3).
- Second, LIME uses the black-box model to generate predictions of the generated random instances (*cf.* Line 4).
- Third, LIME constructs a local regression model using the generated random instances and their generated predictions from the black-box model (*cf.* Line 7).
- Finally, the coefficients of the regression model indicate the contribution of each metric on the prediction of the instance of interest according to the black-box model (*cf.* Line 8).

Figure 5.2 shows a visual explanation of LIME (the middle column). The blue bars indicate the supporting (+) scores of metrics towards the prediction as defective, while the red bars indicate the contradicting (-) scores of metrics towards the prediction as defective. In this example, the NotifyBuilder.java of the release 2.10.0 of the Apache Camel project is predicted (74%) as defective due to a supporting score of 0.75 for a condition of {#ClassCoupled > 5}, a supporting score of 0.6 for a condition of {#LineComment > 24}, and a supporting score of 0.5 for a condition of {#DeclareMethodPublic > 5}. On the other hand, the remaining probability of 26% of not defective could be explained by a contradict score of 0.77 for a condition of {#MajorDeveloper ≤ 2}.

LIME-HPO: Optimising the hyperparameter settings of LIME

Since LIME involves parameter settings (e.g., the number of randomly generated instances that LIME uses to construct local regression models), we propose to optimise the parameter settings of the LIME algorithm using a Hyper Parameter Optimisation (*LIME-HPO*). We use a differential evolution algorithm [202] to find an optimal value of the number of randomly generated instances where the objective function is to maximise the goodness-of-fit (R^2) of the local regression models of LIME. We use the implementation of the differential evolution technique as provided by the `DEoptim` function of the `DEoptim` R package [152] using the following parameter settings:

- The lower boundary of 100 and the upper boundary of 10000 for the population of the number of randomly generated instances used by LIME.
- The number of population (*NP*) of 10.
- The crossover probability (*cr*) of 0.5.
- The differential weighting factor from interval (f) of 0.8.
- The number of procedure iterations of 10 for generating population.

Given a black-box model and an instance to explain, LIME-HPO performs 6 key steps to generate an instance explanation as follows:

- First, LIME-HPO randomly generates a set of candidate of size *NP* within the population boundary for the number of randomly generated instances surrounding the instance of interest.
- Second, for each ca ndidate, LIME-HPO uses the black-box model to generate predictions of the generated random instances and constructs local regression models (Steps 2 and 3 of LIME).
- Third, LIME-HPO find the best candidate of this generation for the number of randomly generated instances that produces the local regression model with the highest goodness-of-fit.
- Fouth, LIME-HPO generates a set of candidate for the next generation using the set of candidate of the current generation with the crossover probability (*cr*) and the differential weighting factor from interval (f).
- Fifth, LIME-HPO reiterates the procedure until reaching the number of procedure iterations.
- Finally, LIME-HPO derives the coefficients of the local regression model that yields the highest goodness-of-fit across all generations as the contribution of each metric on the prediction of the instance of interest according to the black-box model.

Since LIME involves random perturbation (Line 3 in Algorithm 1), different samplings may produce different instance explanations. To mitigate the randomisation of LIME when

Algorithm 3: BreakDown's algorithm [201]

```
Input  : $b$ is a black-box model,
         $x$ is an instance to explain, and
         $D_{train}$ is a set of training instances used
         to construct the black-box model.
Output : $c$ is a set of contributions of metrics on the prediction of the instance $x$.
$M$ = independent variables in $x$
$M_{initial} = \varnothing$
$Y'_{train}$ = predict($b, D_{train}$)
$EY_{initial}$ = average($Y'_{train}$)
$D_{initial} = D_{train}$
for $i$ in $\{1, ..., \text{size}(M)\}$ do
    for $j$ in $\{M - M_{initial}\}$ do
        $D_{substituted} = D_{initial}$
        $D_{substituted}[, j] = x[j]$
        $Y'_{substituted,j}$ = predict($b, D_{substituted}$)
        $dy_j$ = abs(average($Y'_{substituted,j}$) $- EY_{initial}$)
    end
    $dy_{m_{max}}$ = find_max($dy_{\{M - M_{initial}\}}$)
    $c_{m_{max}} = dy_{m_{max}}$
    $EY_{initial}$ = average($Y'_{substituted,m_{max}}$)
    $D_{initial}[, m_{max}] = x[m_{max}]$
    $M_{initial} = M_{initial} \cup m_{max}$
end
return $c$
```

re-generating instance explanations, Ribeiro[1] suggests to set a random seed prior to applying LIME. Thus, our LIME-HPO follows this suggestion by setting a random seed.

BreakDown: Explaining a global model

BreakDown [55, 201] is a model-agnostic technique that uses the greedy strategy to sequentially measure contributions of metrics towards the expected prediction. Given a black-box model b, an instance to explain x, and training data used to construct the model D_{train}, BreakDown performs 5 key steps to generate an instance explanation as follows:

- First, BreakDown generates the predictions of all instances in the training data and computes the average estimation of such predictions (*cf.* Lines 3-4). In the first iteration, BreakDown uses the original training data as the syntactic training data for calculation (*cf.* Line 5).

[1] https://github.com/marcotcr/lime/issues/119#issuecomment-344743006

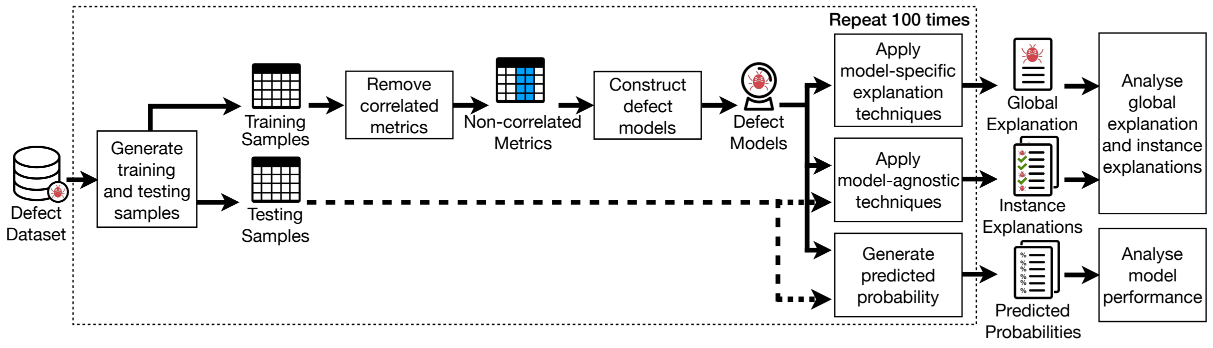

Repeat 100 times
Defect Dataset
Generate training and testing samples
Training Samples
Testing Samples
Remove correlated metrics
Non-correlated Metrics
Construct defect models
Defect Models
Apply model-specific explanation techniques
Apply model-agnostic techniques
Generate predicted probability
Global Explanation
Instance Explanations
Predicted Probabilities
Analyse global explanation and instance explanations
Analyse model performance

Figure 5.3 An overview diagram of the design of our study.

- Second, BreakDown sequentially substitutes the values of each metric in the syntactic training data with the value of such metric appeared at the instance of interest (*cf.* Lines 7-9).

- Third, BreakDown generates the predictions of the substituted training data, and identify the metric that produces the greatest absolute difference between the expected predictions of the syntactic training data and the substituted training data (*cf.* Lines 10-13).

- Fourth, BreakDown allocates such greatest difference in expected predictions made by the metric as its contribution (*cf.* Line 14).

- Finally, BreakDown considers the set of expected predictions of the substituted training data with the greatest difference in expected predictions as the new set of expected predictions (*cf.* Lines 15-16) and re-iterates to calculate the contributions of the metrics in which their contributions are not allocated (*cf.* Line 17).

Figure 5.2 shows a visual explanation of BreakDown (the right column). The light blue bars indicate the supporting (+) probability of metrics towards the prediction as defective, while the light brown bars indicate the contradicting (-) probability of metrics towards the prediction as defective. In this example, the NotifyBuilder.java of the release 2.10.0 of the Apache Camel project is predicted (74%) as defective due to a supporting probability of 0.11 for #MajorDeveloper, a supporting probability of 0.09 for #LineComment, a supporting probability of 0.9 for #ClassCoupled, and a supporting probability of 0.07 for AverageCyclomatic.

5.4 Experimental Design

In this section, we discuss our criteria for selecting the studied datasets; and the design of the case study that we perform to address our six research questions. Figure 5.3 provides an overview of the design of our case study.

5.4.1 Studied Datasets

Recently, Yatish *et al.* [233] showed that (1) some issue reports that are addressed within 6 months after a release do not realistically affect a studied release (false positive), while (2) some issue reports that realistically affect the studied release are addressed later than 6 months after the release (false negative). Thus, the approximation of post-release window periods (e.g., 6 months) that were popularly-used in many defect datasets may introduce biases to the construct to the validity of our results.

Thus, we select a corpus of publicly-available defect datasets provided by Yatish *et al.* [233] where the ground-truths were labelled based on the affected releases. The datasets consist of 32 releases that span 9 open-source software systems. Each dataset has 65 software metrics along 3 dimensions, i.e., 54 code metrics, 5 process metrics, and 6 human metrics. Table 5.1 shows a statistical summary of the studied datasets.

Code metrics describe the relationship between properties extracted from source code and software quality. These code metrics are extracted using the `Understand` tool from SciTools along 3 dimensions, i.e., complexity (e.g., McCabe Cyclomatic), volume (e.g., lines of code), and object-oriented (e.g., a coupling between object classes). Among these code metrics, some properties are extracted at the method level, and thus three aggregation schemes (i.e., minimum, average, and maximum) are used to summarise these metrics to the file level.

Process metrics describe the relationship between development process and software quality. For each instance, there are (1) the number of commits, (2) the number of added lines of code, (3) the number of deleted lines of code, (4) the number of active developers, and (5) the number of distinct developers. Similar to Rahman *et al.* [174], the number of added and deleted lines of code of each instance is normalized by the total number of added and deleted lines, respectively.

Human factors describe the relationship between the ownership of instances and software quality [17, 173, 216]. Ownership metrics are based on the traditional code ownership heuristics of Bird *et al.* [17], for each instance, the ownership of each developer is measured using the proportion of the code changes made by the developer on the total code changes. There are two granularities of code changes, i.e., lines of code level (LINE), and commit level (COMMIT), while there are two levels of ownership of an instance for developers, as recommended by Bird *et al.* [17]. That is, developers with low code ownership (i.e., less than 5% code contribution on an instance) are considered as minor authors. On the other hand, developers with high code ownership (i.e., more than 5% code contribution on an instance) are considered as major authors. Ownership metrics consist of (1) the number of the owner (i.e., the developer with the highest code contribution on an instance), (2) the number of the minor authors, and (3) the number of major authors with respect to the two granularities of code changes.

Table 5.1 A statistical summary of the studied systems.

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

5.4.2 Generate Training and Testing Samples

To generate training and testing samples, we opt to use an out-of-sample bootstrap validation technique [39, 47, 68, 205, 212], which leverages aspects of statistical inference. We use the out-of-sample bootstrap validation technique (1) to ensure that the generated training samples are representative to the original dataset and (2) to ensure that the produced estimates are the least bias and most reliable [212]. We first generate a bootstrap sample of sizes N with replacement from the studied defect datasets. The generated sample is also of size N. We construct defect models using the bootstrap samples, while we interpret the samples that do not appear in the generated bootstrap samples at the instance-level. On average, 36.8% of the original dataset will not appear in the bootstrap samples, since the samples are drawn with replacement [39]. We repeat the out-of-sample bootstrap process for 100 times and report their average calculations.

5.4.3 Remove Correlated Metrics

Prior studies raise concerns that collinearity (i.e., correlated metrics) often impacts the global explanation of defect models [85, 87, 207, 211]. For example, a defect model that is constructed with correlated metrics could produce different global explanations when reordering the model formula of regression models. Recently, the bagging technique for random forest is proposed to mitigate collinearity (i.e., different trees are constructed with different subset of metrics), prior studies found that some trees are still constructed with correlated metrics [85, 203]. Since LIME builds a local regression model which is known to be sensitive to collinearity [85, 207], it is necessary to remove correlated metrics to mitigate such impact prior to constructing defect models.

Prior studies introduce many techniques to remove irrelevant metrics and correlated metrics (e.g., Correlation-based Feature Selection (CFS), Information Gain (IG), and stepwise regression) [5, 26, 34, 40, 97, 140, 160, 198]. My prior work [89] found that such feature selection techniques cannot mitigate correlated metrics (e.g., CFS produces a subset of metrics that are highly correlated with the dependent variable while having the least correlation among themselves. Yet, some of the independent variables are still highly correlated), suggesting that correlation analyses must be applied. However, such correlation analyses often involve manual selection by a domain expert. To ensure the scalability of our framework, we apply the AutoSpearman technique on the training samples. AutoSpearman automatically selects one metric of a group of the highest correlated metrics that shares the least correlation with other metrics that are not in that group [89]. We use the implementation of the AutoSpearman technique as provided by the `AutoSpearman` function of the `Rnalytica` R package [87]. We observe that AutoSpearman mitigates correlated metrics and selects only 22-27 of 65 metrics. In other words, as high as 38-43 metrics share strong correlations among themselves, which are then removed by AutoSpearman.

5.4.4 Construct Defect Models

Shihab [193] and Hall *et al.* [65] show that logistic regression and random forests are the two most-popularly-used classification techniques in the literature of software defect prediction, since they are explainable and have built-in model explanation techniques (i.e., the ANOVA analysis for the regression technique and the variable importance analysis for the random forests technique). Recent studies [49, 210, 213] also demonstrate that automated parameter optimisation can improve the performance and stability of defect models. Using the findings of prior studies to guide our selection, we choose (1) the commonly-used classification techniques that have built-in model-specific explanation techniques (i.e., logistic regression and random forests) and (2) the top-5 classification techniques when performing automated parameter optimisation [210, 213] (i.e., random forests, C5.0, AVNNet, GBM, and xGBTree).

We use the implementation of Logistic Regression as provided by the `glm` function of the `stats` R package [214]. We use the implementation of automated parameter optimisation of Random Forests, C5.0, AVNNet, GBM, and xGBTree as provided by the `train` function of the `caret` R package [111] with the options of `rf`, `C5.0`, `avNNet`, `gbm`, and `xgbTree` for the method parameter, respectively. We neither re-balance nor normalize our training samples to preserve its original characteristics and to avoid any concept drift for the explanations of defect models [208].

5.4.5 Apply Model-specific Explanation Techniques

We apply model-specific explanation techniques to generate global explanations—what factors are associated with software quality. We use the ANOVA Type-II analysis for logistic regression and the scaled Permutation Importance analysis for random forests, as suggested by our recent work [85]. We use the *usage* (i.e., the percentage of training instances that satisfy all of the terminal nodes after the split which are associated with the metric) of metrics to the generate global explanation of C5.0 [172]. We use the combinations of the absolute values of the weights derived across hidden layers in neural networks to generate the global explanation of AVNNet [51]. We use the relative influence of metrics derived from boosted trees to generate the global explanation of GBM and xGBTree [48, 158].

We use the implementation of the ANOVA Type-II analysis as provided by the `Anova` function of the `car` R package [45]. We use the implementation of the scaled Permutation Importance analysis as provided by the `importance` function of the `randomForest` R package [23]. We use the implementation provided by the `varImp` function of the `caret` R package [111] to generate global explanation of C5.0, GBM, and xGBTree.

5.4.6 Apply Model-agnostic Techniques

We apply model-agnostic techniques to generate instance explanations of the predictions of defect models. We use three model-agnostic techniques, i.e., two state-of-the-art LIME (Local Interpretable Model-Agnostic Explanations) and BreakDown, and our improvement of LIME with Hyper Parameter Optimisation based on the differential evolution technique (LIME-HPO) The technical descriptions are presented in Section 5.3 We use the implementation of LIME as provided by the `lime` and `explain` functions of the `lime` R package [164]. We use the implementation of the differential evolution technique as provided by the `DEoptim` function of the `DEoptim` R package [152]. We use the implementation of BreakDown as provided by the `broken` function of the `breakDown` R package [15].

5.4.7 Generate Predicted Probability

We use defect models to generate predicted probabilities (i.e., defect-proneness) of testing instances. The predicted probabilities range from 0 (not defective) to 1 (likely to be defective). We use the classification threshold of 0.5 to map predicted probabilities to a binary decision of defect and clean. Predicted probabilities of above 0.5 indicate defect, otherwise clean.

5.4.8 Analyse Global Explanation and Instance Explanations

We analyse global explanation and instance explanations to address RQs 1, 2, 3, 4, and 5. We also conduct a survey study of 20 practitioners to evaluation instance explanations generated by the studied model-agnostic techniques in RQ6. The motivation, approach, and results for each RQ are explained in detail in Section 5.5.

5.4.9 Analyse Model Performance

To ensure that the generated global explanation and instance explanations are derived from accurate defect models, we evaluate the model performance of the studied classification techniques.

First, we use the Area Under the receiver operator characteristic Curve (AUC) to measure the discriminatory power of our models, as suggested by recent research [52, 114, 174]. The axes of the curve of the AUC measure are the coverage of non-defective modules (true negative rate) for the x-axis and the coverage of defective modules (true positive rate) for the

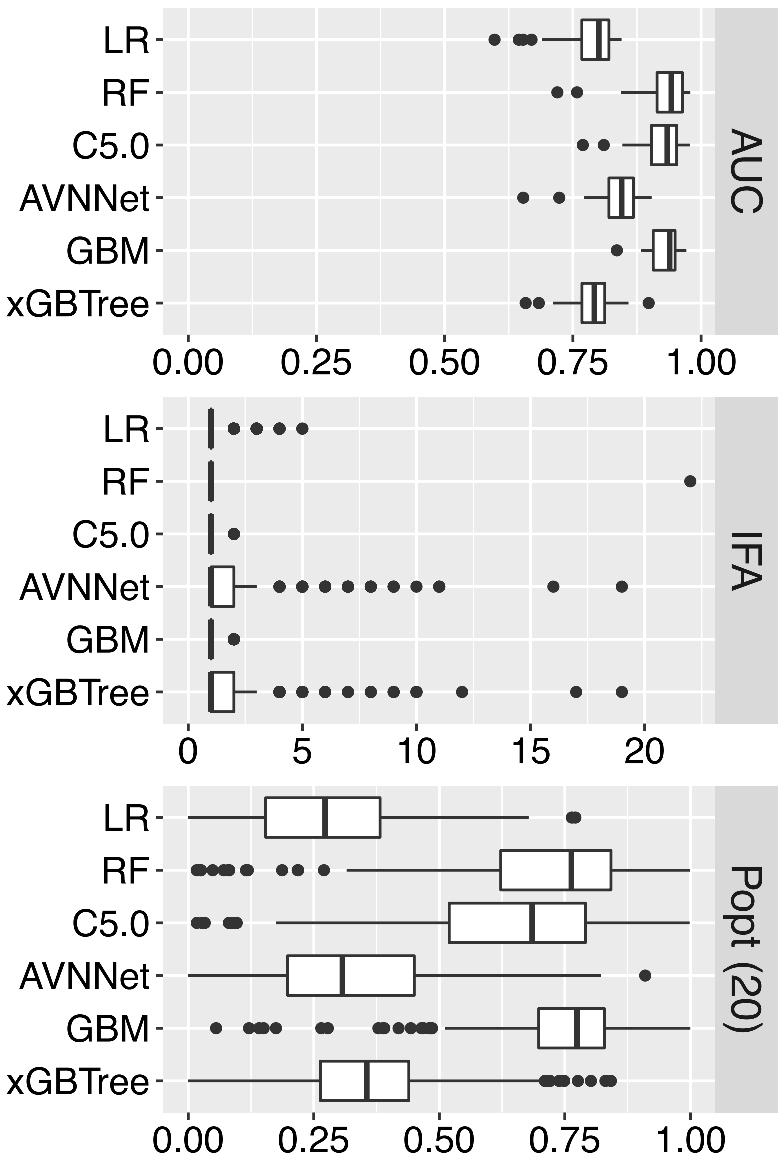


Figure 5.4 The distributions of model performance for all studied defect datasets of each classification technique.

y-axis. The AUC measure is a threshold-independent performance measure that evaluates the ability of models in discriminating between defective and clean instances. The values of AUC range between 0 (worst), 0.5 (no better than random guessing), and 1 (best) [66].

Second, we use the Initial False Alarm (IFA) measure to identify the number of false alarms encountered before the first defective module [79]. To calculate the IFA measure, we sort the modules in descending order of their risk predicted by a model. Then, the IFA measure is computed as k, where k is the number of non-defective modules that are predicted as defective by a model before the first defective module. The values of IFA range from 1 (best) to n, where n is the number of all modules.

Third, we use the P_{opt} measure [96, 135, 232] to measure the effort-aware predictive performance of defect models. The P_{opt} measure is defined by the area Δ_{opt} between the effort-based cumulative lift charts of the optimal model and a defect model. The axes of the effort-based cumulative lift charts are the proportion of effort for the x-axis and the coverage of defective modules for the y-axis. For the optimal model, all modules are sorted in descending order of the actual defect density (i.e., the proportion of the number of defects and the lines of code of each module). On the other hand, for a defect model, all modules are sorted in decreasing order of the predicted probabilities of each module. The P_{opt} measure is computed as $P_{opt} = 1 - \Delta_{opt}$. The values of P_{opt} range between 0 (worst), 0.5 (no better than random guessing), and 1 (best).

Preliminary Analysis. Figure 5.4 shows the distributions of the model performance of all of the studied classification techniques for all of the studied defect datasets. **Our studied classification techniques achieve a median AUC of 0.79-0.94, a median IFA of 1, and a median Popt20 of 0.27-0.77,** indicating that our studied classification techniques are highly accurate.

5.5 Experimental Results

We present the results of our study with respect to our six research questions.

(RQ1) Does LIME with Hyper Parameter Optimisation (LIME-HPO) outperform default LIME in terms of the goodness-of-fit of the local regression models?

Motivation. Since LIME generates instance explanations from local regression models that are constructed using the randomly generated instances around the neighbours of the instance to be explained, we use the goodness-of-fit (R^2) of the local regression models as a proxy for measuring the performance of LIME when generating instance explanations. Prior studies [49, 210, 213] have shown that hyper parameter optimisation can be used to improve the performance of defect models. Yet, little is known about whether hyper parameter optimisation can improve the goodness-of-fit of the LIME algorithm when generating explanations for the predictions of defect models.

Approach. To address RQ1, we analyse the goodness-of-fit (R^2) of LIME and LIME-HPO when generating explanations for the predictions of defect models for all of the studied datasets. For each bootstrap sample of each defect dataset, we use the overview diagram (see Figure 5.3) to generate instance explanations of the testing instances. Then, we compute the goodness-of-fit of the local regression models that are used to generate these instance explanations for LIME and LIME-HPO. We apply Wilcoxon signed-rank test [229] to identify whether distributions of the goodness-of-fit of the local regression models produced by LIME and LIME-HPO are statistically different. We also apply Cliff's $|\delta|$ effect size test to measure the magnitude of the differences. Finally, we report the results using boxplots in Figure 5.5.

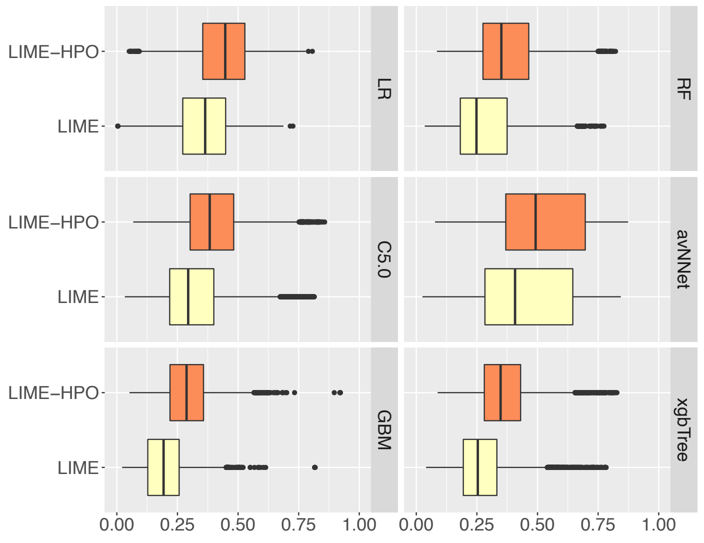


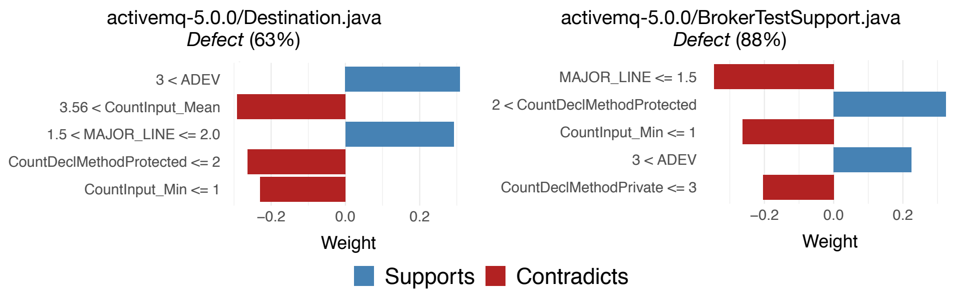
Figure 5.5 The distributions of the goodness-of-fit (R^2) of the local regression models constructed with LIME and LIME-HPO for all of the studied defect datasets and the studied classification techniques.

Results. **LIME-HPO always outperform default LIME in terms of the goodness-of-fit of the local regression models with an average improvement of 8%.** Figure 5.5 shows the distributions of the goodness-of-fit of the local regression models constructed with LIME and LIME-HPO for all studied defect datasets. After performing hyper parameter optimisation (LIME-HPO), we find that LIME-HPO improves the goodness-of-fit of the local regression models by (at the median) 6.6% for LR, 7.4% for RF, 6.7% for C5.0, 6.1% for AVNNet, 8.0% for GBM, and 7.6% for xGBTree. We observe that the average R^2 improvement among six classification techniques is 8%. Moreover, the results of Wilcoxon signed-rank test confirm that the improvement in the goodness-of-fit of the local regression models of LIME-HPO over LIME are statistically significant for all of the studied classification techniques. The Cliff's $|\delta|$ effect size test also shows that the effect size of such differences are large for GBM; medium for LR, RF, C5.0, and xGBTree; and small for AVNNet.

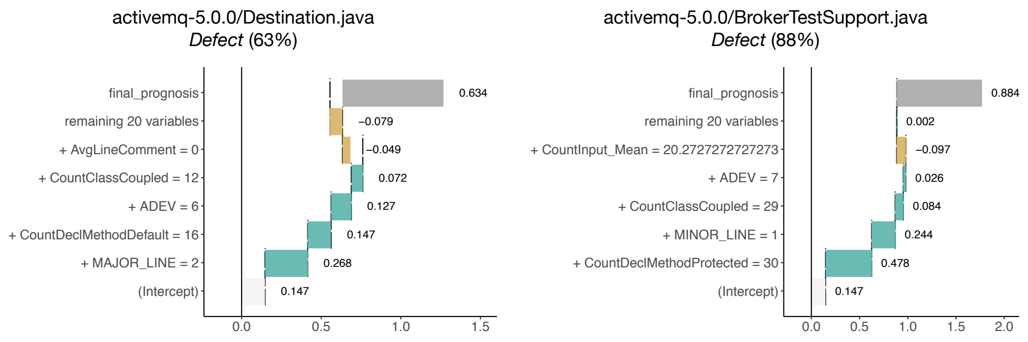
(RQ2) Can model-agnostic techniques explain the predictions of defect models?

Motivation. Traditionally, model-specific explanation techniques (e.g., ANOVA) are used to generate global explanations. However, such global explanations cannot justify each individual prediction of defect models—i.e., why a software module is likely to be defective in the future. Recent research introduces model-agnostic techniques for explaining the predictions of any black-box models [55, 182]. Yet, these model-agnostic techniques have not been investigated in the context of software engineering (particularly for defect prediction).
Approach. To address RQ2, we investigate the variation of instance explanations generated by model-agnostic techniques for explaining the predictions of defect models. In other words,

given a model trained from the same training data, do different predictions (i.e., test data) have different explanations. For each bootstrap sample of each defect dataset, we use the overview diagram (see Figure 5.3) to generate instance explanations of the testing instances. Instance explanations are the importance scores of each metric that contribute to the final probability of each prediction. For each instance explanation, we transform the scores of metrics into the ranking of metrics (e.g., from [ADEV = 0.8, MINOR_DEV = 0.15, CC = 0.05] to [1st = ADEV, 2nd = MINOR_DEV, 3rd = CC]). We then compute the rank differences of each metric among instance explanations of the correctly predicted defective instances. For example, given two instance explanations, the ranking of one instance explanation is [1st = ADEV, 2nd = MINOR_DEV, 3rd = CC], while that of another explanation is [1st = CC, 2nd = MINOR_DEV, 3rd = ADEV]. In this example, ADEV appears at the 1st rank in one instance explanation, while appearing at the 3rd rank in another instance explanation. Thus, the rank difference of ADEV is |1 – 3| = 2. We apply this calculation for all of the studied metrics for all instance explanations and report the results using box plots.



(a) An example instance explanation of LIME. The blue bars indicate supporting (positive) scores towards a file being predicted as defective, while the red bars indicate contradict (negative) scores towards its prediction.



(b) An example instance explanation of BreakDown. The x-axis presents the contribution (probability score) of each metric in the y-axis.

Figure 5.6 An illustrative example of instance explanations generated by LIME and Break-Down, respectively.

Results. Model-agnostic techniques can explain the predictions of defect models. To illustrate the visual explanation of our studied model-agnostic techniques, we select the Apache ActiveMQ 5.0.0 dataset and logistic regression (LR) as the subject of this illustrative example. Figures 5.6a and 5.6b presents an illustrative example of instance explanations of two testing instances that are correctly predicted as defective (i.e., Destination.java and BrokerTestSupport.java) produced by LIME and BreakDown, respectively. In this example, Destination.java and BrokerTestSupport.java are predicted as defective files with the predicted probability of 63% and 88%, respectively.

LIME. Figure 5.6a (left) shows the visual explanations for explaining the predictions of Destination.java that is generated by LIME. The blue bars indicate the supporting (+) scores of metrics towards the prediction as defective, while the red bars indicate the contradicting (-) scores of metrics towards the prediction as defective. Figure 5.6a (left) shows that Destination.java is predicted (63%) as defective due to a supporting score of 0.3 for a condition of $\{ADEV > 3\}$ and a supporting score of 0.29 for a condition of $\{1.5 < MAJOR\_LINE \leq 2\}$. On the other hand, the remaining probability of 37% of not defective could be explained by a contradict score of 0.29 for a condition of $\{CountInput\_Mean > 3.56\}$, a contradict score of 0.27 for a condition of $\{CountDeclMethodProtected \leq 2\}$, and a contradict score of 0.22 for a condition of $\{CountInput\_Min \leq 1\}$. These instance explanations indicate the most important conditions that support and contradict a file being predicted as defective.

BreakDown. Figure 5.6b (left) shows the visual explanations for explaining the predictions of Destination.java that is generated by BreakDown. The light blue bars indicate the supporting (+) probability of metrics towards the prediction as defective, while the light brown bars indicate the contradicting (-) probability of metrics towards the prediction as defective. Figure 5.6b (left) shows that Destination.java is predicted (63%) as defective due to a supporting probability of 0.27 for MAJOR_LINE, a supporting probability of 0.15 for CountDeclMethodDefault, a supporting probability of 0.13 for ADEV, and a supporting probability of 0.07 for CountClassCoupled. In contrast, an CountClassCoupled value of 12 contradicts the prediction by a probability of 0.8, and an AvgLineComment value of 0 contradicts the prediction by a probability of 0.05.

Given the same defect models, different predictions have different instance explanations. Figure 5.7 shows the distributions of the rank difference of the metric among instance explanations for all of the studied defect datasets. We find that the rank differences of metrics among instance explanations are, at the median, 20 for LIME, 22 for LIME-HPO, and 21 for BreakDown. In other words, the most important metric of an instance may appear as the rank 21th-23th of another instance. We observe similar results for all studied classification techniques (the online appendix [83] provides the results of each classification technique). As shown in Figures 5.6a and 5.6b, we observe that while these two instances are predicted as defective by defect models, their instance explanations are different. According to this example, on the above sub-figure (LIME), the number of active developers (ADEV), which appears as the most important metric of the instance explanation of Destination.java, appears as the 4th important metric of the instance explanation of BrokerTestSupport.java. Similarly, on the below sub-figure (BreakDown), the lines of code contributed by major developers (MAJOR_LINE), which appears as the most important metric of the instance explanation of Destination.java, appears as the 2nd important metric of the instance ex-

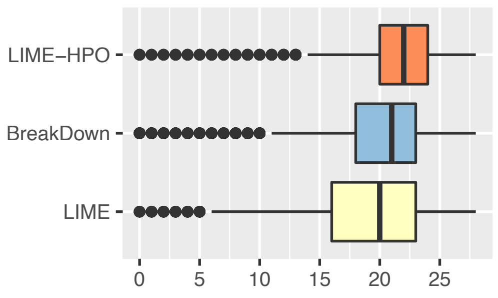


Figure 5.7 The distributions of rank differences of each metric across instances explanations for all studied defect datasets.

planation of BrokerTestSupport.java. The variation of instance explanations among the predictions of defect models indicates that one global explanation of defect models does not imply instance explanations (and vice versa), highlighting the need for model-agnostic techniques for explaining the predictions defect models.

(RQ3) Do instance explanations generated by model-agnostic techniques overlap with the global explanation of defect models?

Motivation. Recent research raised concerns about the trustworthiness of instance explanations as generated by model-agnostic techniques. For example, Ribeiro *et al.* [182] argue that instance explanations must correspond to how the trained model behaves. Guidotti *et al.* [58] argue that the local approximation models should mimic the black-box models when predicting an unseen dataset. The results of RQ2 suggest that instance explanations generated by the studied model-agnostic techniques have a great variation among each prediction of defect models. Yet, little is known about whether these instance explanations generated by the studied model-agnostic techniques are overlapping with the global explanation of defect models.

Approach. To address RQ3, we investigate whether instance explanations generated by model-agnostic techniques are overlapping with the global explanation of defect models. For each bootstrap sample, we use the overview diagram (see Figure 5.3) to generate a global explanation from training instances and generate instance explanations from testing instances. Since the out-of-sample bootstrap validation technique leverages aspects of statistical inference [39, 47, 68, 205, 212], both training and testing samples are approximately equivalent to the population (i.e., the original dataset). Thus, explanations derived from both training and testing instances should also be approximately equivalent. To generate a global explanation, we use the ANOVA Type-II analysis for logistic regression, the scaled Permutation Importance analysis for random forests, the usage of metrics for C5.0, the combinations of the absolute weights across hidden layers for AVNNet, and the relative influence of metrics derived from boosted trees for GBM and xGBTree. To generate instance explanations for testing instances, we use the LIME, LIME-HPO, and BreakDown techniques.

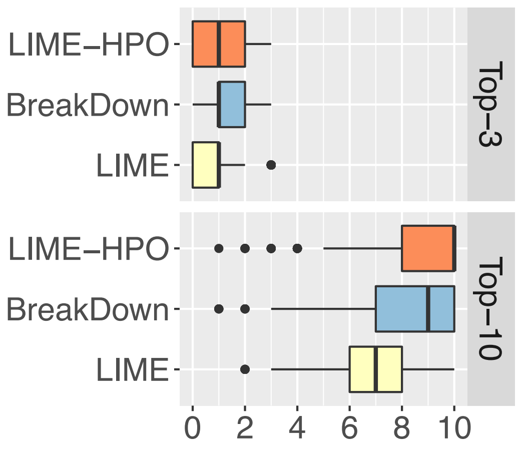


Figure 5.8 The distributions of the top-k overlapping metrics between the global explanation and instance explanations for all of the studied defect datasets and the studied classification techniques.

While a defect model generates only one global explanation from training instances, model-agnostic techniques generate many instance explanations from testing instances. Thus, we need to summarise instance explanations to the model level prior to comparing with a global explanation of each bootstrap sample. To summarise instance explanations, we apply the Scott-Knott Effect Size Difference test (ScottKnottESD) [206] to identify the ranking of metrics that is statistically distinct across ranks while being non-negligible different within ranks. Then, we identify the top-*k* overlapping metrics between global explanations and the summary of instance explanations. The top-k overlapping metrics are the number of the top-k metrics of a global explanation that consistently appear in the top-k metrics of the summary of instance explanations. For example, the global ranking of metrics is [1st = LOC, 2nd = CC, 3rd = ADEV], while the summarised ranking of metrics for instance explanations is [1st = ADEV, 2nd = MINOR_DEV, 3rd = CC]. In this example, the top-3 overlapping metrics are 2 out of 3 metrics. Finally, we apply Wilcoxon signed-rank test [229] to identify whether the distributions of the top-k overlapping metrics between the global explanation and instance explanation produced by LIME, LIME-HPO, and BreakDown are statistically different. We also apply Cliff's $|\delta|$ effect size test to measure the magnitude of the differences.

<u>Results</u>. **Despite the variation of the ranking of the top-10 important metrics for instance explanations in RQ2, their overall ranking is mostly overlapping with (but is not exactly the same) as that of the global explanations.** Figure 5.8 shows that, at the median, 7, 10, and 9 of the top-10 summarised important metrics for instance explanations are overlapping with the top-10 global important metrics for LIME, LIME-HPO, and BreakDown, respectively. The results of Wilcoxon signed-rank test show that the differences of the top-10 overlapping metrics are statistically significant among the studied model-agnostic techniques (for LIME and LIME-HPO, LIME-HPO and BreakDown, and LIME and BreakDown). The Cliff's $|\delta|$ effect size test also shows that the magnitude of such differences are large for LIME and LIME-HPO, medium for LIME and BreakDown, and

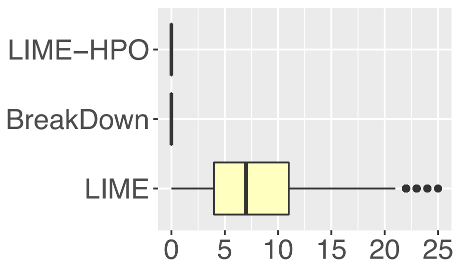


Figure 5.9 The distributions of rank differences of each metric when re-generating instance explanations for all of the studied defect datasets.

negligible for LIME-HPO and BreakDown, suggesting that LIME-HPO is comparable to BreakDown. On the other hand, at the median, at least one of the top-3 summarised important metrics for instance explanations is overlapping with the top-3 global important metrics. The detailed results of each studied classification techniques are available in the online appendix [83].

(RQ4) Do model-agnostic techniques generate the same instance explanation when they are re-generated for the same instance?

Motivation. Recent research raised concerns about the reliability of instance explanations generated by model-agnostic techniques. For example, Lundberg *et al.* [125] argue that instance explanations must remain the same when they are re-generated for the same instance. Assuming that one wants to generate an explanation for a file predicted as defective, model-agnostic techniques (that involve random perturbation like LIME) might generate different synthetic instances, leading to different explanations for the same instance. Thus, little is known about whether model-agnostic techniques generate the same instance explanation when they are re-generated for the same instance and the same defect model.

Approach. To address RQ4, we analyse the reliability of the instance explanations generated by LIME, LIME-HPO, and BreakDown. Ideally, re-generating instance explanations of the same instance from the same model using the same model-agnostic technique should produces the same explanation. Therefore, we use the variation in instance explanations when re-generating with the same setting as a proxy for measuring the reliability of model-agnostic techniques. Rather than generating instance explanations of all testing instances, we randomly select only one testing instance as the instance of interest. Similar to RQs 1, 2, and 3, we use the overview diagram (see Figure 5.3) to generate instance explanations of this instance of interest. We re-generate the instance explanation of the selected instance for 100 repetitions. We compute the rank differences of each metric when re-generating instance explanations and report the results of all studied defect datasets using boxplots.

Results. **Regardless of the studied classification techniques, LIME-HPO and BreakDown consistently generate the same instance explanation for the same instance. On the other hand, LIME generates different instance explanations when re-generating**

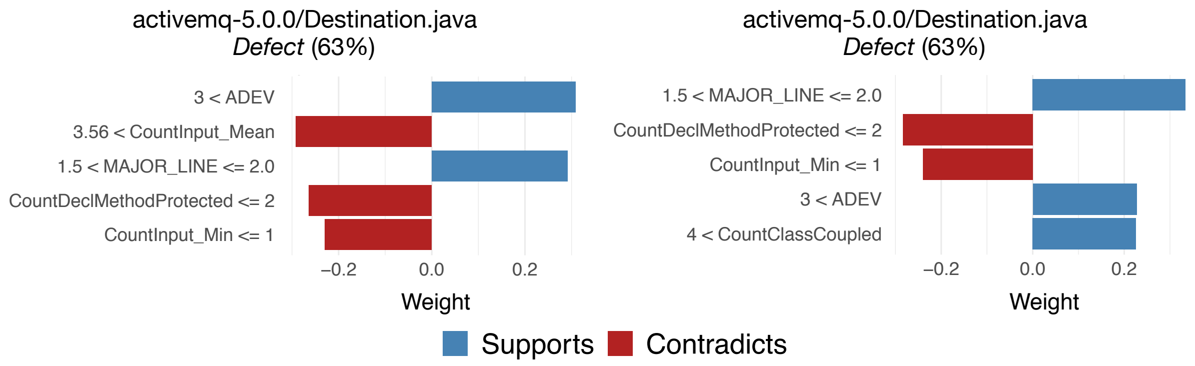


Figure 5.10 An illustrative example of instances explanations of a defective testing instance when regenerating instance explanations with LIME.

instance explanations of the same instance. Figure 5.9 shows the distributions of rank differences of each metric when re-generating instance explanations for all studied defect datasets. Ideally, instance explanations of an instance should be the same when re-generating using the same model and the same model-agnostic technique. Regardless of the studied classification techniques, we find that LIME-HPO and BreakDown consistently produce the same instance explanation for the same instance. On the other hand, we find that LIME produces inconsistent instance explanations across repetitions with the median rank differences of 7. We report the detailed results of rank differences for each studied classification technique in the online appendix [83].

To further illustrate the variation of instance explanations generated by LIME across repetitions, similar to RQ2, we select the Apache ActiveMQ 5.0.0 dataset as the subject of this illustrative example. Figure 5.10 shows an illustrative example of instance explanations of a defective testing instance when re-generating instance explanations with LIME. We observe that the model consistently identifies the instance as defective with the same predicted probability of 0.63 for both repetitions. However, while these instance explanations of the same instances are generated from the same model using the same model-agnostic technique (LIME), such explanations—the top-5 important metrics—vary greatly.

(RQ5) What is the computational time of model-agnostic techniques for explaining the predictions of defect models?

Motivation. The computational time is one of the most important criteria when deploying software analytics in practice. Model-agnostic techniques may introduce significant overhead to the current practices of defect modelling workflow. Yet, little is known about whether the computational time of model-agnostic techniques is practical to be deployed by practitioners.
Approach. To address RQ5, we analyse the computational time of model-agnostic techniques for explaining the predictions of defect models. To do so, similar to RQs 1, 2, 3, and 4, we use the overview diagram (see Figure 5.3) to construct defect models and generate instance explanations. For each studied defect dataset, we generate one set of bootstrap

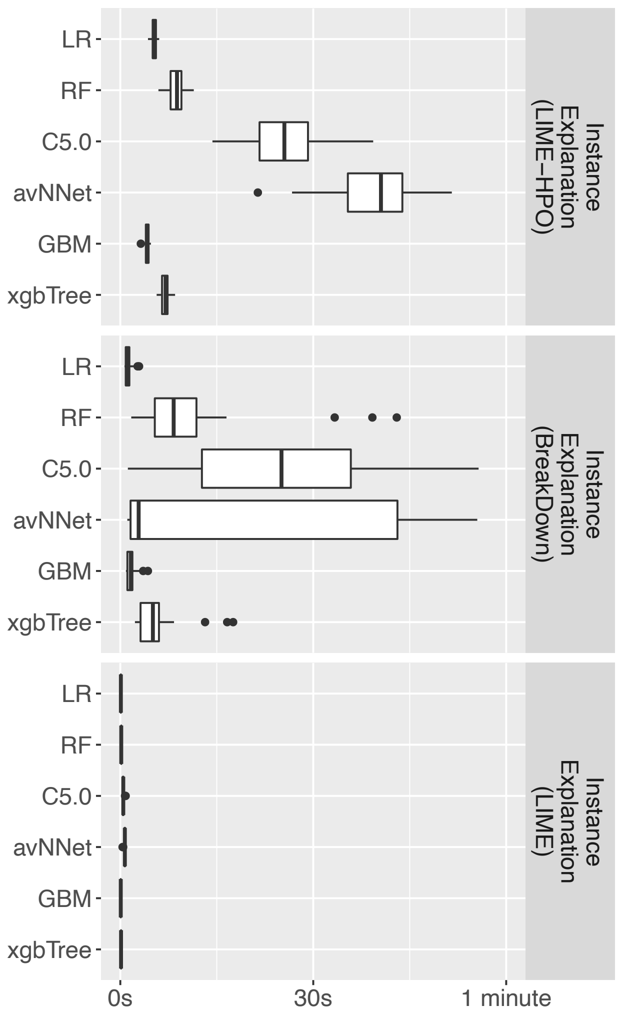
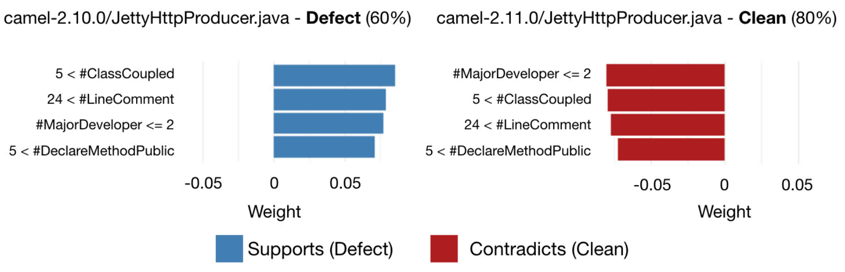


Figure 5.11 The distributions of computational time of model-agnostic techniques for explaining the predictions of defect models for all of the studied defect datasets.

training and testing instances. We construct a defect model using bootstrap training instances. Then, we randomly select one testing instance and generate an instance explanation using model-agnostic techniques to measure their computation time. The computational time is based on a standard computing machine with an Intel Core i7-8700K processor and 32GB of RAM. Finally, we report the results using box plots.

Results. **The computational time of LIME-HPO, BreakDown, and LIME is less than a minute to generate instance explanations for all of the studied classification techniques.** Figure 5.11 shows the distributions of computational time of model-agnostic techniques for explaining the predictions of defect models for all of the studied defect datasets. We find that, regardless of the studied classification techniques, the computational time that the studied model-agnostic techniques take to generate an instance explanation is at most one minute for all of the studied defect datasets. This finding suggests that all of the studied model-agnostic techniques is practical to be used in real-world deployments.



T-contrast Explanation: *JettyHttpProducer.java* is predicted as defective in release 2.10.0, but is predicted as clean in release 2.11.0.
The reasons are (1) 5 < *# of class coupled,* (2) 24 < *lines of comment,* (3) *# of major developers <= 2,* and (4) 5 < *# of public methods*.

Figure 5.12 An example of the Time-contrast explanations generated by model-agnostic techniques for explaining the predictions of defect models.

(RQ6) How do practitioners perceive the contrastive explanations generated by model-agnostic techniques?

Motivation. Referring to a theory of explanations described in Section 5.2, Miller [146] and Lipton *et al.* [120] argue that explanations can be presented as answers to why-questions and humans tend to be cognitively attached to digest contrastive explanations. *Contrastive explanations* focus on only the differences on properties within an object (Property-contrast), between two Objects (Object-contrast), and within an object over Time (Time-contrast) [221]. Thus, contrastive explanations may be more valuable and more intuitive to humans. Yet, little is known about whether contrastive explanations generated by model-agnostic techniques can answer why-questions.

Approach. To address RQ6, we conducted a survey study of 20 software practitioners to investigate their perceptions of instance explanations generated by model-agnostic techniques. As suggested by Kitchenham and Pfleeger [104], we performed the following steps when conducting this survey study:

(Step-1) Setting the objectives: The survey aimed to investigate whether instance explanations generated by model-agnostic techniques (1) can be used to answer the why-questions (i.e., Property-contrast, Objective-contrast, and Time-contrast); (2) build appropriate trusts of the predictions of defect models; and (3) are necessary and useful.

(Step-2) Survey design: We first introduced a concept of explainable defect models with respect to the literature of eXplainable Artificial Intelligence (XAI). Then, we used the Releases 2.10.0 and 2.11.0 of the Apache Camel project as the subject of the study to generate instance explanations. We presented 3 types of explanations for investigation, i.e., **Property-contrast explanation** (e.g., why was file *A* predicted as defective rather than clean?), **Object-contrast explanation** (e.g., why was file *A* predicted as defective, while file *B* was predicted as clean?), and **Time-contrast explanation** (e.g., why was file *A* predicted as defective in version 1.2, but was subsequently predicted as clean in version 1.3?). Figure 5.12 illustrates an example of the Time-contrast explanations generated by model-agnostic techniques, while other examples of the Property-contrast and Object-contrast explanations are provided in the online appendix [83].

The survey design is a cross-sectional study where participants provide their responses at one fixed point in time. The survey consists of demographic and three sets of questions with respect to the 3 objectives of the study. There are 11 closed-ended questions and 20 open-ended questions. The survey takes approximately 15 minutes to complete and is anonymous.

(Step-3) Developing a survey instrument: For closed-ended questions, we used agreement and evaluation ordinal scales. To mitigate the inconsistency of the interpretation of numeric ordinal scales, we labelled each level of the ordinal scales with words as suggested by Krosnick [110], i.e., strongly disagree, disagree, neutral, agree, and strongly agree. The format of our survey instrument was an online questionnaire. We used an online questionnaire service provided by Google Forms.[2] When accessing the survey, each participant was provided with an explanatory statement that describes the purpose of the study, why the participant is chosen for this study, possible benefits and risks, and confidentiality.

(Step-4) Evaluating the survey instrument: We carefully evaluated the survey (i.e., pre-testing [122]) prior to recruiting participants. We evaluated it with focus groups (i.e., practitioners) to assess the reliability and validity of the survey. We re-ran it to identify and fix potential problems (e.g., missing, unnecessary, or ambiguous questions) until reaching a consensus among the focus groups. Finally, the survey has been rigorously reviewed and approved by Monash University Human Research Ethics Committee (MUHREC Project ID: 22542). We also provide the ethics approval certificate in our online appendix [83].

(Step-5) Obtaining valid data: The target population of our study is software practitioners. To reach the target population, we used a recruiting service provided by the Amazon Mechanical Turk to recruit 20 participants as a representative subset of the target population. We also applied participant filter options of "Employment Industry - Software & IT Services" and "Job Function - Information Technology" to ensure that the recruited participants are valid samples representing the target population.

(Step-6) Analysing the data: To analyse the data, we first checked the completeness of the collected data (i.e., whether all questions are appropriately answered). Then, we manually analysed the answers to the open-ended questions to extract in-depth insights. For closed-ended questions, we summarised and presented key statistical results.

<u>Results</u>. **65% of the participants agree that model-agnostic techniques can generate the Time-contrast explanation to answer the why-questions.** Similarly, we found that 55% and 50% of the participants agree (and strongly agree) that model-agnostic techniques can generate the Property-contrast and Object-contrast explanations, respectively.

55% and 65% of the participants perceived that the Time-contrast explanations generated by model-agnostic techniques are necessary and useful, respectively. Similarly, we found that 40% and 30% of the participants perceive that the Property-contrast and

[2]https://www.google.com/forms

Object-contrast explanations generated by model-agnostic techniques are necessary, respectively. We found that 55% and 40% of the participants perceive that the Property-contrast and Object-contrast explanations generated by model-agnostic techniques are useful, respectively. Finally, we found that 50%, 45%, and 70% of the participants agree (and strongly agree) that instance explanations generated by model-agnostic techniques can build appropriate trusts of the predictions of defect models for the Property-contrast, Object-contrast, and Time-contrast explanations.

5.6 Discussion

In this section, we discuss the key differences between model-agnostic techniques and a tree-based technique.

5.6.1 A Comparison of Model-Agnostic Techniques with a Tree-based Technique

There are many approaches and granularity levels to explain the predictions of defect models (i.e., global explanation, subgroup explanation, instance explanation). Traditionally, we can use tree-based models to predict and explain the characteristics of defective files. For example, Tan *et al.* [204] use Alternative Decision Tree technique (ADTree) as provided by Weka [62] to explain the predictions of defect models. Chen *et al.* [29] use Fast-and-Frugal Trees (FFT) technique to construct comprehensible defect models. An explanation of each prediction can be generated by deriving a decision node of the decision tree that matches with the instance to explain. Below, we select the ADTree technique as the subject of this discussion and discuss the strengths and weakness of the tree-based technique with respect to the model accuracy, the locality of the explanation, and the visual explanation.

Model Accuracy

Practitioners often make decisions whether defect models should be deployed in practice based on their model accuracy. We first evaluate the model accuracy of the decision tree technique for predicting defective files with respect to three performance measures (i.e., AUC, Initial False Alarm (IFA), and Popt(20)) for all of the 32 studied defect datasets. We report the evaluation results of ADTree in the online appendix [83].

We find that ADTree achieves a median AUC of 0.75, a median IFA of 53, and a median Popt(20) of 0.02. When comparing the model accuracy to other classification techniques as shown in Figure 5.4, we find that ADTree is the least top-performing classification technique in terms of AUC, IFA, and Popt(20), raising concerns that the explanations that are derived from such inaccurate models could be misleading. However, such black-box AI/ML-based classification techniques are complex and hard to explain. Thus, model-agnostic techniques play a key role in explaining the predictions of highly accurate yet complex classification techniques.

Visual Explanation

Practitioners often make a decision whether the predictions should be trusted based on the understand-ability of the visual explanations. We conduct a preliminary survey with 20 practitioners to better understand which visual explanations are the most preferred by practitioners. We find that the visual explanation of ADTree is the most preferred by practitioners (60% of practitioners agree or strongly agree), as such visual explanation is simple to digest and involves logical reasoning. While the visual explanation of LIME also involves logical reasoning, practitioners are confused about the bar colours of LIME that explain the supporting and contradict scores (i.e., LIME uses the red colour to explain contradicting scores, which imply that such metrics contribute towards a prediction as clean). Despite the advantages of BreakDown that decompose the final probability score into each score for each feature, practitioners raise concerns that the visual explanation lacks necessary details (e.g., optimal threshold values for each metric) and is difficult to understand. Therefore, future studies should develop a novel visual explanation that is understandable to domain experts using human-centred design approaches (e.g., a co-creation design session and the Wizard-of-Oz prototyping technique).

The Locality of Explanation

Practitioners often make a decision as to whether the predictions should be trusted based on the locality of the explanations. The locality of explanations refers to the scope that such explanations are derived from. For example, an explanation generated by a variable importance technique of the random forest technique is derived from the global level of the prediction models. On the other hand, an explanation generated by a model-agnostic technique (e.g., LIME) is derived from a local model that is constructed from instances around their neighbours. Similarly, an explanation generated by a decision tree technique is derived from a decision node which can cover *N* instances. Although the locality of explanation between LIME and a decision tree is similar, the key difference is the flexibility of the visual representation and the choice of AI/ML-based classification techniques. In other words, a decision node can be used to explain only the decision tree technique. While the model accuracy of such decision tree technique is not as competitive as complex AI/ML-based classification techniques, model-agnostic techniques can be used to explain any classification techniques.

Summary. Decision tree is easy to understand, but not as accurate as other complex AI/ML learning algorithms. Complex AI/ML learning algorithms (e.g., xGBTree, neural network) are more accurate, but it is very hard to understand their predictions. Thus, the main goal of our paper is to leverage model-agnostic techniques to explain the predictions of any accurate yet complex AI/ML learning algorithms. However, practitioners perceive that decision tree is the most preferred visual explanation, suggesting that future studies should invent new visual explanations that are directed towards practitioners' needs.

5.7 Related Work

We discuss key related work in order to situate the contributions of our paper with respect to explainable software analytics and analytical models for software defects.

5.7.1 Explainable Software Analytics

Despite the advances in analytical modelling in software engineering, recent work raises a concern about a lack of explainability of analytical models in software engineering [32]. Practitioners also share similar concerns that analytical models in software engineering must be explainable and actionable in order to be of practical use [32, 115, 142]. For example, Dam *et al.* [32] argue that making software analytics models explainable to software practitioners is as important as achieving accurate predictions. Lewis *et al.* [115] emphasize that defect modelling should be more actionable to help Google engineers debug their programs. Menzies and Zimmermann [142] also emphasize that software analytics must be actionable.

Key Difference. To the best of our knowledge, little research in software analytics explores a theory of global explanations, the differences of the goals, scopes, and targets of global explanations and instance explanations. Moreover, model-agnostic techniques, i.e. techniques for explaining an individual prediction, have not yet been introduced in the context of software engineering.

5.7.2 Analytical Models for Software Defects (i.e., Defect Models)

Analytical models for software defects play a foundational role in optimising the limited resources of software quality assurance activities and in building empirical theories of software quality. Below, we discuss the literature with respect to the two main purposes of defect models.

To predict software defects. First, defect models are used to optimize the limited quality assurance resources on the most risky software modules [33, 140, 210, 213]. There are a plethora of studies focus on investigating advanced features and advanced techniques in order to improve the predictive ability of defect models. For example, Wang *et al.* [227] leverage a multiple kernel learning to produce a multiple kernel classifier through ensemble learning method, which has the advantages of both multiple kernel learning and ensemble learning. Wang *et al.* [224] use a deep belief network (DBN) to automatically learn semantic features to improve the predictive ability of defect models.

The improvement of the predictive ability of defect models is critical to practitioners when deploying defect models in practice.

Key Difference. Unlike prior studies that focused on improving the predictive ability of defect models, this study focuses on investigating techniques to explain software defect predictions.

To explain software defects. Second, defect models are used (1) to understand factors that are associated with software defects, (2) to establish effective quality improvement plans, (3) to provide actionable guidance to avoid pitfalls that lead to software defects in the past, and (4) to build empirical theories of software quality [17, 131, 216]. There are a plethora of studies focusing on investigating the best modelling workflow and advanced techniques in order to improve the explainability of defect models, which will be discussed below.

Investigating the best modelling workflow to improve the explainability of defect models

Explanations of defect models may not be accurate and reliable if care is not taken in the analytical modelling workflow for software defects. Hall *et al.* [65] raised a critical concern that different researchers often develop different analytical workflows, which makes it hard to derive practical guidelines of the best defect modelling workflows. Recent studies demonstrate that if pitfalls are not mitigated when collecting defect datasets [209, 233] and designing the analytical workflow [141, 205, 207] for software defects, the predictions and explanations that are derived from defect models may be inaccurate and unreliable. For example, Menzies and Shepperd [141] raised concerns that there are many components that could potentially impact the predictions of defect models.

Recent studies reveal many components of the analytical workflow that impact the predictions and explanations of defect models [207]. For example, noise in defect datasets [52], the quality of issue reports [209], defect labelling techniques [233], feature selection techniques [53, 89], collinearity analysis [85, 86, 89], class rebalancing techniques [208], model construction [52], parameter optimisation [2, 3, 49, 210, 213], model evaluation [212], and model interpretation [85].

Key Difference. While these studies focused on developing practical guidelines to develop the most accurate and reliable analytical models to predict and explain software defects, this paper focused on investigating advanced techniques to improve the explainability of defect models.

Investigating advanced techniques to improve the explainability of defect models

As discussed in Section 5.3, there are techniques to generate explanations with different goals, scopes, and target of explanations. Below, we discuss prior studies focused on (1) explaining a black-box model; (2) explaining a group of predictions; and (3) explaining an individual prediction.

Explaining a black-box model

Prior studies have been leveraged well-established explainable classification techniques, such as regression models [174, 177], random forests [97, 167], decision trees [237], decision rules [186]. In addition, Chen *et al.* [29] point out that Fast-and-Frugal Tree is more accurate, comprehensible, and operational than the well-established explainable classification techniques in the context of software defect prediction. Moreover, a domain-specific classification technique like Bellwether [108] has shown to mitigate the conclusion instability when transferring knowledge from one software project to another.

Despite the recent advances of well-established explainable classification techniques and domain-specific classification techniques in the context of software engineering, such techniques only derive knowledge of the learned models from the whole training dataset without justifying an individual prediction. Instead of explaining a black-box model, prior studies [12, 138] proposed techniques with an attempt to explain a smaller group of predictions with similar data characteristics in order to improve the predictive ability and explainability of defect models, such techniques still cannot justify each individual prediction and uphold the privacy laws (i.e., GDPR).

Explaining an individual prediction

Recently, model-agnostic algorithms that treat the original model as a black-box have been proposed to explain the predictions of any learners at the instance level. For example, Ribeiro *et al.* [182] proposed a Local Interpretable Model-Agnostic Explanations (LIME) that leverages the approximation of a simple linear model locally around the prediction.
Key Difference. Unlike prior studies that focus on explaining black-box models or a group of predictions, this study is the first to investigate model-agnostic techniques for explaining an individual prediction from testing instances in the domain of software engineering.

5.8 Threats to Validity

5.8.1 Construct Validity

Prior studies show that the parameters of learners have an impact on the performance of defect models [49, 106, 133, 134, 210]. While we use a default parameter setting of 100 trees for random forests, recent work [80, 210, 220] find that the parameters of random forests are insensitive to the performance of defect models. Thus, the parameters of random forests do not pose a critical threat to the validity of our study.

Due to the technical limitations of our studied classification techniques, correlated metrics must be removed prior to explaining the prediction models. One might suggest that LASSO should be used to penalise collinearity (i.e., correlated metrics). We found that the top-rank metric that is correlated with another will be less important by half when using LASSO (as they penalise the important score by half to another correlated metric). Yet, they are still correlated. We noted that this is still an open problem for ML domains. Thus, software practitioners should not draw implications solely from the most important metric, but should also consider its group of correlated metrics.

We use 100 iterations to draw reliable bootstrap estimates of the studied measures in the experiments. However, such high bootstrap iterations often come with a high computation cost. Thus, 100 iterations of bootstrap validation may not be practical for compute-intensive AI/ML algorithms like deep learning.

To ensure that our survey is reliable and valid, we carefully evaluated the survey (i.e., pre-testing [122]) prior to recruiting participants. We evaluated it with focus groups (i.e., practitioners) to assess the reliability and validity of the survey. We re-ran it to identify and fix potential problems (e.g., missing, unnecessary, or ambiguous questions) until reaching a consensus among the focus groups. Finally, the survey has been rigorously reviewed and approved by Monash University Human Research Ethics Committee (MUHREC Project ID: 22542).

5.8.2 Internal Validity

We studied a limited number of model-specific explanation techniques (i.e., the ANOVA Type-II analysis for logistic regression and the scaled Permutation Importance analysis for random forests) and model-agnostic techniques (i.e., BreakDown and LIME). Our results, therefore, may not generalise to other defect explainers. However, other techniques for generating explanations can be investigated in future studies. In this study, we provide a detailed methodology for others who wish to revisit our study with other techniques for generating explanations.

5.8.3 External Validity

In this study, our experiments rely on one defect prediction scenario, i.e., within-project defect prediction. However, there are a variety of defect prediction scenarios in the literature (e.g., cross-project defect prediction [27, 237], just-in-time defect prediction [96, 163], and heterogenous defect prediction [157]). Therefore, the results may differ in other scenarios. Future studies should revisit our study in other defect prediction scenarios.

The number of our studied datasets is limited and thus may produce results that cannot be generalised to other datasets and domains. However, it is a carefully curated dataset where the ground truths were labelled based on the affected releases. Future work can revisit and replicate our study with other datasets.

The number of survey participants is limited to a recruitment of 20 software practitioners from Amazon Mechanical Turk. Thus, the results of the survey may not be representative to the perceptions of all software practitioners. Future work can revisit and replicate the survey study with other groups of software practitioners.

5.9 Conclusions

We investigate model-agnostic techniques for explaining the predictions of defect models. Through a study of 32 publicly-available defect datasets of 9 large-scale open-source software systems, the experimental results lead us to conclude that (1) model-agnostic techniques are needed to explain individual predictions of defect models; (2) instance explanations generated by model-agnostic techniques are mostly overlapping with the global explanation of defect models (except for LIME) and reliable when they are re-generated (except for LIME); (3) model-agnostic techniques take less than a minute to generate instance explanations; and (4) more than half of the practitioners perceive that the instance explanations are necessary and useful to understand the predictions of defect models. Since the implementation of the studied model-agnostic techniques is readily available in both Python and R, we recommend model-agnostic techniques be used to explain the predictions of defect models.

5.9.1 Chapter Remarks

In this chapter, we set out to investigate the best techniques for explaining the predictions of defect models. The experimental results lead us to suggest that future studies should use both of the studied model-agnostic techniques (i.e., LIME and BreakDown) to explain the predictions of defect models to support decision- and policy-making. The experimental results from Chapter 4 enable the defect modelling workflow that produces the most accurate and reliable explanation of defect models, while the experimental results from Chapters 3 and 5 suggest the best techniques for explaining defect models and their predictions. However, our mission is not ended. Researchers and practitioners raise concerns that defect models must be explainable [32, 115, 143]. Nevertheless, no prior studies investigate how do practitioners perceive when adopting explainable defect models. Thus, as a post-evaluation study, we plan to investigate how do practitioners perceive when adopting explainable defect models produced by our recommended framework.

Chapter 6

Conclusions

In this thesis research, we aimed to **increase the explainability of defect prediction models to better support SQA planning**. To address this goal, we hypothesised that: *Explainable defect prediction models are needed to support SQA planning. Empirical studies are the way forward to identify the best explainable defect prediction framework to generate the most reliable explanations.* To validate the hypothesis, we formulated 3 key research questions: (1) how do correlated metrics impact the explanation of defect prediction models?, (2) which feature selection techniques should be used to mitigate correlated metrics for generating the most reliable explanation of defect prediction models?, and (3) should model-agnostic techniques be used to explain the predictions of defect prediction models?

In Chapter 2, we first conducted a literature analysis to understand the current state of research in defect prediction studies. Then, we conducted a qualitative survey to investigate practitioners' perceptions of defect prediction models. The results of the literature analysis show that most recent defect prediction studies (89%) focus on the prediction of defect prediction models. Little research has been done on understanding defect prediction models and their predictions to support SQA planning. Despite receiving little attention from the research community, the results of the qualitative survey show that 82%-84% of the respondents perceived understanding defect prediction models and their predictions as useful and 74%-78% of the respondents are willing to adopt them. These findings motivated us to further explore the explanation of defect prediction models to support SQA planning.

Through studies of publicly-available defect datasets that span across proprietary and open source domains, the experimental results show that:

(1) Correlated metrics impact the explanation of defect prediction models (Chapter 3).

(2) Removing correlated metrics improve the consistency of the explanation of defect prediction models with little impact on model performance. After removing correlated metrics, ones should use the ANOVA Type-II technique to explain logistic regression models, while using the scaled Permutation Importance technique to explain random forests models (Chapter 3).

(3) Commonly-used feature selection techniques do not mitigate correlated metrics. On the other hand, AutoSpearman automatically mitigates correlated metrics with little impact on model performance (Chapter 4).

(4) BreakDown and LIME can explain the predictions of defect prediction models within a few seconds. Since the implementation of the both studied techniques is readily available, future studies should use BreakDown and LIME to explain the predictions of defect prediction models to support decision- and policy-making (Chapter 5).

Below, we discuss the implications to researchers and practitioners, and highlight some key future research opportunities for software analytics researchers.

Implications for researchers

⟹ **More research effort is needed to improve the explainability of defect prediction models to understand such models and their predictions.** The analysis of related work in Chapter 2 shows that most of research effort (91%) has been put to improve the predictability of defect prediction models. On the other hand, little research has been done on understanding defect prediction models and their predictions. Despite receiving little attention from research community, as high as 82% of the respondents perceive understanding defect prediction models and their predictions as useful and 74%-78% of the respondents are willing to adopt them. These findings highlight the need for future research to put more effort in improving the explainability of defect prediction models to understand such models and their predictions.

⟹ **Strong correlation among studied metrics may impact the conclusions of prior studies that rely on the explanation of defect prediction models.** The results of Chapter 3 show that correlated metrics that impact the explanation of defect prediction models are prevalent in publicly-available defect datasets. Thus, the conclusions of prior studies that rely on such defect datasets and do not mitigate correlated metrics may be altered after mitigating correlated metrics. To refine and ensure the reliability of the conclusions of such studies, future research can revisit such studies and mitigate correlated metrics with correlation analyses, e.g., AutoSpearman (Chapter 4) that automatically mitigates correlated metrics better than other commonly-used feature selection techniques, prior to constructing defect prediction models.

⟹ **Explainability of defect prediction could be used to uncover new knowledge and refine the conclusions of prior studies.** Prior studies apply model explanation techniques to understand the most influential factors that impact software quality, assuming that such influential factors are representative to all of the defective code. However, the results of Chapter 5 show that such generic model explanation does not hold true for all instances since each individual instance explanation varies across different instances. More fine-grained approaches, e.g., model-agnostic techniques like LIME, BreakDown, and SHAP, could be used to uncover new knowledge about why a software module is likely to be defective and refine the conclusions of prior studies like code ownership [17, 216], code review practices [131, 216, 217], and code smells [99]. Thus, future work should revisit prior conclusions with instance explanations to better understand the influential factors at a more fine-grained level.

⟹ **Instance explanations may be applicable to other software engineering research topics.** (Chapter 5) Model-agnostic techniques like LIME, BreakDown, and SHAP can be used to open the black-box of predictive models in software engineering. Due to the nature of the model-agnostic techniques that can explain any learning algorithms, we advocate that such instance explanations should be applied beyond mining software engineering data for predictions. Thus, future work should integrate such model-agnostic techniques to understand how predictive models in software engineering work and why they make such predictions.

Implications for practitioners

⟹ **AutoSpearman can be used to automatically mitigate strong correlation among studied metrics prior to constructing and explaining defect prediction models.** The results of Chapter 3 highlight the negative impact of correlated metrics on the explanation of defect prediction models and suggest that correlated metrics must be mitigated prior to constructing and explaining defect prediction models. To do so, in Chapter 4, we investigated the consistency and correlation of subsets of metrics produced by commonly-used feature selection techniques and our contribution, AutoSpearman. The experimental results show that AutoSpearman mitigates correlated metrics better than other commonly-used feature selection techniques, suggesting that AutoSpearman should be used to automatically mitigate correlated metrics when aiming to explain defect prediction modes and their predictions.

⟹ **Instance explanations can provide more actionable guidance than model explanations.** Instance explanations can be used to answer why a file is predicted as defective (and not defective). As shown in Chapter 5, model-agnostic techniques can generate reliable instance explanations that are needed to explain individual predictions of defect prediction models. Such reliable instance explanations can help software practitioners in defect diagnosis and potentially provides clues towards solutions. Thus, software practitioners should adopt instance explanations.

Future Research Opportunities

Finally, our results also highlight future research opportunities for software analytics researchers (but not limited to):

⟹ **Developing practical guidelines of data science pipeline that enable a reliable explanation in software engineering.** Prior studies show that data science pipelines often impact the predictive accuracy [53, 207, 209, 210, 212, 213, 233] and the explainability [85, 86, 89] of prediction models in software engineering. Thus, future work should focus on developing practical guidelines for analytical modelling workflow that enable a reliable model-level and instance-level explanation, which is one crucial step towards *actionable software analytics*.

$\Longrightarrow$ **Developing domain-specific model-agnostic algorithms for software engineering.** Many model-agnostic techniques are originated from XAI literature. However, according to the experimental results in Chapter 5, there are rooms for improvement. We envision that a domain-specific model-agnostic technique for software engineering, that is more adapted to the characteristics of software engineering data, can address the limitation of the current state-of-the-art model-agnostic techniques. Furthermore, studies in this thesis mainly focus on the file-level defect prediction. We encourage future studies to develop domain-specific model-agnostic algorithms for other levels of defect prediction, e.g., JITLine [170] which is developed for the line-level defect prediction.

$\Longrightarrow$ **Exploring the explanation of defect prediction models in other scenarios.** Studies in this thesis mainly focus on with-in project defect prediction models. However, there are various other scenarios of defect prediction models, e.g., cross-project [118, 157]. This thesis provides a detailed explanation of experimental setups and techniques used in each study. Thus, future work can explore the explanation of defect prediction models in other scenarios.

Bibliography

[1] (2020). Rnalytica: An R package of the Miscellaneous Functions for Data Analytics Research.

[2] Agrawal, A. and Menzies, T. (2018). Is Better Data Better Than Better Data Miners?: On the Benefits of Tuning SMOTE for Defect Prediction. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 1050–1061.

[3] Agrawal, A., Menzies, T., Minku, L. L., Wagner, M., and Yu, Z. (2018). Better Software Analytics via" DUO": Data Mining Algorithms Using/Used-by Optimizers. *arXiv preprint arXiv:1812.01550*.

[4] Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who Should Fix This Bug? In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 361–370.

[5] Arisholm, E., Briand, L. C., and Johannessen, E. B. (2010). A Systematic and Comprehensive Investigation of Methods to Build and Evaluate Fault Prediction Models. *Journal of Systems and Software (JSS)*, 83(1):2–17.

[6] Assila, A., Ezzedine, H., et al. (2016). Standardized Usability Questionnaires: Features and Quality Focus. *Electronic Journal of Computer Science and Information Technology: eJCIST*, 6(1).

[7] Barnett, J. G., Gathuru, C. K., Soldano, L. S., and McIntosh, S. (2016). The Relationship between Commit Message Detail and Defect Proneness in Java Projects on GitHub. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 496–499.

[8] Bas, C. V. F. (1980). *The Scientific Image*. Oxford University Press.

[9] Basili, V. R., Briand, L. C., and Melo, W. L. (1996). A Validation of Object-oriented Design Metrics as Quality Indicators. *Transactions on Software Engineering (TSE)*, 22(10):751–761.

[10] Berry, W. D. (1993). *Understanding Regression Assumptions*, volume 92. Sage Publications.

[11] Bettenburg, N. and Hassan, A. E. (2010). Studying the Impact of Social Structures on Software Quality. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pages 124–133.

[12] Bettenburg, N., Nagappan, M., and Hassan, A. E. (2012). Think Locally, Act Globally: Improving Defect and Effort Prediction Models. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 60–69.

[13] Bhattacharya, P. and Neamtiu, I. (2011). Assessing Programming Language Impact on Development and Maintenance: A Study on C and C++. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 171–180.

[14] Biecek, P. and Grudziaz, A. (2017). pyBreakDown: Python implementation of R package breakDown. *Github Repository https://github.com/MI2DataLab/pyBreakDown.*

[15] Biecek, P. and Grudziaz, A. (2018). breakDown: Model Agnostic Explainers for Individual Predictions. R package version 0.1.6. *Software available at URL: https://cran.r-project.org/package=breakDown.*

[16] Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., and Devanbu, P. (2009a). Fair and Balanced?: Bias in Bug-fix Datasets. In *Proceedings of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 121–130.

[17] Bird, C., Murphy, B., and Gall, H. (2011). Don't Touch My Code ! Examining the Effects of Ownership on Software Quality. In *Proceedings of the European Conference on Foundations of Software Engineering (ESEC/FSE)*, pages 4–14.

[18] Bird, C., Nagappan, N., Devanbu, P., Gall, H., and Murphy, B. (2009b). Does Distributed Development Affect Software Quality?: An Empirical Case Study of Windows Vista. *Communications of the ACM*, 52(8):85–93.

[19] Blake, C. and Merz, C. (1998). UCI Repository of Machine Learning Databases. *University of California, Irvine, CA*, 55.

[20] Bowes, D., Hall, T., Harman, M., Jia, Y., Sarro, F., and Wu, F. (2016). Mutation-Aware Fault Prediction. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 330–341.

[21] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[22] Breiman, L. (2002). Manual on setting up, using, and understanding random forests v3. 1. *Statistics Department University of California Berkeley, CA, USA*, 1.

[23] Breiman, L., Cutler, A., Liaw, A., and Wiener, M. (2006). randomForest : Breiman and Cutler's Random Forests for Classification and Regression. R package version 4.6-12. *Software available at URL: https://cran.r-project.org/package=randomForest.*

[24] Briand, L. C., Melo, W. L., and Wust, J. (2002). Assessing the Applicability of Fault-proneness Models across Object-oriented Software Projects. *Transactions on Software Engineering (TSE)*, 28(7):706–720.

[25] Briand, L. C., Wüst, J., Daly, J. W., and Porter, D. V. (2000). Exploring the Relationships between Design Measures and Software Quality in Object-oriented Systems. *Journal of Systems and Software (JSS)*, 51(3):245–273.

[26] Cahill, J., Hogan, J. M., and Thomas, R. (2013). Predicting Fault-prone Software Modules with Rank Sum Classification. In *Proceedings of the Australian Software Engineering Conference (ASWEC)*, pages 211–219.

[27] Canfora, G., De Lucia, A., Di Penta, M., Oliveto, R., Panichella, A., and Panichella, S. (2013). Multi-objective Cross-project Defect Prediction. In *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST)*, pages 252–261.

[28] Chambers, J. M. (1992). Statistical Models in S. Wadsworth. *Pacific Grove, California*.

[29] Chen, D., Fu, W., Krishna, R., and Menzies, T. (2018). Applications of Psychological Science for Actionable Analytics. In *Proceedings of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 456–467.

[30] Chidamber, S. R. and Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. *Transactions on Software Engineering (TSE)*, 20(6):476–493.

[31] Coelho, E. and Basu, A. (2012). Effort Estimation in Agile Software Development using Story Points. *International Journal of Applied Information Systems (IJAIS)*, 3(7).

[32] Dam, H. K., Tran, T., and Ghose, A. (2018). Explainable Software Analytics. In *Proceedings of the International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 53–56.

[33] D'Ambros, M., Lanza, M., and Robbes, R. (2010). An Extensive Comparison of Bug Prediction Approaches. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 31–41.

[34] D'Ambros, M., Lanza, M., and Robbes, R. (2012). Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison. *Empirical Software Engineering (EMSE)*, 17(4-5):531–577.

[35] Dash, M., Liu, H., and Motoda, H. (2000). Consistency based Feature Selection. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 98–109.

[36] Devanbu, P., Zimmermann, T., and Bird, C. (2016). Belief & Evidence in Empirical Software Engineering. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 108–119.

[37] Di Penta, M., Cerulo, L., Guéhéneuc, Y.-G., and Antoniol, G. (2008). An Empirical Study of the Relationships between Design Pattern Roles and Class Change Proneness. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 217–226.

[38] Edwards, P., Roberts, I., Clarke, M., DiGuiseppi, C., Pratap, S., Wentz, R., and Kwan, I. (2002). "Increasing Response Rates to Postal Questionnaires: Systematic Review". *Bmj*, 324(7347):1183.

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

[40] Elish, K. O. and Elish, M. O. (2008). Predicting Defect-prone Software Modules using Support Vector Machines. *Journal of Systems and Software (JSS)*, 81(5):649–660.

[41] Esteves, G., Figueiredo, E., Veloso, A., Viggiato, M., and Ziviani, N. (2020). Understanding Machine Learning Software Defect Predictions. *Automated Software Engineering*.

[42] Fisher, R. (1925). Intraclass correlations and the analysis of variance. *Statistical Methods for Research Workers*, pages 187–210.

[43] Fox, J. (2015). *Applied regression analysis and generalized linear models.* Sage Publications.

[44] Fox, J. and Monette, G. (1992). Generalized Collinearity Diagnostics. *Journal of the American Statistical Association (JASA)*, 87(417):178–183.

[45] Fox, J., Weisberg, S., and Price, B. (2020). car: Companion to Applied Regression. R package version 3.0-2. *Software available at URL: https://cran.r-project.org/web/packages/car.*

[46] French, S. (2014). Decision Analysis. *Wiley StatsRef: Statistics Reference Online.*

[47] Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The Elements of Statistical Learning*, volume 1. Springer series in statistics.

[48] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232.

[49] Fu, W., Menzies, T., and Shen, X. (2016). Tuning for Software Analytics: Is it really necessary? *Information and Software Technology*, 76:135–146.

[50] Garner, S. R. et al. (1995). Weka: The Waikato Environment for Knowledge Analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference (NZCSRSC)*, pages 57–64.

[51] Gevrey, M., Dimopoulos, I., and Lek, S. (2003). Review and comparison of methods to study the contribution of variables in artificial neural network models. *Ecological Modelling*, 160(3):249–264.

[52] Ghotra, B., McIntosh, S., and Hassan, A. E. (2015). Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 789–800.

[53] Ghotra, B., Mcintosh, S., and Hassan, A. E. (2017). A large-scale study of the impact of feature selection techniques on defect classification models. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 146–157.

[54] Gil, Y. and Lalouche, G. (2017). On the Correlation between Size and Metric Validity. *Empirical Software Engineering (EMSE)*, 22(5):2585–2611.

[55] Gosiewska, A. and Biecek, P. (2019). iBreakDown: Uncertainty of Model Explanations for Non-additive Predictive Models. *arXiv preprint arXiv:1903.11420.*

[56] Gousios, G., Pinzger, M., and Deursen, A. v. (2014). An Exploratory Study of the Pull-based Software Development Model. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 345–355.

[57] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018a). A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42.

[58] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Pedreschi, D., and Giannotti, F. (2018b). A Survey Of Methods For Explaining Black Box Models. 51(5):1–45.

[59] Guo, L., Ma, Y., Cukic, B., and Singh, H. (2004). Robust Prediction of Fault-proneness by Random Forests. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, pages 417–428.

[60] Guyon, I. and Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182.

[61] Hair, J. F., Black, W. C., Babin, B. J., Anderson, R. E., Tatham, R. L., et al. (2006). Multivariate Data Analysis (Vol. 6).

[62] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

[63] Hall, M. A. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato Hamilton.

[64] Hall, M. A. and Smith, L. A. (1997). Feature Subset Selection: A Correlation Based Filter Approach.

[65] Hall, T., Beecham, S., Bowes, D., Gray, D., and Counsell, S. (2012). A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *Transactions on Software Engineering (TSE)*, 38(6):1276–1304.

[66] Hanley, J. a. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(4):29–36.

[67] Harrell Jr, F. E. (2013). Hmisc: Harrell miscellaneous. R package version 3.12-2. *Software available at URL: http://cran.r-project.org/web/packages/Hmisc*.

[68] Harrell Jr, F. E. (2015). *Regression Modeling Strategies : With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*. Springer.

[69] Harrell Jr, F. E. (2017). rms: Regression Modeling Strategies. R package version 5.1-1.

[70] Hassan, A. E. (2009). Predicting Faults using the Complexity of Code Changes. In *Prooceedings of the International Conference on Software Engineering (ICSE)*, pages 78–88.

[71] Hata, H., Mizuno, O., and Kikuno, T. (2012). Bug prediction based on fine-grained module histories. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 200–210.

[72] Hemmati, H., Nadi, S., Baysal, O., Kononenko, O., Wang, W., Holmes, R., and Godfrey, M. W. (2013). The MSR Cookbook: Mining a Decade of Research. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 343–352.

[73] Herraiz, I., German, D. M., and Hassan, A. E. (2011). On the Distribution of Source Code File Sizes. In *Proceedings of the International Conference on Software Technologies (ICSOFT)*, pages 5–14.

[74] Hilderbrand, C., Perdriau, C., Letaw, L., Emard, J., Steine-Hanson, Z., Burnett, M., and Sarma, A. (2020). Engineering gender-inclusivity into software: ten teams' tales from the trenches. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 433–444.

[75] Hilton, D. J., McClure, J. J., and Slugoski, B. R. (2005). The Psychology of Counterfactual Thinking. chapter The course of events: counterfactuals, causal sequences, and explanation, pages 44–60. Routledge Research International Series in Social Psychology, Routledge, Abingdon, Oxon, UK.

[76] Hinkle, D. E., Wiersma, W., and Jurs, S. G. (2003). *Applied Statistics for the Behavioral Sciences*, volume 663. Houghton Mifflin College Division.

[77] Horne, Z., Muradoglu, M., and Cimpian, A. (2019). Explanation as a Cognitive Process. *Trends in Cognitive Sciences*.

[78] Hosseini, S., Turhan, B., and Gunarathna, D. (2017). A Systematic Literature Review and Meta-analysis on Cross Project Defect Prediction. *Transactions on Software Engineering (TSE)*, 45(2):111–147.

[79] Huang, Q., Xia, X., and Lo, D. (2017). Supervised vs Unsupervised Models: A Holistic Look at Effort-Aware Just-in-Time Defect Prediction. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 159–170.

[80] Jiang, Y., Cukic, B., and Menzies, T. (2008). Can Data Transformation Help in the Detection of Fault-prone Modules? In *Proceedings of the International Workshop on Defects in Large Software Systems (DEFECTS)*, pages 16–20.

[81] Jiarpakdee, J., Tantithamthavorn, C., Dam, H. K., and Grundy, J. (2020). An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models. *Transactions on Software Engineering (TSE)*.

[82] Jiarpakdee, J., Tantithamthavorn, C., Dam, H. K., and Grundy, J. (2020). An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models. *Transactions on Software Engineering (TSE)*, page To Appear.

[83] Jiarpakdee, J., Tantithamthavorn, C., Dam, H. K., and Grundy, J. (2020a). Online Appendix for "An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models". https://github.com/awsm-research/model-agnostic-online-appendix.

[84] Jiarpakdee, J., Tantithamthavorn, C., and Grundy, J. (2021a). Online supplementary materials for Practitioners' Perceptions of the Goals and Visual Explanations of Defect Prediction Models. *Available at URL: https://doi.org/10.5281/zenodo.4536698*.

[85] Jiarpakdee, J., Tantithamthavorn, C., and Hassan, A. E. (2019a). The Impact of Correlated Metrics on the Interpretation of Defect Models. *Transactions on Software Engineering (TSE)*.

[86] Jiarpakdee, J., Tantithamthavorn, C., Ihara, A., and Matsumoto, K. (2016). A Study of Redundant Metrics in Defect Prediction Datasets. In *Proceedings of the International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 51–52.

[87] Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2018a). Artefact: An R Implementation of the AutoSpearman Function. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 711–711.

[88] Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2018b). AutoSpearman: Automatically Mitigating Correlated Metrics for Interpreting Defect Models. In *Proceeding of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 92–103.

[89] Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2018c). AutoSpearman: Automatically Mitigating Correlated Software Metrics for Interpreting Defect Models. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 92–103.

[90] Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2019b). Online Appendix for "The Impact of Automated Feature Selection Techniques on the Interpretation of Defect Models". *Available at URL: https://github.com/software-analytics/autospearman-extension-appendix.*

[91] Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2020b). The Impact of Automated Feature Selection Techniques on the Interpretation of Defect Models. *Empirical Software Engineering (EMSE).*

[92] Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2021b). Practitioners' Perceptions of the Goals and Visual Explanations of Defect Prediction Models. In *Proceedings of the International Conference on Mining Software Repositories (MSR).*

[93] John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant Features and the Subset Selection Problem. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 121–129.

[94] Jureczko, M. and Madeyski, L. (2010). Towards Identifying Software Project Clusters with Regard to Defect Prediction. In *Proceedings of the International Conference on Predictive Models in Software Engineering (PROMISE)*, page 9.

[95] Kabinna, S., Shang, W., Bezemer, C.-P., and Hassan, A. E. (2016). Examining the Stability of Logging Statements. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 326–337.

[96] Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., and Ubayashi, N. (2013). A Large-Scale Empirical Study of Just-In-Time Quality Assurance. *Transactions on Software Engineering (TSE)*, 39(6):757–773.

[97] Kaur, A. and Malhotra, R. (2008). Application of Random Forest in Predicting Fault-prone Classes. In *Proceedings of International Conference on the Advanced Computer Theory and Engineering (ICACTE)*, pages 37–43.

[98] Keung, J., Kocaguneli, E., and Menzies, T. (2013). Finding Conclusion Stability for Selecting the Best Effort Predictor in Software Effort Estimation. *Automated Software Engineering*, 20(4):543–567.

[99] Khomh, F., Di Penta, M., and Gueheneuc, Y.-G. (2009). An Exploratory Study of the Impact of Code Smells on Software Change-proneness. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, pages 75–84.

[100] Khomh, F., Di Penta, M., Guéhéneuc, Y.-G., and Antoniol, G. (2012). An Exploratory Study of the Impact of Antipatterns on Class Change-and Fault-proneness. *Empirical Software Engineering (EMSE)*, 17(3):243–275.

[101] Kim, S., Zhang, H., Wu, R., and Gong, L. (2011). Dealing with Noise in Defect Prediction. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 481–490.

[102] Kim, S., Zimmermann, T., Whitehead Jr, E. J., and Zeller, A. (2007). Predicting Faults from Cached History. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 489–498.

[103] Kirbas, S., Caglayan, B., Hall, T., Counsell, S., Bowes, D., Sen, A., and Bener, A. (2017). The Relationship between Evolutionary Coupling and Defects in Large Industrial Software. *Journal of Software: Evolution and Process*, 29(4).

[104] Kitchenham, B. A. and Pfleeger, S. L. (2008). Personal opinion surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92.

[105] Kohavi, R. and John, G. H. (1997). Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1-2):273–324.

[106] Koru, A. G. and Liu, H. (2005). An Investigation of the Effect of Module Size on Defect Prediction Using Static Measures. *Software Engineering Notes (SEN)*, 30:1–5.

[107] Kraemer, H. C., Morgan, G. A., Leech, N. L., Gliner, J. A., Vaske, J. J., and Harmon, R. J. (2003). Measures of Clinical Significance. *Journal of the American Academy of Child & Adolescent Psychiatry (JAACAP)*, 42(12):1524–1529.

[108] Krishna, R. and Menzies, T. (2017). Simpler Transfer Learning (Using "Bellwethers"). pages 1–23.

[109] Krishna, R. and Menzies, T. (2020). Learning Actionable Analytics from Multiple Software Projects. *Empirical Software Engineering (EMSE)*, 25(5):3468–3500.

[110] Krosnick, J. A. (1999). Survey research. *Annual Review of Psychology*, 50(1):537–567.

[111] Kuhn, M., Wing, J., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., Team, R., et al. (2017). caret: Classification and regression training. R package version 6.0–78. *Software available at URL: https://cran.r-project.org/web/packages/caret*.

[112] Lamkanfi, A., Pérez, J., and Demeyer, S. (2013). The Eclipse and Mozilla Defect Tracking Dataset: A Genuine Dataset for Mining Bug Information. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 203–206.

[113] Leake, D. B. (2014). *Evaluating Explanations: A Content Theory*. Psychology Press.

[114] Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *Transactions on Software Engineering (TSE)*, 34(4):485–496.

[115] Lewis, C., Lin, Z., Sadowski, C., Zhu, X., Ou, R., and Whitehead Jr, E. J. (2013). Does Bug Prediction Support Human Developers? Findings from a Google Case Study. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 372–381.

[116] Lewis, J. R. (1992). Psychometric evaluation of the post-study system usability questionnaire: The PSSUQ. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 36, pages 1259–1260.

[117] Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., and Liu, H. (2017a). Feature Selection: A Data Perspective. *ACM Computing Surveys (CSUR)*, 50(6):94.

[118] Li, Z., Jing, X.-Y., Zhu, X., and Zhang, H. (2017b). Heterogeneous Defect Prediction through Multiple Kernel Learning and Ensemble Learning. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 91–102.

[119] Liang, H., Yu, Y., Jiang, L., and Xie, Z. (2019). Seml: A semantic LSTM model for software defect prediction. *IEEE Access*, 7:83812–83824.

[120] Lipton, P. (1990). Contrastive explanation. *Royal Institute of Philosophy Supplement*, 27:247–266.

[121] Lipton, Z. C. (2016). The Mythos of Model Interpretability. In *Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine Learning*, pages 96–100.

[122] Litwin, M. S. (1995). *How to Measure Survey Reliability and Validity*, volume 7. Sage.

[123] Lombrozo, T. (2006). The structure and function of explanations. *Trends in Cognitive Sciences*, 10(10):464–70.

[124] Lu, H., Kocaguneli, E., and Cukic, B. (2014). Defect Prediction between Software Versions with Active Learning and Dimensionality Reduction. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, pages 312–322.

[125] Lundberg, S. M. and Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4765–4774.

[126] Mason, C. H. and Perreault Jr, W. D. (1991). Collinearity, Power, and Interpretation of Multiple Regression Analysis. *Journal of Marketing Research (JMR)*, pages 268–280.

[127] Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451.

[128] McCabe, T. J. (1976). A Complexity Measure. *Transactions on Software Engineering (TSE)*, (4):308–320.

[129] McHugh, M. L. (2013). The Chi-square Test of Independence. *Biochemia Medica*, 23(2):143–149.

[130] McIntosh, S. and Kamei, Y. (2017). Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction. *Transactions on Software Engineering (TSE)*.

[131] McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2014). The Impact of Code Review Coverage and Code Review Participation on Software Quality. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 192–201.

[132] McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2016). An Empirical Study of the Impact of Modern Code Review Practices on Software Quality. *Empirical Software Engineering (EMSE)*, 21(5):2146–2189.

[133] Mende, T. (2010). Replication of Defect Prediction Studies: Problems, Pitfalls and Recommendations. In *Proceedings of the International Conference on Predictive Models in Software Engineering (PROMISE)*, pages 1–10.

[134] Mende, T. and Koschke, R. (2009a). Revisiting the Evaluation of Defect Prediction Models. *Proceedings of the International Conference on Predictive Models in Software Engineering (PROMISE)*, pages 7–16.

[135] Mende, T. and Koschke, R. (2009b). Revisiting the Evaluation of Defect Prediction Models. In *Proceedings of the International Conference on Predictor Models in Software Engineering (PROMISE)*, page 7.

[136] Meneely, A., Williams, L., Snipes, W., and Osborne, J. (2008). Predicting Failures with Developer Networks and Social Network Analysis. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, pages 13–23.

[137] Menzies, T. (2018). The Unreasonable Effectiveness of Software Analytics. *IEEE Software*, 35(2):96–98.

[138] Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B., and Zimmermann, T. (2013). Local Versus Global Lessons for Defect Prediction and Effort Estimation. *Transactions on Software Engineering (TSE)*, 39(6):822–834.

[139] Menzies, T., Caglayan, B., Kocaguneli, E., Krall, J., Peters, F., and Turhan, B. (2012). The Promise Repository of Empirical Software Engineering Data.

[140] Menzies, T., Greenwald, J., and Frank, A. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. *Transactions on Software Engineering (TSE)*, 33(1):2–13.

[141] Menzies, T. and Shepperd, M. (2012). Special Issue on Repeatable Results in Software Engineering Prediction. *Empirical Software Engineering (EMSE)*, pages 1–17.

[142] Menzies, T. and Zimmermann, T. (2018a). Software Analytics: So What? *IEEE Software*, (4):31–37.

[143] Menzies, T. and Zimmermann, T. (2018b). Software Analytics: Whats next? *IEEE Software*, (4).

[144] Mersmann, O., Beleites, C., Hurling, R., and Friedman, A. (2019). microbenchmark: Accurate Timing Functions. R package version 1.4-4. *Software available at URL: https://cran.r-project.org/package=microbenchmark*.

[145] Miller, K. W. and Larson, D. K. (2005). Agile software development: human values and culture. *IEEE Technology and Society Magazine*, 24(4):36–42.

[146] Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artifical Intelligence*, 267:1–38.

[147] Misirli, A. T., Shihab, E., and Kamei, Y. (2016). Studying High Impact Fix-Inducing Changes. *Empirical Software Engineering (EMSE)*, 21(2):605–641.

[148] Mitchell, T. M. (1997). Machine Learning. *McGraw Hill*.

[149] Mohanty, H., Mohanty, J., and Balakrishnan, A. (2017). *Trends in Software Testing*. Springer.

[150] Morales, R., McIntosh, S., and Khomh, F. (2015). Do Code Review Practices Impact Design Quality? : A Case Study of the Qt, VTK, and ITK Projects. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 171–180.

[151] Moser, R., Pedrycz, W., and Succi, G. (2008). A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 181–190.

[152] Mullen, K., Ardia, D., Gil, D. L., Windover, D., and Cline, J. (2011). DEoptim: An R package for global optimization by differential evolution. *Journal of Statistical Software*, 40(6):1–26.

[153] Nagappan, N. and Ball, T. (2005). Use of Relative Code Churn Measures to Predict System Defect Density. *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 284–292.

[154] Nagappan, N., Ball, T., and Zeller, A. (2006). Mining Metrics to Predict Component Failures. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 452–461.

[155] Nagappan, N., Murphy, B., and Basili, V. (2008). The Influence of Organizational Structure on Software Quality. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 521–530.

[156] Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K., and Murphy, B. (2010). Change Bursts as Defect Predictors. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, pages 309–318.

[157] Nam, J., Fu, W., Kim, S., Menzies, T., and Tan, L. (2017). Heterogeneous Defect Prediction. *Transactions on Software Engineering (TSE)*, page In Press.

[158] Natekin, A. and Knoll, A. (2013). Gradient Boosting Machines, A Tutorial. *Frontiers in Neurorobotics*, 7:21.

[159] Nelson, E. A. (1967). Management Handbook for the Estimation of Computer Programming Costs. Technical report, SYSTEM DEVELOPMENT CORP SANTA MONICA CA.

[160] Okutan, A. and Yıldız, O. T. (2014). Software Defect Prediction using Bayesian Networks. *Empirical Software Engineering (EMSE)*, 19(1):154–181.

[161] Osman, H., Ghafari, M., and Nierstrasz, O. (2018). The Impact of Feature Selection on Predicting the Number of Bugs. *arXiv preprint arXiv:1807.04486*.

[162] Pandari, Y., Thangavel, P., Senthamaraikannan, H., and Jagadeeswaran, S. (2019). HybridFS: A Hybrid Filter-Wrapper Feature Selection Method. R package version 0.1.3. *Software available at URL: https://cran.r-project.org/package=HybridFS*.

[163] Pascarella, L., Palomba, F., and Bacchelli, A. (2018). Fine-Grained Just-In-Time Defect Prediction. *Journal of Systems and Software (JSS)*.

[164] Pedersen, T. L. and Benesty, M. (2019). lime: Local Interpretable Model-Agnostic Explanations. R package version 0.4.0. *Software available at URL: https://cran.r-project.org/web/packages/lime*.

[165] Peng, K. and Menzies, T. (2020a). Defect Reduction Planning (using TimeLIME). *arXiv preprint arXiv:2006.07416*.

[166] Peng, K. and Menzies, T. (2020b). How to Improve AI Tools (by Adding in SE Knowledge): Experiments with the TimeLIME Defect Reduction Tool. *arXiv preprint arXiv:2003.06887*.

[167] Petkovic, D., Sosnick-Pérez, M., Okada, K., Todtenhoefer, R., Huang, S., Miglani, N., and Vigil, A. (2016). Using the random forest classifier to assess and predict student learning of Software Engineering Teamwork. In *The Frontiers in Education Conference (FIE)*, pages 1–7.

[168] Petrić, J., Bowes, D., Hall, T., Christianson, B., and Baddoo, N. (2016). The Jinx on the NASA Software Defect Data Sets. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 13–17.

[169] Planning, S. (2002). The Economic Impacts of Inadequate Infrastructure for Software Testing. *National Institute of Standards and Technology*.

[170] Pornprasit, C. and Tantithamthavorn, C. (2021). JITLine: A Simpler, Better, Faster, Finer-grained Just-In-Time Defect Prediction. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*.

[171] Quinlan, J. R. (1987). Simplifying Decision Trees. *International Journal of Man-machine Studies*, 27(3):221–234.

[172] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*.

[173] Rahman, F. and Devanbu, P. (2011). Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 491–500.

[174] Rahman, F. and Devanbu, P. (2013). How, and Why, Process Metrics are Better. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 432–441.

[175] Rahman, F., Khatri, S., Barr, E. T., and Devanbu, P. (2014). Comparing Static Bug Finders and Statistical Prediction. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 424–434.

[176] Rahman, F., Posnett, D., Herraiz, I., and Devanbu, P. (2013). Sample Size vs. Bias in Defect Prediction. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 147–157.

[177] Rajbahadur, G. K., Wang, S., Kamei, Y., and Hassan, A. E. (2017). The Impact of Using Regression Models to Build Defect Classifiers. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 135–145.

[178] Ray, B., Posnett, D., Filkov, V., and Devanbu, P. (2014). A Large Scale Study of Programming Languages and Code Quality in Github. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, pages 155–165.

[179] Regulation, G. D. P. (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with regard to the Processing of Personal Data and on the Free Movement of such Data, and Repealing Directive 95/46. *Official Journal of the European Union (OJ)*, 59(1-88):294.

[180] Ribeiro, M. (2016). lime: Explaining the predictions of any machine learning classifier. *Github Repository https://github. com/marcotcr/lime.*

[181] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016a). Model-agnostic Interpretability of Machine Learning. *arXiv preprint arXiv:1606.05386.*

[182] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016b). Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDDM)*, pages 1135–1144.

[183] Ribeiro, M. T., Singh, S., and Guestrin, C. (2018). "Anchors: High-precision model-agnostic explanations". In *Proceedings of the AAAI Conference on Artificial Intelligence.*

[184] Robles, G. (2010). Replicating MSR: A Study of the Potential Replicability of Papers Published in the Mining Software Repositories Proceedings. In *Proceedings of the International Conference on Mining Software Repositories (MSR)*, pages 171–180.

[185] Rodríguez, D., Ruiz, R., Cuadrado-Gallego, J., and Aguilar-Ruiz, J. (2007). Detecting Fault Modules Applying Feature Selection to Classifiers. In *Proceedings of the International Conference on Information Reuse and Integration (IRI)*, pages 667–672.

[186] Rodríguez, D., Ruiz, R., Riquelme, J. C., and Aguilar-Ruiz, J. S. (2012). Searching for Rules to Detect Defective Modules: A Subgroup Discovery Approach. *Information Sciences*, 191:14–30.

[187] Romano, J., Kromrey, J. D., Coraggio, J., and Skowronek, J. (2006). Appropriate Statistics for Ordinal Level Data: Should we really be using T-test and Cohen's d for Evaluating group differences on the NSSE and other surveys. In *Annual meeting of the Florida Association of Institutional Research (FAIR)*, pages 1–33.

[188] Romanski, P. and Kotthoff, L. (2013). FSelector: Selecting attributes. R package version 0.19. *Software available at URL: https://cran.r-project.org/web/packages/FSelector.*

[189] Salmon, W. C. (1984). *Scientific explanation and the causal structure of the world.* Princeton University Press Princeton, N.J.

[190] Sarle, W. (1990). The VARCLUS Procedure. SAS/STAT UserGuide. SAS Institute. *Inc., Cary, NC, USA*.

[191] Shepperd, M., Bowes, D., and Hall, T. (2014). Researcher Bias: The Use of Machine Learning in Software Defect Prediction. *Transactions on Software Engineering (TSE)*, 40(6):603–616.

[192] Shepperd, M., Song, Q., Sun, Z., and Mair, C. (2013). Data Quality: Some Comments on the NASA Software Defect Datasets. *Transactions on Software Engineering (TSE)*, 39(9):1208–1215.

[193] Shihab, E. (2012). *An Exploration of Challenges Limiting Pragmatic Software Defect Prediction*. PhD thesis, Queen's University.

[194] Shihab, E., Bird, C., and Zimmermann, T. (2012). The Effect of Branching Strategies on Software Quality. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 301–310.

[195] Shihab, E., Jiang, Z. M., Ibrahim, W. M., Adams, B., and Hassan, A. E. (2010). Understanding the Impact of Code and Process Metrics on Post-release Defects: A Case Study on the Eclipse Project. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 4–10.

[196] Shimagaki, J., Kamei, Y., McIntosh, S., Hassan, A. E., and Ubayashi, N. (2016). A Study of the Quality-Impacting Practices of Modern Code Review at Sony Mobile. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 212–221.

[197] Shin, Y., Meneely, A., Williams, L., and Osborne, J. A. (2011). Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *Transactions on Software Engineering (TSE)*, 37(6):772–787.

[198] Shivaji, S., Whitehead, E. J., Akella, R., and Kim, S. (2013). Reducing Features to Improve Code Change-Based Bug Prediction. *Transactions on Software Engineering (TSE)*, 39(4):552–569.

[199] Shrikanth, N. and Menzies, T. (2020). Assessing Practitioner Beliefs about Software Defect Prediction. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 182–190.

[200] Smith, E., Loftin, R., Murphy-Hill, E., Bird, C., and Zimmermann, T. (2013). "Improving Developer Participation Rates in Surveys". In *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 89–92.

[201] Staniak, M. and Biecek, P. (2018). Explanations of Model Predictions with live and breakDown Packages. *arXiv preprint arXiv:1804.01955*.

[202] Storn, R. and Price, K. (1997). Differential Evolution–A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359.

[203] Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., and Zeileis, A. (2008). Conditional Variable Importance for Random Forests. *BMC Bioinformatics*, 9(1):307.

[204] Tan, M., Tan, L., Dara, S., and Mayeux, C. (2015). Online Defect Prediction for Imbalanced Data. In *Proceedings of the International Conference on Software Engineering (ICSE)*, volume 2, pages 99–108.

[205] Tantithamthavorn, C. (2016). Towards a Better Understanding of the Impact of Experimental Components on Defect Prediction Modelling. In *Companion Proceeding of the International Conference on Software Engineering (ICSE)*, pages 867—-870.

[206] Tantithamthavorn, C. (2017). ScottKnottESD : The Scott-Knott Effect Size Difference (ESD) Test. R package version 2.0. *Software available at URL: https://cran.r-project.org/web/packages/ScottKnottESD*.

[207] Tantithamthavorn, C. and Hassan, A. E. (2018). An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 286–295.

[208] Tantithamthavorn, C., Hassan, A. E., and Matsumoto, K. (2018a). The Impact of Class Rebalancing Techniques on The Performance and Interpretation of Defect Prediction Models. *Transactions on Software Engineering (TSE)*, page In Press.

[209] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Ihara, A., and Matsumoto, K. (2015). The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models. In *Proceeding of the International Conference on Software Engineering (ICSE)*, pages 812–823.

[210] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., and Matsumoto, K. (2016a). Automated Parameter Optimization of Classification Techniques for Defect Prediction Models. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 321–332.

[211] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., and Matsumoto, K. (2016b). Comments on Bias: The Use of Machine Learning in Software Defect Prediction”. *Transactions on Software Engineering (TSE)*, 42(11):1092–1094.

[212] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., and Matsumoto, K. (2017). An Empirical Comparison of Model Validation Techniques for Defect Prediction Models. *Transactions on Software Engineering (TSE)*, 43(1):1–18.

[213] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., and Matsumoto, K. (2018b). The Impact of Automated Parameter Optimization on Defect Prediction Models. *Transactions on Software Engineering (TSE)*, page In Press.

[214] Team, R. C. and contributors worldwide (2017). stats : The R Stats Package. R Package. Version 3.4.0.

[215] Thomas, M. and Thimbleby, H. (2018). Computer Bugs in Hospitals: A New Killer. *IT, Cybersecurity and Risk to Patients, Gresham College, Gresham College, available at:(accessed 26 February 2018)*.

[216] Thongtanunam, P., McIntosh, S., Hassan, A. E., and Iida, H. (2016). Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 1039–1050.

[217] Thongtanunam, P., McIntosh, S., Hassan, A. E., and Iida, H. (2017). Review Participation in Modern Code Review. *Empirical Software Engineering (EMSE)*, 22(2):768–817.

[218] Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., and Matsumoto, K.-i. (2015). Who Should Review My Code? A File Location-based Code-reviewer Recommendation Approach for Modern Code Review. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 141–150.

[219] Tian, Y., Nagappan, M., Lo, D., and Hassan, A. E. (2015). What Are the Characteristics of High-Rated Apps? A Case Study on Free Android Applications. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 301–310.

[220] Tosun, A. and Bener, A. (2009). Reducing False Alarms in Software Defect Prediction by Decision Threshold Optimization. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 477–480.

[221] Van Bouwel, J. and Weber, E. (2002). Remote Causes, Bad Explanations? *The Journal for the Theory of Social Behaviour*, 32:437–449.

[222] Wan, Z., Xia, X., Hassan, A. E., Lo, D., Yin, J., and Yang, X. (2018). Perceptions, expectations, and challenges in defect prediction. *Transactions on Software Engineering (TSE)*.

[223] Wang, S., Chollak, D., Movshovitz-Attias, D., and Tan, L. (2016a). Bugram: Bug Detection with N-gram Language Models. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 708–719.

[224] Wang, S., Liu, T., Nam, J., and Tan, L. (2018). Deep Semantic Feature Learning for Software Defect Prediction. *Transactions on Software Engineering (TSE)*.

[225] Wang, S., Liu, T., and Tan, L. (2016b). Automatically Learning Semantic Features for Defect Prediction. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 297–308.

[226] Wang, S. and Yao, X. (2013). Using Class Imbalance Learning for Software Defect Prediction. *Transactions on Reliability*, 62(2):434–443.

[227] Wang, T., Zhang, Z., Jing, X., and Zhang, L. (2016c). Multiple Kernel Ensemble Learning for Software Defect Prediction. *Automated Software Engineering*, 23(4):569–590.

[228] Wattanakriengkrai, S., Thongtanunam, P., Tantithamthavorn, C., Hata, H., and Matsumoto, K. (2020). Predicting Defective Lines Using a Model-Agnostic Technique. *IEEE Transactions on Software Engineering (TSE)*.

[229] Wilcoxon, F. (1992). Individual Comparisons by Ranking Methods. In *Breakthroughs in statistics*, pages 196–202.

[230] Wu, R., Zhang, H., Kim, S., and Cheung, S.-C. (2011). Relink: Recovering Links between Bugs and Changes. In *Proceedings of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 15–25.

[231] Xu, Z., Liu, J., Yang, Z., An, G., and Jia, X. (2016). The Impact of Feature Selection on Defect Prediction Performance: An Empirical Comparison. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, pages 309–320.

[232] Yang, Y., Zhou, Y., Liu, J., Zhao, Y., Lu, H., Xu, L., Xu, B., and Leung, H. (2016). Effort-Aware Just-In-Time Defect Prediction: Simple unsupervised models could be better than supervised models. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, pages 157–168.

[233] Yatish, S., Jiarpakdee, J., Thongtanunam, P., and Tantithamthavorn, C. (2019). Mining Software Defects: Should We Consider Affected Releases? In *In Proceedings of the International Conference on Software Engineering (ICSE)*, page To Appear.

[234] Zeller, A., Zimmermann, T., and Bird, C. (2011). Failure is a Four-Letter Word: A Parody in Empirical Research. In *Proceedings of the International Conference on Predictive Models in Software Engineering (PROMISE)*, pages 1–7.

[235] Zhang, F., Hassan, A. E., McIntosh, S., and Zou, Y. (2017). The Use of Summation to Aggregate Software Metrics Hinders the Performance of Defect Prediction Models. *Transactions on Software Engineering (TSE)*, 43(5):476–491.

[236] Zhang, H. (2009). An Investigation of the Relationships between Lines of Code and Defects. In *Proceedings of the International Conference on Software Maintenance (ICSME)*, pages 274–283.

[237] Zimmermann, T., Nagappan, N., Gall, H., Giger, E., and Murphy, B. (2009). Cross-project Defect Prediction. In *Proceedings of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 91–100.

[238] Zimmermann, T., Premraj, R., and Zeller, A. (2007). Predicting Defects for Eclipse. In *Proceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE)*, pages 9–19.

[239] Zimmermann, T., Zeller, A., Weissgerber, P., and Diehl, S. (2005). Mining Version Histories to Guide Software Changes. *Transactions on Software Engineering (TSE)*, 31(6):429–445.

[240] Zou, W., Lo, D., Chen, Z., Xia, X., Feng, Y., and Xu, B. (2018). How Practitioners Perceive Automated Bug Report Management Techniques. *Transactions on Software Engineering (TSE)*.