2018

Explanatory and Causality Analysis in Software Engineering

Yasser Ali Alshehri
yaalshehri@mix.wvu.edu

Explanatory and Causality Analysis in Software Engineering

Yasser Ali Alshehri

Dissertation submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Engineering

Katerina Goseva-Popstojanova, Ph.D., Chair
Hany H. Ammar, Ph.D.
Gianfranco Doretto, Ph.D.
Vinod Kulathumani, Ph.D.
Mario Perhinschi, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2018

Keywords: Software fault proneness, software development effort, explanatory model, case-control study, conditional logistic regression, ordinal logistic regression, causality regression, structural equation modelling

Abstract

Explanatory and Causality Analysis in Software Engineering

Yasser Ali Alshehri

Software fault proneness and software development efforts are two key areas of software engineering. Improving them will significantly reduce the cost and promote good planning and practice in developing and managing software projects. Traditionally, studies of software fault proneness and software development efforts were focused on analysis and prediction, which can help to answer questions like 'when' and 'where'. The focus of this dissertation is on explanatory and causality studies that address questions like 'why' and 'how'.

First, we applied a case-control study to explain software fault proneness. We found that Bugfixes (Prerelease bugs), Developers, Code Churn, and Age of a file are the main contributors to the Postrelease bugs in some of the open-source projects. In terms of the interactions, we found that Bugfixes and Developers reduced the risk of post release software faults. The explanatory models were tested for prediction and their performance was either comparable or better than the top-performing classifiers used in related studies. Our results indicate that software project practitioners should pay more attention to the prerelease bug fixing process and the number of Developers assigned, as well as their interaction. Also, they need to pay more attention to the new files (less than one year old) which contributed significantly more to Postrelease bugs more than old files.

Second, we built a model that explains and predicts multiple levels of software development effort and measured the effects of several metrics and their interactions using categorical regression models. The final models for the three data sets used were statistically fit, and performance was comparable to related studies. We found that project size, duration, the existence of any type of faults, the use of first- or second generation of programming languages, and team size significantly increased the software development effort. On the other side, the interactions between duration and defective project, and between duration and team size reduced the software development effort. These results suggest that software practitioners should pay extra attention to the time of the project and the team size assigned for every task because when they increased from a low to a higher level, they significantly increased the software development effort.

Third, a structural equation modeling method was applied for causality analysis of software fault proneness. The method combined statistical and regression analysis to find the direct and indirect causes for software faults using partial least square path modeling method. We found direct and indirect paths from measurement models that led to software postrelease bugs. Specifically, the highest direct effect came from the change request, while changing the code had a minor impact on software faults. The highest impact of the code change

resulted from the change requests (either for bug fixing or refactoring). Interestingly, the indirect impact from code characteristics to software fault proneness was higher than the direct impact. We found a similar level of direct and indirect impact from code characteristics to code change.

Acknowledgments

I start with the name of God and then my father and my mother, who were always with me sharing joys and watching my growth in this life. I grieve because they are no longer alive to see this special moment. I know how it would have made them feel.

The dissertation could not have been accomplished without the assistance, patience, and support of many individuals. First to be thanked is my great teacher and advisor Prof. Katerina Goseva-Popstojanova, who taught me so many aspects of research and what should strong research looks like. I learned from her immense knowledge how to be a great researcher, reviewer, teacher, and data scientist.

I offer gratitude to my family back home, to my inspiring brother Dr. Mohammed, my sister Hind, and my nieces and nephew. Many thanks are due to my grandmothers, uncles, and aunties for their love and good wishes and to my family in West Virginia, including my wife Aeshah and my little sweetheart Rawan, who was born in the first year of my Ph.D. journey. Special thanks go to my West Virginian family member Noble N. Nkwocha, his wife Karen Anthony, and their children for all their support and love.

I offer special thanks to my supervisor, Mr. Dale Dzielski from the Lane Department of Computer Science and Electrical Engineering (LCSEE). Throughout my work as a graduate service assistant, he always guided me to achieve the best. Many thanks are due to Dr. Brian Woerner, the chair of the LCSEE department, Dr. Muhammad Choudhry, the chair of the Computer Engineering program, and Dr. Bojan Cukic, the chair of the Computer Science department at the University of North Carolina at Charlotte for their support.

I would also like to extend my thanks to Dr. Hany Ammar, Dr. Gianfranco Doretto, and Dr. Vinod K. Kulathumani; I learned a great deal from them through their courses Advanced Real-Time Systems, Machine Learning, and Sensor Networks. Many thanks to them and to Dr. Mario Perhinschi for agreeing to serve on my PhD committee and guiding my research with their valuable comments and suggestions.

My gratitude extends to my friends in the lab who were very supportive. Special thanks go to Thomas Devine for his support and for providing me with the extracted dataset metrics of Eclipse releases. I would like to thank Mohammad J. Ahmad for providing me with the extracted metrics of the Apache projects. Special thanks are also extended to Jary Hernandez and my other fellows in the lab, Thomas Kyanko, Di Pang, and Jacob Tyo.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation offers four major contributions to two areas of software engineering. The two areas of study are software fault proneness and software development efforts. Most of the related works are focused on analysis and prediction. This area lacks of explanatory and causality studies. This area also lacks systemic modeling techniques that can overcome challenges associated with the nature of the software project data and provide explanatory models to fit the data. The contributions aim to provide an explanatory approach using methods that (1) consider main confounders[1] and their interactions results in models that (2) are not affected by multicollinearity, and (3) and are statistically fit for the data. So, models can explain how different confounders and their interactions are affecting both software fault proneness and software development effort. This can help software practitioners to improve their practice in software development and testing, and in estimating the development effort, which can significantly help to improve quality of software products and reduce the extra cost that results from the inaccurate effort estimation or software faults.

[1]In this dissertation, depending on the specific approach used, we use different terms for independent variables, such as metrics, exposure, confounders, and features.

The contributions to these two areas include (1) an explanatory approach using a matched case-control method in the software fault proneness area; (2) an explanatory approach using a categorical regression method for modeling confounders and interactions of confounders in the software development efforts area; and (3) use of the structural equation modeling SEM as a causality modeling technique to explain software fault proneness and direct and indirect effects of different metrics on software faults.

The first contribution of this work is to establish a methodology that can build a well-fitting explanatory model to explain the main contributors of software faults from confounders (i.e., metrics) and interactions of confounders. The methodology is based on a case-control study that accounts for exposure and matching between files in terms of their size. Matching between files can potentially tighten the range of the upper and lower values around the odds (i.e., the probability of success over the probability of failure). We tested conditional logistic regression (CLR) algorithm to allow for matching between files based on their size when the model was built. Further, interactions in the model gave more insight and new explanations that have not been discussed in similar works [3, 4, 5, 6, 7, 8, 9]. The selection of the initial set of confounders was based on the pair-wise correlation coefficients between all pairs of confounders. We divided the data based on the postrelease bugs status: (1) cases groups are files with one postrelease bug or more and (2) controls group are files with no postrelease bugs. Then, sampling was made for the cases group, followed by matching with files with the same size from the controls group. Then, the initial model was tested for multicollinearity. Lastly, the backward elimination modeling was applied because of the existence of interactions between confounders. The final model was measured for the goodness of fit. We built a total of twelve models from twelve releases of Eclipse and Apache projects. All final models were fit to the data and reported some consistent results across all projects as discussed in more details in Chapter 3.

Second, we measured the prediction performance of the models built for explanatory purposes. We used the same samples and the same models with their confounders and interactions for this purpose. To evaluate the performance of the conditional logistic regression (CLR), we needed to compare the performance with other commonly used classifiers as benchmark. However, for the other classifiers, we were not able to use the interactions of

the confounders. Therefore, we used all the set of the change metrics for all other classifiers. We also applied a new algorithm that accounts for confounders selection and shrinkage by minimizing insignificant confounders' coefficients to zeros and relied on the remaining metrics for prediction. This process can automate the manual selection of confounders that was applied earlier and provide good predictions. The prediction performance of this algorithm was compared with the performance of five top-performing classifiers in the area (i.e., logistic regression, naive Bayes, decision tree J48, random forest, and decision list PART). The work of this part is covered in detail in Chapter 4.

Third, we applied an explanatory study in the area of software development effort. This area also lacks systemic modeling techniques that can overcome challenges associated with the nature of the software project data and provide explanatory models to fit the data. Software development effort, measured in man-hours, was discretized to ordinal format with other selected independent confounders. The independent confounders were chosen based on the early findings and popularity of the confounders in the related works. Other considerations were made to treat missing data, study the association between independent metrics, and test models for multicollinearity during the process of building the final models. Insignificant interactions and metrics were eliminated to leave the model with only significant terms (i.e., confounders and their interactions). The final models were tested for goodness of fit and the ability of the final models to predict different classes for software development effort were also considered using the most popular performance metrics. The findings regarding the main confounders and interactions can be used by the software project managers and practitioners to help to produce software products with less challenge concerning development effort. The proposed methodology was applied to three open and private data sets, which discussed in more detail in Chapter 5.

Last, we explored applying the causality concept in the software fault proneness area. Causality is important because there is a clear distinction between correlation and causation. We conducted a causality using one of the popular methods, structural equation modeling (SEM) [10, 11]. We built our own methodology based on several aspects from earlier works. The methodology leads to a final graphical causal model, which includes significant con-

founders and underlying variables (i.e., latent variables) that cause the postrelease faults. The final model was achieved after statistical analysis, model specification, estimation, and model validation. The methodology of this work and the case study using the Eclipse project are explained in Chapter 6.

The rest of this dissertation is organized as follows. A highlight on related studies from all the areas, which includes (1) explanatory and prediction of software fault proneness, (2) explanatory and prediction of software development effort, and (3) causality studies are discussed in Chapter 2. In Chapter 3 we discuss the motivation, methodology, case studies, and results of explanatory work proposed by this study using a case-control method. The prediction of explanatory models and the comparison of prediction performance with other classifiers including the Group Lasso regression algorithm are explained in Chapter 4. The explanatory and prediction studies for the software development effort are discussed in Chapter 5, including the methodology and the case study of the three open and private projects datasets. The causality study of the software fault proneness area is described in Chapter 6. The threats to validity and future directions of research are discussed at the end of each chapter. Last, the dissertation is concluded in Chapter 7.

Chapter 2

Related Works

This chapter highlights related studies of the two areas: software fault proneness and software development efforts. The chapter focuses on studies that used explanatory approaches in the two areas. Further, it highlights some of the recent studies that applied different approaches of predictions, including the performance they achieve from applying prediction. Additionally, the chapter covers related studies that used the causality approach in software engineering and in other fields. This chapter is organized as follows. The related works on software fault proneness are discussed in Section 2.1. Then, the related works of explanatory and prediction in software development efforts are discussed in Section 2.2. Last, the related works of causality in software engineering and in other fields are explained in Section 2.3. At the end of every section, we describe the novelty our work is adding to the specific area.

2.1 Related Works on Software Fault Proneness

The studies focused on software fault proneness can be classified into three main categories: 1) studies focused on analysis [12, 13, 14, 15, 16, 17, 18, 19]; 2) studies focused on prediction [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]; and 3) studies focused on explanatory methods to identify impacts of metrics on the software fault proneness [3, 4, 5, 6, 7, 8, 9].

2.1.1 Analysis studies on software fault proneness

The first category is based on statistical and quantitative analysis to find relations between different metrics and software faults [12, 13, 14, 15, 16, 17, 18, 19, 32, 33]. Three studies found that LOC are not associated with Postrelease bugs [12, 13, 14]. Other studies found that LOC is negatively correlated with fault density [32, 33]. Fenton and Ohlesson [12] did not find any relation between prerelease and Postrelease bugs using a dataset from Ericsson Telecom. The study found that 72% to 94% of prerelease faults were detected in files with no Postrelease faults. On a contrary, the two replicated studies [13, 14] found the Prerelease bugs and Postrelease bugs are associated. The study [16] found that prerelease and Postrelease faults are positively correlated. Other studies [17, 19] investigated where faults are localized and a common source of failure. Two studies [30, 31] found that 75% of software faults were localized in 20% of files. Misirli et al. [18] used the Spearman test to explain the correlation between change metrics and faults of Eclipse 2.1 and 3.0. The study found a high correlation between age (old files) and Prerelease faults. Additionally, the study found a high correlation between number of revisions and the Postrelease faults. A high correlation was found between the number of developers and Postrelease faults. Another statistical test (Mann Whitney-U) was used to examine the differences between two groups of files (failure prone, and non failure prone) [15]. Code churn and complexity of the code metrics were found to be statistically significantly different between the two groups. The two metrics were highly associated with software faults on the Eclipse dataset and had a low association in Microsoft Windows Vista. Other metrics were consistent in both systems.

2.1.2 Software fault proneness prediction

The second category of related works dealt with predicting software fault proneness [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]. Predictions were made using mainly two types of modeling: numerical prediction or classification. Models used regression results in numerical values for number of faults in a software unit. At the same time, the output from classification models can tell us if the software unit is faulty or fault free.

Hall et al. [34] reviewed 208 papers in software fault proneness and investigated the algorithms applied, metrics applied, and datasets used. In terms of the algorithms, LR algorithm was used most often (in 40% of papers), followed by the NB algorithm, used in 25%. Other algorithms such as J48, neural networks, and RF were each used by around 8% of papers [34]. Object-oriented metrics were used in 20%, followed by the static code metrics used in 18%. Change metrics, a combination of static code (8%) and change metrics (8%), LOC (8%), and source code (5%), came next. Regarding datasets, 50% of papers used the Eclipse project, 13% used telecom project(s), 12% used Mozilla, and only 1% of papers used Apache projects.

Analysis of specific metrics and their role in improving prediction performance was conducted by [28, 29, 25, 35, 36]. Older files (more than a year in age) had fewer faults than new files [28]. Another study [35] found that the number of developers has no impact on software faults. Ostrand et al. [36] developed a prediction model that considered the developers metric to find a number of faults related to every developer. Another two studies found that using the summation of added lines and deleted lines as a code churn metric improved prediction performance [29, 25].

Schröter et al. [20] encouraged to work in mining static metrics, change metrics, and human-related metrics and used them to predict software faults. Zimmermann et al.[23] built classification models for Eclipse releases 2.0, 2.1, and 3.0 and reported precision, recall, and accuracy of all classification models using both file and package levels. The reported accuracy was between 80% and 90%. At the file level, the recall values were very low, not exceeding 30%, and the precision values were between 47% and 72% at the package level. Catal et al. [37] investigated the effects of dataset size, metrics set, and feature selection

techniques on software fault predictions, using NASA datasets from the PROMISE repository and applying nine machine learning algorithms (e.g., J48, RF, NB). In terms of the AUC, RF outperformed other classifiers when used with large datasets, and NB performed the highest when used with small datasets.

Moser et al. [21] investigated whether change metrics provide better prediction than static code metrics. The authors applied several machine learning algorithms to Eclipse releases 2.0, 2.1, and 3.0 and found the change metrics provided better performance than static code metrics on all datasets for all algorithms. Krishnan et al. [22] used change metrics, the same Eclipse releases as [21] and another four releases (Europa, Ganymede, Helios, and Galileo). The predictions were done at file level using the J48 algorithm. In a follow-up work, Krishnan et al. [38] investigated whether the predictions of software fault proneness improved as the Eclipse product line evolved. The study found that there was no statistically significant difference between the AUC and the true positive rate (TPR) of the top algorithms, including J48. Therefore, J48 was used, and its performance on releases 2.0, 2.1, and 3.0 was compared with the previous works [23, 21].

Gue et al. [39] analyzed the performance of RF with other machine learning algorithms (e.g., logistic regression, J48, Naive Bayes, and random forest). The study used five datasets from NASA project, applied different types of features (e.g., McCabe complexity, line count, Halstead, branches count), and measured the performance using specificity, sensitivity, and the probability of false positive. The random forest machine learning algorithm outperformed all other machine learning algorithms. Lessmann et al. [40] proposed a framework for bench-marking classification models for software fault proneness. The study applied 22 machine learning algorithms on ten public datasets from the NASA Metrics Data repository. The main finding of the study was that there were no significant differences among the performance of the top 17 algorithms (out of the 22 algorithms used) in terms of the AUC values. Another benchmark study [41] that focused only on Bayesian networks machine learning algorithms applied 15 BN machine learning algorithms to predict software faults proneness using 11 NASA MDP datasets [41]. The study compared the performance of

all machine learning algorithms using the ROC curve and H-measure metric. The study found that augmented naive Bayes machine learning algorithms perform as well as or better than naive Bayes machine learning algorithm. Another study [42] found that decision tree regression performed better than other types of regression.

More recent works applied and proposed different methods for software fault prediction, such as Bayesian networks [41, 43], deep learning [44], semi-supervised deep fuzzy clustering [45], faults prediction after reducing irrelevant, redundant features, and using reliable features [46, 47], back propagation neural networks [48], combining genetic algorithms with deep neural network DNN in [49] and with back-propagation learning algorithm in [50], multiple kernel ensemble learning [51], and non-negative sparse graph-based label propagation [52].

G-Lasso is an extension of the linear lasso regression that can be used for binary classification. Linear lasso regression and G-Lasso have been employed in different areas such as medicine [53, 54, 55, 56], image processing [57, 58, 59], and finance [60]. The results of using the linear lasso and G-Lasso in these other areas have been promising, which motivated us to use the algorithm for software fault proneness prediction. To the best of our knowledge, G-Lasso has not been used neither in software engineering in general nor for prediction of software fault proneness in particular. Security classification was the closest area to ours and used G-Lasso [61]. In [61], the authors applied lasso for binary and multiclassification and found that it performed better than SVM and k-NN algorithms. The overall accuracy achieved by the binary classification with lasso was 78%, the recall was 89%, and the precision was 79%.

In Chapter 4, we used CLR and G-Lasso algorithms for classification for the first time in the area of software fault proneness prediction, and compared their performance to the performance of five widely classifiers. We also explored performance at the data sets used. Our work generalized the related work that did benchmarking analysis of different machine learning algorithms [40]. The approach and comparison of all results with related studies are covered in details in Chapter 4.

2.1.3 Explanatory studies on software fault proneness

With respect to the explanatory studies, they aim to explain why things happen and quantify the contribution of every confounder on software faults. This kind of studies has been used widely in medical research to find how factors (such as smoking, or eating habits) contribute to certain diseases, like cancer or heart problems [62, 63]. Several related works exist in the area of software fault proneness [3, 4, 5, 6, 7, 8, 9].

Cataldo et al [3] investigated two areas of dependencies: logical dependencies and high levels of work. The analysis involved measuring for VIF, to test for multicollinearity, and a pairwise correlation test to test for the correlation between two different variables. The model was built in a forward fashion, and the metrics were excluded based on the VIF values. The initial model contained only the dependent variable and the intercept. Then, one metric at a time was added to the initial model until the final model was reached. To measure the impact of this metric, χ^2, $\Delta\chi^2$ were calculated along with the deviance percentage to measure the goodness of fit of every model.

Following the same approach, another explanatory study by Shihab et al [5] was conducted to find what confounders affect Postrelease faults. Using logistic regression, the study started with 34 metrics from static code and change confounders and ended up with four confounders that showed significant impact on Postrelease faults. Total lines of code and prerelease bugs were found to be consistent in Eclipse 2.0, 2.1, and 3.0. Other confounders were found to be highly significant in a single release such as the number of parameters, number of static methods, and anonymous type declaration. Additionally, the study measured the performance of the reduced models and compare them with the models that used all metrics. Recall and precision values were comparable between models with the complete set of confounders and reduced models. Recall values ranged between 15.8 to 32.4% and precisions ranged between 58.6 to 66.3%.

Bettenburg and Hassan [6, 4] explored human factors of developers and their impact on software fault proneness using a similar approach. They considered confounders that included in the content of messages exchanged between developers such as the amount of source code, amount of patches, the amount of stack traces, and the number of URL links.

Other factors related to the social structure, dynamics of communications, and workflow measures were also considered. Both studies applied variance inflation factor VIF values to test the multicollinearity by allowing variables with VIF values less than 10. The main findings were: source code in the content increases faults by 67%, the number of patches in the content increases software faults by 18 times, and consistent flow of the information decreases software faults.

Human factor was also chosen as an area for an explanatory work in [7]. The study used Openstack and Eclipse datasets to explain the relationship between human discussion and software faults. Confounders in this research included the length of comments, number of comments, time for discussion and experience of developers involved in the change process. Experience of developers are found to reduce the risk for software faults. Positive words between developers was considered in the model as a confounder but did not show any significant impact on software faults. The model was built using the same method used in [3] by applying logistic regression and diagnose multicollinearity using VIF values. The model is validated using 10-k cross-validation and reported recall (ranged between 0.59-0.74), precision (ranged between 0.37-0.82), the area under curve AUC (ranged between 0.56-0.71).

Explaining the software fault proneness on mobile applications was investigated in [8]. The selection of mobile applications of the study is based on their popularity, being open-source, simplicity, and having a large code base. The source code confounders used in the study were lines of code, coupling, cohesion, and platform (e.g., Android). The study applied logistic regression to explain confounders and they applied VIF to asses multicollinearity and eliminate confounders with VIF higher than 5. The results indicate that both platform and coupling are statistically significant in most projects. The main finding was that cohesion and coupling have the greatest impact to explain software fault-proneness.

Explaining log-related confounders (e.g., log density, logging level, and log lines) was an area of study in [9]. The study applied logistic regression on Hadoop and JBoss software projects to explain confounders that cause Postrelease faults. The main finding of the study is that the log-related confounders improve the explanatory power by 40% along with the static code and change confounders.

Many empirical research works have been focused on software fault proneness. They were, with only a few exceptions, descriptive and predictive in nature. Therefore, the confounders that affect software fault proneness are still not well understood. Some of the reasons for the lack of understanding are (1) relying on the one confounder at a time analysis method and not considering the multicollinearity of several confounders, (2) no exploring the interactions between confounders, and (3) relying on predictive models to conduct explanatory studies.

In our work, the selection of confounders is based on earlier findings and the results of the pairwise correlation to eliminate high collinear confounders. We add the interaction of the confounders in the model. Multicollinearity diagnosis test is used to test the initial model which contained confounders and interactions. Then, the final model is achieved based on a backward fashion, starting with the highest order term (interaction). A goodness of fit model was applied to confirm that the elimination of each interaction does not affect the reduced model. Another goodness of fit test was also applied to the final model, which was not done in other previous related works. The motivation and the methodology of this work are covered in details in Chapter 3.

2.2 Explanatory Work on Software Development Efforts

Earlier research on prediction of software development efforts was based on three approaches. The first approach was based on expert judgment in estimation [64, 65, 66, 67]. The purposes of that research was to build accurate models based on expert judgment and compare them with mathematical and machine learning models. The second approach was to use a mathematical model to estimate the effort [68, 69, 70, 71, 72]. The most common mathematical model is the COCOMO model [72], which uses the size of a project (measured in thousands of coded lines) as the main parameter to estimate effort. The third approach

was based on modeling techniques and used either analogy-based models (e.g., [73, 74, 75]) or machine-learning models (e.g., [76, 77, 78, 79]). Most of the research used the third approach because of its simplicity and low cost [80], specifically when applying several machine learning methods to improve estimation.

Because our work falls into the third category, we focus on the related studies that applied predictions to obtain an accurate estimation. Further, because we included interactions of metrics in our model, we highlight some related studies that involved interactions for prediction.

Angelis et al. [81] applied OLS on numerical and categorical data using ISBSG metrics. Missing values were treated by elimination the complete row (i.e., list-wise deletion). They used three numerical metrics: function points (Size), efforts, and max team size. The study transformed data to a logarithmic format to obtain normal distribution and implement OLS regression. The study used development-type and language-type metrics in a categorical format. The study applied a correlation test using χ^2 values, and highly correlated metrics were eliminated. Briand et al [82] used OLS, ANOVA, CART, and analogy-based approaches as well as a combination of methods (CART with OLS and CART with analogy). They used the Laturi data set, which consists of 206 projects collected from companies from Finland. They found that size, organization type, and target platform are the top prediction metrics, and that the CART model performed better than other approaches for local or cross-company projects. A replicated assessment work [83] used the same approaches that were used in [82] with a multi organization software project data set. Size and maximum team size were the two major overriding variables in all models. The results of the replicated work showed that CART performed relatively poorly and the OSL model was sufficient to predict the effort. OLS was also used in several other studies [84, 85, 64, 86].

Several other studies applied machine learning algorithms to predict efforts [87, 78, 88, 89, 76, 90, 91, 92, 77, 93]. Some studies proposed or applied single method of prediction, and other applied several methods, ensemble methods, or methods applied on fuzzy decision tree.

Srinivasan and Fisher [87] applied regression tree and neural networks on COCOMO and Kemerer's data sets and compared their accuracy with different arithmetical models such as COCOMO and Software Lifecycle Management SLIM. Kocaguneli et al. [75] proposed an analogy-based method that outperformed linear regression and neural network estimators. They applied the same method for the transfer learning methodology to estimate efforts. Kumari and Pushkar [91] proposed cuckoo Search algorithm and hybridizes it with neural networks to improve the prediction of effort estimation. An estimation method based on fuzzy logic was proposed in [94], and it shewed a significant improvement in prediction compared to other studies. Sarro et al. [95] a genetic programming method on the Desharnais, Finnish, and Miyazaki data sets, and they measured their performances using MMRE, PRED(25), and MdMRE. The best results achieved for MMRE, MdMRE, and PRED(25) were 0.51 in Miyazaki, 0.43 in Desharnais, and 0.32 in Miyazaki and Desharnais. A causal discovery algorithm using a PC search algorithm was proposed in [79] to predict direct, indirect, and bi-directed edges between software efforts metrics. Sigweni et al. [96] applied leave-one out cross validation method based on chronological orders of software projects (i.e., grow one at a time technique) and not based on random selection of projects for Desharnais and Finnish data set. The study found that the proposed technique is more realistic than the traditional method.

Baskeles et al. [89] applied neural networks, regression trees, and support vector regression for the NASA and Turkish industry data sets. Li et al. [90] used Neural networks and regression trees on Desharnais and Maxwell data sets. Radinski and Hoffmann [76] applied twenty-three machine learning methods (e.g., Bayes, lazy, rules, and trees) on four datasets (i.e., COCOMO, Desharnais, Maxwell, and QQDefects) to predict software development effort. Andreou and Papatheocharous [78] used fuzzy decision trees to predict cost on the ISBSG data set. Huang et al. [88] proposed a model based on fuzzy decision trees for software cost prediction for the COCOMO dataset. Kocaguneli et al. [97] applied ensemble methods combining multiple learners with multiple preprocessing method and found that CART regression ranked at the top. Elish [93] applied ensemble learning method using five classifiers on five data sets. The result of the ensemble method outperformed the performance of individual classifiers. Another ensemble method was developed by [98] and the study found that

applying principle component analysis with the CART regression was ranked at the top of all other methods. Several machine learning methods (i.e., k-NN, support vector regression, multilayer perceptron, and decision trees) were applied in [92]. Nassif et al. [77] applied four types of neural networks (i.e., multilayer perceptron, general regression neural network, radial basis function neural network, and cascade correlation neural network) on ISBSG data set. Jodpimai et al. [99] applied five data preprocessing techniques with five learning techniques (i.e., regression analysis, support vector regression, classification and regression tree, k-NN, and radial basis function). The study did not find a dominant learning that outperformed all other algorithms.

In the area of software effort estimation, ordinal regression has been used once by [100]. In this study, the models were built for prediction and used on three data sets: Maxwell, COCOMO81, and ISBSG. The study used both categorical and numerical independent metrics. The authors log transformed the size and duration metrics to gain normality in the distribution. The response confounder, effort, was discretized to four levels in all data sets using the equal frequency method. The learning model was developed using forty-two projects to test ten projects. The authors evaluated the model using MMRE, PRED(25), hit rate, and correct classification.

Some studies have considered the effect of metrics when they interact with each other. Interactions were considered with a goal to improve the prediction. Three studies have used interactions in prediction models [101, 102, 103].

Gray et al. [103] used logistic regression for three response metrics related to the overestimation, underestimation, and error estimation. For every model, the study developed all possible scenarios including interactions of metrics. The goal of adding interaction was to get the best fit model according to the Akaike information criterion (AIC). For all the scenarios involving the three models, interactions did not improve the fit. Tsunoda et al. [101] aimed to determine whether interactions can change accuracy by comparing them to models without interactions. The study emphasized the necessity of using interactions to improve prediction. However, not enough evidence was shown to prove the case. A slight

improvement in performance occurred with NASA data set when the model used interactions. Tsunoda et al. [102] investigated the role of moving windows in estimating efforts. This involved interactions between size and timing (the age of the project). The results of the interaction showed a slight improvement in the evaluation criteria for some cases.

Missing values are very common in software development estimation. Next, we highlight the major treatments applied by studies in this field and their main findings. Strike et al. [85] evaluated the use of several techniques: listwise deletion, mean imputation, and eight different types of hot-deck imputation. The study found that hot-deck imputation perform consistently with the highest precision and least bias compared to other techniques. k-NN achieved better performance than toleration and mean imputation techniques in [104]. Another study [105] suggested the use of multinomial logistic regression over listwise deletion, mean imputation, regression imputation, and expectation maximization for missing data treatment with ISBSG projects. Song et al. [106] found k-NN to improve prediction models built using the C4.5 algorithm. The study also found that the percentage of the missing data should not exceed 40%. Idri et al. [107] studied three techniques of missing data treatment (i.e., toleration, deletion, and k-NN) using two types of analogy models. The study found that using k-NN can improve the performance of the models more than using toleration and deletion.

Our proposed approach (in Chapter 5) shares a similar goal, which explains how specific metrics and interactions contribute to the software effort. The involvement of the interactions should add more explanations from the model unlike related works. In addition, we use a holistic approach that includes a test for correlation, discretization of numerical data, handling of missing values, multicollinearity, elimination of insignificant interactions and metrics, and goodness of fit. The methodology and implementation of the three case studies of different datasets are explained in details in Chapter 5.

2.3 Related Works on Causality

Structural equation modeling (SEM) has not been used in the field of software engineering or in any closely related areas. However, this method has been used in many other areas such as psychology (e.g., [11, 108, 109]), education (e.g., [110]), biology (e.g., [111]), neuroscience (e.g., [112]), ecologic studies (e.g., [113]), accounting (e.g., [114]), marketing (e.g., [115, 116]), hospitality management (e.g., [117]), operations management (e.g., [118]), management information systems (e.g., [119]), and strategic management (e.g., [120]).

In [11], the SEM methodology was described as a guide for researches in psychology. Martens [108] reviewed 99 papers published in the *Journal of Counseling Psychology* between 1987 and 2003. The study analyzed how researchers handled issues such as normality and goodness of fit. For example, in nineteen percent of studies the authors discussed the normality of their data. The goodness of fit test using χ^2 was the most common approach, reported by ninety percent of the studies. A comparative fit index (CFI) was used by sixty percent of the studies, the Tucker-Lewis index (TLI) was used by forty-three percent of the studies, and the root mean square error of approximation (RMSEA) was used by thirty-eight percent of the studies. In the same domain, an earlier study [109] reviewed seventy-two papers between 1977 and 1987.

Hair et al. [120] analyzed more than a hundred papers published in top-ranked journals between 2000 to 2011 in the field of strategic management. The range of the number independent variables used in the surveyed papers was from seven to 114. The range of the number of latent variables used in these papers was from two to thirty-one. The number of variables per latent variable ranged from one to ten, with a median equal to three variables. This indicates that more than fifty percent of the studies used no fewer than three variables per latent variable, which is the recommended number.

Hair et al. [116] analyzed 204 studies published in the last thirty years in top-ranked journals in the marketing field. The range of the number of latent variables was from two to twenty-nine, with a median of seven. The number of variables per latent variable ranged from one to twenty-seven, with a median of four. The total number of variables used in the sample of papers ranged from four to 131, with a median of twenty-four. Henseler et

al. [115] addressed the specific requirements and typical research problems of international marketing using path modeling techniques. In the business intelligence field, Jakli et al. [121] used partial least squares (PLS) in an SEM model to analyze the interrelated role of compatibility in predicting business intelligence and analytics, finding that compatibility perceptions have a direct positive impact on use intention.

Kaufmann and Gaeckler [122] analyzed seventy-five papers published in top-ranked journals between 2002 and 2013 in the field of supply chain management. The number of latent variables in analyzed studies ranged from three to twenty-one, with a median of six. The number of structural model relations (i.e., relations between latent variables) ranged from three to twenty-five with a median of eight.

Ringle et al. [119] surveyed 109 papers from journals on information technology and management information system. The number of latent variables used in these studies went from three to thirty-six, with a median of seven. The number of structural models relations ranged from two and sixty-four, with a median of eight. The median number of variables used with every latent variable was three. The total number of independent variables ranged from five to 1,064, with a median of twenty-six independent variables.

Ali et al. [117] reviewed hospitality management journals published from 2001 to 2015. A total of twenty-nine papers were reviewed in this study. The sample sizes used in reviewed studies ranged from 106 to 1,500 observations. The number of latent variables ranged from three to twenty, with a median of seven. The number of structural paths ranged from three to twenty-four, with a median of six. The total number of independent variables ranged from twelve to seventy-eight, with a median of twenty-two.

Peng and Lai [118] reviewed forty-two papers from the top eight journals in operations management field. The sample size for all papers ranged from thirty-five to 3,926 observations. In the management field, Kulikowski [123] applied SEM and found a direct relationship between pay for individual performance and work engagement. Additionally, the authors found an indirect relationship between the two factors through pay satisfaction.

Causal modeling using SEM has not been used in the software engineering field. However, some related works used BN for prediction, network structure, and decision-making [124, 125, 126, 127, 43, 128, 129]. BN are famous of their ability to predict causal relationships for continuous and discrete variables. Here, we discuss the studies in the field of software fault proneness prediction, which was also discussed earlier.

Fenton et al. [124] reviewed methods used for software fault proneness prediction and suggested using the BN method. The authors claimed that software faults are influenced by other factors that are not related to code complexity, such as the difficulty of the problem and analyst skill. In addition, using causal models could help understanding software faults at every stage of the software development because the cost of defects differ at every stage. The researchers also introduced a prototype model to explain how BN prediction can be implemented. The study built a model with two stages of software development: the first stage covers specification, design, and coding, and second stage covers the testing phase. Another work by Fenton et al. [125] encouraged the use of BN to predict models to support effective risk management and decisions. The researchers claimed that reliance on regression model is not enough to explore all causal effects on software quality. In a follow up work, Fenton et al. [126] used BN as a general approach that can be applied to any lifecycle, which helped for decision-making purposes. The method was built to overcome the limitation of earlier work [124, 125] by avoiding a separate model for each development lifecycle. The general model can detect specification, development defects, and testing defects. Fenton et al. [128] used BN to predict software defects and software reliability. The study emphasized that any model using defects in one phase to predict defect at subsequent phase should incorporate causal factors such as testing and quality levels. Therefore, the study applied BN to predict post-release defects and reliability, taking into account the design process quality and testing quality levels. The study [128] also used dynamic discretization approach to improve the accuracy of software defect prediction.

Wagner [130] transferred the activity based quality models (ABQM) [131] to a BN model. The main aim of the model was to provide quality managers with a systematic method to conduct assessment and prediction. The study proposed a four-step approach for transferring activity-based quality models to BN network. The four steps can be summarized as follows:

(1) define the activity that was intended to be measured (e.g., maintenance), (2) define facts and activities that are related, (3) add additional variables related to facts and activities, (4) apply BN to predict the outcome of the model. The study pointed out that it is necessary to answer questions like what variable is more important than others, which we trying to address in this work and from our previous attempts of the explanatory work.

Other studies used BN for software fault proneness and reliability prediction [127, 43, 128, 129]. Van Koten and Gary [129] constructed a BN model using object-oriented metrics for software maintainability prediction. The model was compared with other regression tree and multiple regression models, and used absolute residual, magnitude of relative error MRE, and pred measures to compare performance of all models. The study found BN model outperformed the regression tree and multiple regression models. Pai et al. [127] realized the importance of validating relationship of performance measures with external quality metrics. The study built a model using two response variables (i.e., fault proneness and fault content) and seven independent variables from the object-oriented static code metrics [127]. The study did not consider direct effect between independent variables and the response variables, and did not consider the indirect effect. Okutan and Olcay [43] used object-oriented metrics on Promise data repository to predict software fault proneness. The study built a causal model for every data set used based on score system of association between every pair of metrics. The study eliminated metric based on the score result, which suggested no association of the eliminated metrics with any other metric. The study showed some of the direct and indirect effects between some of the object-oriented metrics used. The study did not include change metrics which would increase more complexity to each model. The study also did not consider the high correlation and multicollinearity of the model.

A benchmark analysis study [40] was conducted using twenty-two classifiers on ten NASA MDP datasets. The performance of BN in the study was among that of the top six classifiers in terms of the area under the curve (AUC). BN followed random forest, neural networks MLP1 and MLP2, and support vector machines LS-SVM and L-SVM. The performance of BN had no statistically significant differences from those of higher-ranked classifiers. Another benchmark study [41], which focused only on BN classifiers, applied fifteen BN classifiers

to predict software fault proneness using eleven NASA MDP data sets [41]. The study compared the performance of all classifiers using the ROC curve and H-measure metrics. They found that augmented naive Bayes classifiers performed similarly or better than naive Bayes classifier.

Some studies have combined SEM with other approaches to replace the estimation from the traditional ML to BN or to take advantage of the prediction ability of BN algorithms (e.g., [132]). Further, some studies have applied neural network prediction on SEM networks. To use neural networks, the causal network should consist only of measured variables. In other words, latent variables are not allowed in neural networks. Therefore, the SEM should not contain any latent variable, to be applicable for neural network prediction.

SEM and the Gibbs sampling using the Markov chain Monte Carlo (MCMC) method was applied in [133]. The study estimated the parameters using the Gibbs sampling instead of the maximum likelihood ML, the estimation method that is normally used with SEM. The integration of SEM and BN appeared in [132]. The study proposed a causal model for management decision support that links SEM and BN, with a goal to examine the factors that cause customer retention. The prediction accuracy performance gained from the BN models varied from seventy-four to ninety-three percent. Another study [113] linked SEM and BN to explore the effects of environmental factors on freshwater macroinvertebrates. The causal network was built using the SEM concept, and BN was used for prediction and decision-making. Another attempt to combine SEM with prediction via neural networks was made in [134]. The SEM diagram of the model included only measured variables (i.e., no latent variables were included) and one response variable, which measured consumer intention to adopt mobile commerce. Therefore, using the same model for neural network algorithms was explored for prediction. A similar approach was employed by [135] to measure the adoption of inter-organizational systems. Another similar approach was implemented by [136] to measure the customer satisfaction and loyalty with respect to airline services.

In our study, we used software fault proneness case study, which had thirty variables from static code and change variables. Moreover, these variables were extracted at a single point of the development cycle. Therefore, building a network using BN methodology would result in a very complex model and very hard to explain. Our focus is to build an explanatory causal

model which can be fulfilled by using structural equation modeling. Thus, we used only the statistical methods and regression analysis associated with SEM methodology, data analysis, sampling, variable selection, latent variable creation, model specification, model estimation, and model validation. Latent variables gave us an advantage to simplify the model and reduce the number of paths that would result without using them. Also, we considered the high correlation of selected variables to ensure that under every model, no high correlation is detect between any pair of variables. Further, we analyzed the multicollinearity of the linear model using all variables. The detailed methodology, and the case study are explained in Sections 6.2 and 6.3, respectively.

Chapter 3

Using a Case-control Study to Explain Software Fault Proneness

This chapter implements an explanatory work to explain software fault proneness. For that, several steps are established to build a model that fit the data and to explain confounders that contribute to the software faults. In this chapter, a matched study using case-control methodology is suggested to build the explanatory model. Further, the models include interactions between confounders to assess how they add to the model with the main confounders. The main motivation for this work is that there is a need to use a proper method to measure the effect of the main confounders and their interactions, consider the stratification in the analysis, consider the multicollinearity of the model, and eliminate the interactions and confounders that do not statistically harm the model. The proposed methodology is explained in details in Section 3.2, after highlighting on the contribution of this work in Sections 3.1. First, we used Eclipse's Europa as a case study, and then the work was replicated using Ganymede in Section 3.3. The work was also replicated on projects from the Apache software foundation (i.e., Derby, Ant, and Xalan), which is presented in Section 3.4. We provided explanation of the results of all projects in Section 3.5. At the end, the threats of validity of this work are discussed in Section 3.6, and the chapter is concluded in Section 3.7.

3.1 Background and Motivation

Software developers aim to write programs that run without failure [1] because the reliance on software systems has become essential in all businesses and individuals activities. It is important for software programs to function properly and meet the quality requirements, in addition to the end users requirements. However, the process of software development is prone to human errors. Therefore, software faults [2] can occur at any stage of the software life cycle. The main goal for the software engineers is to address where these faults have occurred and why they occur and prevent them from happening in the future. Faults may not affect the system as long as they have not been executed. However, they may become executable at any point in the life of the software, which leads to failure. Treating faults early will produce a functioning software as it was decided at the requirement stage and reduce the extra cost spent in repairing these faults in the future.

Many open-source projects are publicly available online with multiple releases and high amount of data can be extracted from source files. Moreover, bug repositories are also available to trace changes of a software unit (e.g., class or file). All that can be great source for analysis and learning the most efficient way in developing the system, by locating locations of faults, or analyzing the correlation between some features (e.g., complexity, calls, depth) with software faults. Most related works have heavily focused on descriptive and predictive studies, which addressed questions like 'what' and 'when' [137]. Predictive focused on predicting new observation. On the other hand, explanatory works focused on historical data and address questions like 'how' and 'why' [137]. Predictive studies focus on the response variable rather than the independent confounders, by trying to predict the status of the software unit (i.e., fault prone, or non-fault prone). Explanatory works, however, try to address what independent confounders are likely to cause software faults and also quantify the probability of these confounders to occur. To measure the power of a model, each predictive and explanatory models have their distinct measurement power.

[1] A *failure* is a departure of the system or system component behavior from its required behavior [19]

[2] A *fault* is defined as an accidental condition, which if encountered, may cause the system or system component to fail to perform as required [19]

Predictive models performance are measured using several methods that mainly consider the rate of the correct classification and the closeness of the estimated values to the actual values. Explanatory models performance are usually measured by the goodness of fit, which measure the amount of data that can explain the model.

Adding more confounders to a model can help to improve the prediction, but it is not necessarily help to explain. Also, the risk for multicollinearity in the model may increase with this practice. Multicollinearity leads to a model that is not reliable due to the significant change in the estimation when minor change is made. Further, the explanation of the model is misleading, because of the highly correlated confounders are added in the same model. Therefore, in explanatory work, confounders are carefully selected and also tested for multicollinearity. We use observational method (i.e., case study), which is the most common and option, using data from open-source projects.

This chapter presents a novel methodology for explanatory research in software fault proneness. The methodology is applied in a large data set from Eclipse project, and replicate on Apache projects. Specifically, we use, for the first time in software engineering, a case-control study approach to identify confounders (i.e., software metrics) that affect software fault proneness. Our work is inspired by the successful tradition of using case-control studies in areas such as social, behavioral, and health sciences. For example, case-control studies have been used to investigate the relationship between smoking and lung cancer, research related to breast cancer, and sociology and psychology studies [138].

Research in this area use three kinds of confounders. Static code confounders represent the main characteristics of the software unit such as the lines of code, the complexity of the code, the number of Method Calls, and the comment lines. Change confounders represent the change in the code before and after the release. The change happens due to fixing faults realized in the initial design. Changes can also occur based on developmental needs, or updates to the original software. Metrics measuring the change in the original code include: Revision, Developers, refactoring, and Code Churn. Some studies use the terms (process) or (in-process) confounders for these kinds of confounders, but in our case we will use the term change confounders. The third kind is related to the social structure and communication between Developers and bug reporters, or between the different Developers.

In our explanatory model, the response variable is the Postrelease fault proneness of files, measured in the existence or absence of Postrelease faults. To account for multiple explanatory confounders and their interactions, we use case-control analysis based on logistic regression. Specifically, files with Postrelease failures are treated as *cases*, whereas files that are fault free post-release constitute the *controls*. We treat the file size, measured in lines of code (LOC), as the confounding factor that we want to control for, but are not interested in treating it as an explanatory variable. Therefore, we match the cases to controls on LOC. We start the process of building the multivariable model with a large set of static code and change confounders, which are reduced by testing the pairwise correlation (using Spearman test) and analyzing the multicollinearity.

Because of the use of matched data for LOC, we use conditional maximum likelihood estimation for estimating the parameters, which are then used to estimate the odds rations (OR) and their 95% confidence intervals (CI). Instead of the relative risk, case-control approximate the OR which can be used to measure the risk of a factor [139]. The OR quantify the effect of specific software confounders on fault proneness, measured by the presence of Postrelease faults. Note that the primary advantage of matching is that it leads to tighter CI around the OR than the CI that would be achieved without matching. The narrower is the difference between the lower and the upper values of the CI, the more precise is the odds ratio. These evidence-based findings supports decision-making, and software practitioners by leading them to cost-efficient development and improving software quality.

The main contributions of the research work presented in this chapter are

- Proposing a methodology for conducting case-control studies aimed at quantifying the effect of software confounders on software fault proneness. The approach includes multiple explanatory confounders to avoid the limitation of one-factor-at-a-time method. The explanatory confounders can be both categorical (with two ore more levels) and numerical (on an interval or ratio scale).

- Accounting for multicollinearity, which is essential because (1) software engineering confounders can be highly correlated, and (2) the presence of multicollinearity can lead to misleading interpretations in explanatory models.

- Matching the cases (i.e., files with Postrelease faults) to controls (i.e., files with no Postrelease faults) by size, using LOC confounder as a covariate to get the efficient OR estimates, with tight CI. LOC is used as a controlling confounder and not as an explanatory confounder.

- Accounting for interactions of confounders in this explanatory study. The main approach is to explain how a single confounder contribute to the software fault proneness. Interactions can help to understand how would the response confounder affected, when two or more explanatory confounders are interacting. This adds more useful information to software practitioners, when they have to deal with two confounders at the same time and what best practice they should follow.

To the best of our knowledge, this is the first work to adapt the case-control approach to explanatory studies in software engineering in general, and to software fault proneness studies in particular. Using matching allowed us to account for the size of software units (in this case files) without using it as a factor and/or avoiding using fault density as a response variable, which would have been problematic. Last, none of the previous explanatory studies focused on software fault proneness [3, 4, 5, 6, 7, 8, 9] considered the interactions between software confounders that, as shown in this paper, provide additional insights and support better understanding. Thus, a software confounder, which on its own increases fault proneness, may interact with another confounder, and that interaction can lead to a decrease in the fault proneness.

The main empirical findings of the case control study applied on the Europa and Ganymede releases of Eclipse are as follows:

- The static code confounders Average Complexity and Method Calls did not have statistically significant effect on post-release fault proneness. Similar findings apply to their interactions with other software confounders, which were eliminated from the initial model without significantly affecting it.

- Prerelease fault proneness, Developers and Age had the highest impact on software fault proneness. Specifically, a file that is new in a given release (in our case, less that 53 weeks), has experienced prerelease faults, has more Developers involved has a much higher chance to experience Postrelease faults. Specifically,
  - Files with prerelease faults, compared to fault-free files prerelease, were two times more likely to have Postrelease faults. This result confirms the persistence of faults proneness from prerelease to post-release for our case study. A similar conclusion was found in Ganymede, with OR = 2.97.
  - The number of Developers had a strong positive association with Postrelease fault-proneness, with OR of 2.05. Files with more Developers are twice likely to be exposed to Postrelease bugs than files with fewer Developers. In Ganymede, the OR of Developers is 7.33, which means a file with more Developers is seven times more chance for Postrelease faults than a file with fewer Developers.
  - New files in the current release were over six times more likely to have post-release faults than files created in previous releases.
- While prerelease fault proneness and Age on their own increased the Postrelease fault proneness, their interaction decreased it (with OR=0.29). In other words, a new file that had prerelease faults being fixed had 71% less chance of having Postrelease faults.
- The number of Developers and prerelease fault proneness also interacted and together had a strong inverse effect on the odds of post-release faults (OR=0.45). This suggests a mediating effect of the number of Developers on the relationship between pre-release and post-release fault proneness. In Ganymede release, the OR of prerelease fault and the number of Developers was 0.17, which means the risk of Postrelease fault is less by 83%.

- On the other side, the interaction between Age and Developers increased the Postrelease fault proneness, which shows that new files with more Developers have greater chance to have Postrelease faults (with OR 1.54). A similar result was found in Ganymede with OR = 2.64, which indicates that risk for Postrelease faults is 2.6 times higher when Age and number of Developers are interacting.

3.2 Methodology

The proposed methodology for using a Case-control study for exploring the factors that affect software fault proneness is presented in the flow chart shown in Figure 3.1. The four main steps include variable selection and matching, building the full model, eliminating interactions, and eliminating confounders.

In the first step, we select the (independent) explanatory variables that are included in the initial model. This is based on the findings of the previous studies and the results of the Spearman correlation test. Then, the cases (i.e., files with Post-release faults) are selected and matched to controls (files that are fault free Postrelease). Last, the variables are centered to reduce or avoid multicollinearity between interactions and main confounders.

Building the full model is the outcome of the second step, which starts with building the initial model that includes the exposure, confounders, and interactions. This is followed by testing for multicollinearity and, if it is found to exists, removing variables to eliminate it.

The third and fourth steps eliminate from the full model the insignificant interactions and confounders using a backward elimination process that leads to a final (reduced) model.

The elimination of interactions starts with the least significant interaction (i.e., the interaction that has the lowest absolute value of t-statistics). One interaction is eliminated at a time, and then the model is rerun using the likelihood ratio test to check for any significant difference between the two models. This process is repeated until all remaining interactions are significant.

The elimination of interactions is followed by removing the insignificant confounders from the list of confounders that are not involved in any significant interaction. The end result is the final which models the data as efficiently as the full model and that can be used to quantify the effect of software confounders and their interactions on Postrelease software fault proneness.

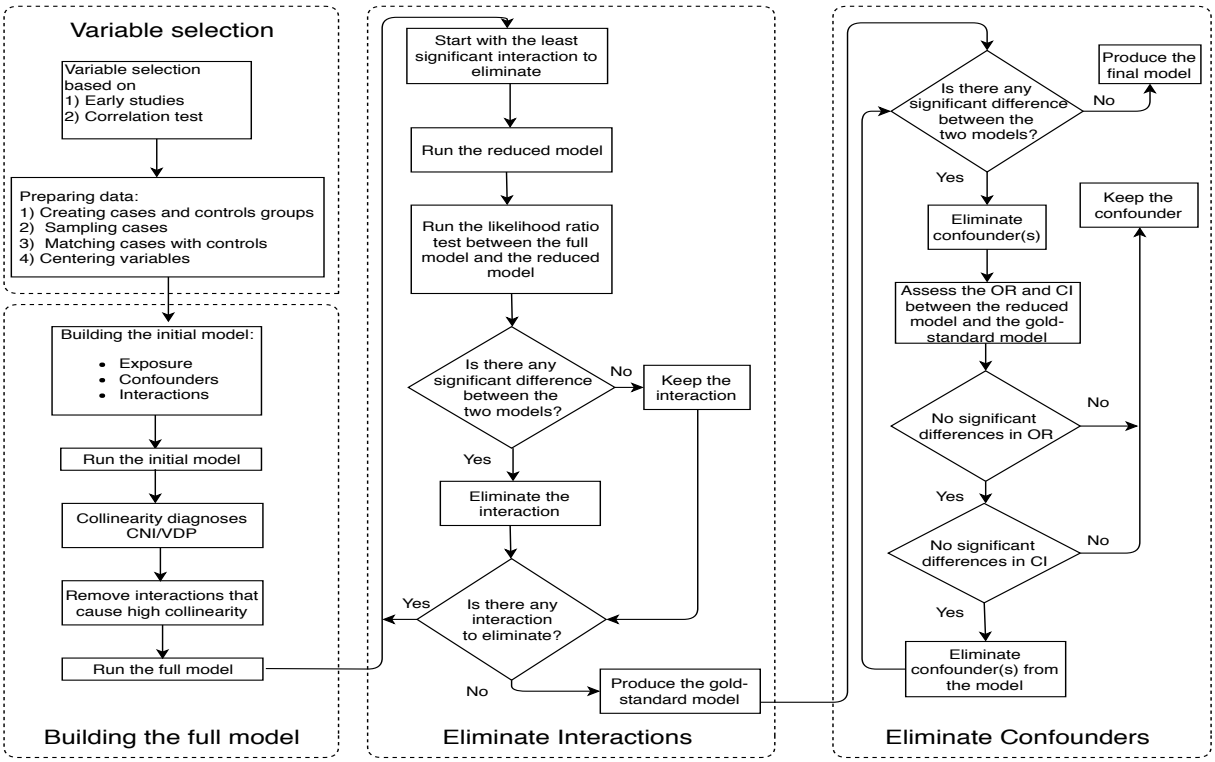


Figure 3.1: Flow chart of the methodology for conducting Case-control studies of software fault proneness

Confounder Selection and Matching

Based on the assumption that no single type of confounders can accurately capture the scope of the problem, we consider (i) static code confounders, (ii) change confounders, and (iii) confounders related to software developers. Static code confounders capture information pertaining to the source code. They range from simple confounders, such as LOC, to confounders that measure structural intricacy, such as cyclomatic complexity, function/method calls, or inheritance. Change confounders (which, in some papers, are referred to as process metrics) reflect the alterations made to source code files over the course of their exis-

tence. Modern version control systems maintain timestamped log files detailing the history of changes made to any given source code file. Examples of change confounders include the number of prerelease faults (i.e., the number of times a file was involved in fault fixing prerelease), number of revisions made to a file, codechurn (i.e., sum of the added and deleted lines), Age of a file (old/new or Age in weeks), time elapsed since the last edit, and average time between edits. Metrics related to software developers may include the number of Developers that have worked on a file and other confounders based on Developers, contributions and social networks.

Including many software confounders in the model would make it very complex and is not necessary because software metrics are known to be correlated. Therefore, we compute the correlation coefficients between pairs of confounders. (Note that since software metrics are known to follow skewed distributions [see for example [2]] we use the non-parametric Spearman correlation coefficient to test the pairwise correlation of the starting set of software confounders.) If two confounders are discovered to be highly correlated (i.e., ≥ 0.70), then it is recommended that one of them be eliminated [140]. If one of the confounders is the exposure, and the second is the confounder, then we remove the confounder and keep the exposure. If both of them are confounders, then we select one of them to be removed. At the end of this process, we are left with the set of confounders to be included in the initial model.

In the context of Case-control studies, confounders are classified into three types: dependent (i.e., response) variable, exposure, and independent variable. The response variable Y in our case is the binary variable that indicates whether or not a software unit (in our case file) has Postrelease faults. The exposure E is an independent variable that is essentially believed to be associated with the response variable. (For instance, smoking factor is believed to be the exposure factor for lung cancer.) In this study, we treat as an exposure the Prerelease faults confounder (also known as Bugfixes), which indicates if a file was involved in bug fixing Prerelease or not. The other confounders are taken as independent variables X_i.

The process of matching cases and controls is important for increasing the efficiency of the results and obtaining tighter CI of the OR estimates[141]. We first sample the cases and match them with files from controls using the LOC as the matching variable. Large files features are different than small files in terms of the complexity, Code Churn, revision, and faults. Therefore, the purpose of matching is to ensure that files are compared at the same size to avoid construct and conclusion validity violations.

Matching means that a sample has to be grouped into levels (strata) depending on the matching factor. The matching factor in this study is the file size, which is measured by the total number of LOC. We can use either one-one or one-many; this should be decided after analyzing the distribution of LOC in cases and control groups. We use Mann-Whitney test to analyze the distribution of LOC in the two samples with 95% confidence level.

The last step of this stage is to center the variables by subtracting all observations X_i from the mean. Multicollinearity is likely to happen between interactions and main confounders, and centering the variables is important to minimize the effect of the multicollinearity [142, 143].

3.2.1 Building the Model

The initial model

The exposure, all the selected confounders, and the interactions are grouped to form the initial model. The following example is used to make the case clear and easy to understand. We assumed that our initial model (Model_0) shown in equation 3.1 consists of one exposure(E), four confounders (X_1, X_2, X_3, X_4), three interactions between exposure and confounders ($E \times X_1$, $E \times X_2$, $E \times X_3$), and two interactions between confounders ($X_1 \times X_2$ and $X_3 \times X_4$). The first step is to test the multicollinearity of this model. If there is no multicollinearity detected in the initial model, then the full model would be the same as the initial model.

Model_0

$$\begin{aligned} Y =& \beta_0 E + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 \\ &+ \beta_5 (E \times X_1) + \beta_6 (E \times X_2) + \beta_7 (E \times X_3) + \beta_8 (X_1 \times X_2) + \beta_9 (X_3 \times X_4) \end{aligned} \tag{3.1}$$

Multicollinearity diagnoses

Multicollinearity is the state when two or more confounders of a model are correlated [144]. Therefore, we have to treat this problem before producing the full model Model_{full}. We conducted the correlation test earlier which helped us to estimate the correlation between every independent variable. Correlation test and centering variables were taken care of during the variable selection process. This step diagnoses multicollinearity after building the the initial model and treat multicollinearity if it exists.

The multicollinearity diagnostic tool used in this study is the values condition indices and variance decomposition proportions (CNI/VDP). This method is suggested by [145], especially for when interactions are involved in the model. While some studies used VIF as a diagnostic tool for multicollinearity, we prefer not to use it for two reasons. First, there is no agreement about the cut-off values that we can rely on to decide what is high and what is low [146, 144]. Second, some models with high VIF are more reliable (with tight CI) than models with low VIF [146].

Condition indices (CNI) indicate the existence of multicollinearity in the model [145, 144]. CNI is considered large when the first CNI (i.e., CNI_1) is larger than thirty as empirically supported by [144]. If high CNI is detected, then we have to consider the VDP values associated with it. High VDP is when at least two confounders have value greater than 0.5. Table 3.1 presents an example of CNI/VDP outputs. If multicollinearity exists between two variables, then we need to consider dropping one of them before refitting the model. We keep this process going until we get a model free of collinearity (i.e., $CNI_1 < 30$).

Table 3.1: CNI/VDP collinearity diagnoses for Europa Model_0

CNI ▷▷		CNI_1	....	CNI_{10}	
E	β_1	VDP_{11}	....	VDP_{110}	△ VDP △
X_1	β_2	VDP_{21}	....	VDP_{210}	
X_2	β_3	VDP_{31}	....	VDP_{310}	
X_3	β_4	VDP_{41}	....	VDP_{410}	
X_4	β_5	VDP_{51}	....	VDP_{510}	
$E \times X_1$	β_6	VDP_{61}	....	VDP_{610}	
$E \times X_2$	β_7	VDP_{71}	....	VDP_{710}	
$E \times X_3$	β_8	VDP_{81}	....	VDP_{810}	
$X_1 \times X_2$	β_9	VDP_{91}	....	VDP_{910}	
$X_3 \times X_4$	β_{10}	VDP_{101}	....	VDP_{1010}	

Full model

In our example, we assume that a multicollinearity was detected between X_3 and $E \times X_3$ in the initial model Model_0. Therefore, we had to drop the interaction $E \times X_3$. Then, when we diagnosed the model again, we found it to be clear of any collinearity. Thus, we have the full model Model_{full} shown by equation 3.2. This model will be carried to the next phase, which involves eliminating interactions.

$$
\begin{aligned}
FullModel(\text{Model}_f): \\
Y = &\beta_1 E + \beta_2 X_1 + \beta_3 X_2 + \beta_4 X_3 + \beta_5 X_4 + \beta_6 (E \times X_1) \\
&+ \beta_7 (E \times X_2) + \beta_9 (X_1 \times X_2) + \beta_{10} (X_3 \times X_4)
\end{aligned} \tag{3.2}
$$

3.2.2 Eliminating interactions

In the elimination process, we use backward elimination, as defined in [145]. This method starts with the highest order (i.e., the highest number of interacted confounders) to the lowest order. In case we have three interacted confounders and two interacted confounders in the same model, we start to eliminate the three interacted confounders before the two interacted confounders.

In this process, we start with the least significant interaction, which is identified by the lowest absolute value of the t-statistics (z-value) [147]. After we eliminate the interaction, we test the likelihood ratio between the original model (i.e., before eliminating the interaction) and reduced model (i.e., after removing the interaction). The likelihood ratio test measures the $\Delta\chi^2$ statistic with 95% confidence level. The results of the $\Delta\chi^2$ should be very close to zero, which indicates that the removal of the interaction does not cause a significant change to the original model, and therefore, the interaction can be removed. The process goes on until all insignificant interactions are tested.

In our example, we assume that the interaction $X_3 \times X_4$ is the least significant interaction. After we remove this interaction, we apply the likelihood ratio test on the original model (i.e., with $X_3 \times X_4$) and the reduced model (i.e., without $X_3 \times X_4$). We assume that the likelihood ratio is very low and the null hypothesis is not rejected based on the p-value > 0.05. The outcome model of this process is the gold-standard model Model_G, which contains only significant interactions and the main confounders shown in Equation 3.3.

Gold standard model (Model_G):

$$\begin{aligned} Y =& \beta_1 E + \beta_2 X_1 + \beta_3 X_2 + \beta_4 X_3 + \beta_5 X_4 + \beta_6 (E \times X_1) \\ & \beta_7 (E \times X_2) + \beta_8 (X_1 \times X_2) \end{aligned} \tag{3.3}$$

3.2.3 Eliminating confounders

The next step is eliminating insignificant confounders that have no interactions in the model. Note that the exposure should never be eliminated even if it has no interactions. Any confounder which was retained as being significantly interacting with other confounders should be retained in the model.

The process of eliminating confounders consists of two sub-processes. The first sub-process deals with OR of the gold-standard model and the reduced model (i.e., after removing the insignificant confounder(s)), which should not have discrepancies. The second sub-process involves the CI of the same two models, which also should not have meaningful differences between them. The smaller the differences between the upper and lower CIs, the better the model is. OR and CI are calculated from the exposure and its interactions only in

the two models. Considering our example (see Equation 3.3), X_1 and X_2 must be retained in the model because they have significant interactions. X_3 and X_4 can be removed because they do not have interactions and it is assumed that they are statistically insignificant. As illustrated in the list below, we have four possible scenarios for the final model.

- Scenario 1: Model$_G$ with no change, the gold standard model.
- Scenario 2: Model$_G$ without X_3
- Scenario 3: Model$_G$ without X_4
- Scenario 4: Model$_G$ without X_3 and X_4

The process of eliminating confounders is explained by algorithm 1. In this algorithm, we start by comparing the fourth scenario with the first scenario. OR of the two models are calculated as in Equation 3.4. The results of OR and CIs of all scenarios can be illustrated as in Table 3.2 and Table 3.3. The first test is to compare OR of the model of the fourth scenario ($OR_{X^i_{3,4}}$) with OR of Model$_G$ (OR_{G^i}). If there are no major discrepancies between the first two columns and the last two columns of Table 3.2, then the test is successful and we can go to test the CI. In the CI assessment, we compare between CI_{G^i} and $CI_{x^i_{3,4}}$ as in Table 3.3. If $CI_{X^i_{3,4}}$ values are either less than or equal to CI_{G^i}, then both confounders X_3 and X_4 can be eliminated; otherwise, we need to move on to the next test in Algorithm 1. The next step is to compare between OR_{G^i} and $OR_{X^i_3}$ and between CI_{G^i} and $CI_{x^i_3}$. If the test in this step fails, we move on to compare between the gold-standard model Model$_G$ and the model after removing X_4. In this step, we compare between OR_{G^i} and $OR_{X^i_4}$ and between $CI^i_{X_4}$ and CI_{G^i}. In case all tests fail, we will keep all confounders, and the final model will be the same as the gold-standard model. Assuming that we did not find any meaningful differences between the first and fourth scenarios and X_3 and X_4 were eliminated, then we would have the final model as shown in equation 3.5.

$$\text{OR} = EXP(\beta_1 E + \beta_2 E \times X_1 + \beta_3 E \times X_2) \tag{3.4}$$

$$\begin{aligned} &\text{Model}_{final} \\ Y =&\beta_1 E + \beta_2 X_1 + \beta_3 X_2 + \beta_6 E \times X_1 + \beta_7 E \times X_2 + \beta_8 (X_1 \times X_2) \end{aligned} \tag{3.5}$$

if *Gold-standard odds ratios of the Model with* X_3 *and* X_4 *(*OR_{G^i}*) almost the same as odds ratios of the model after reducing both* X_3 *and* X_4 *(*$OR_{X^i_{3,4}}$*)* **then**
  if $CI_{X34} \leq CI_G$ **then**
    Eliminate X_3 and X_4
  end
  else if *Gold-standard odds ratios of the Model with* X_3 *and* X_4 *(*OR_{G^i}*) the almost same as odds ratios of the model without* X_3 *(*$OR_{X^i_3}$*)* **then**
    if $CI_{X3} \leq CI_G$ **then**
      Eliminate X_3
    end
    else if *Gold-standard odds ratios of the Model with* X_3 *and* X_4 *(*OR_{G^i}*) almost the same as odds ratios of the model without* X_4 *(*$OR_{X^i_4}$*)* **then**
      if $CI_{X4} \leq CI_G$ **then**
        Eliminate X_4
      end
      else
        Do not eliminate any confounder and Exit
      end
    end
  end
end
;

Algorithm 1: Eliminating confounders

Table 3.2: Odd ratios test for eliminating confounders

	$X_1=0$	$X_1=1$	$X_1=0$	$X_1=1$	$X_1=0$	$X_1=1$	$X_1=0$	$X_1=1$
$X_2=1$	OR_{G^1}	OR_{G^4}	$OR_{X_3^1}$	$OR_{X_3^4}$	$OR_{X_4^1}$	$OR_{X_4^4}$	$OR_{X_{3,4}^1}$	$OR_{X_{3,4}^4}$
$X_2=2$	OR_{G^2}	OR_{G^5}	$OR_{X_3^2}$	$OR_{X_3^5}$	$OR_{X_4^2}$	$OR_{X_4^5}$	$OR_{X_{3,4}^2}$	$OR_{X_{3,4}^5}$
$X_2=3$	OR_{G^3}	OR_{G^6}	$OR_{X_3^3}$	$OR_{X_3^6}$	$OR_{X_4^3}$	$OR_{X_4^6}$	$OR_{X_{3,4}^3}$	$OR_{X_{3,4}^6}$

- OR_{G^i} is the golden-standard odds ratio calculated before eliminating any of the confounders
- $OR_{x_3^i}$ is the odds ratio calculated after eliminating confounder X_3
- $OR_{x_4^i}$ is the odds ratio calculated after eliminating confounder X_4
- $OR_{X_{3,4}^i}$ is the odds ratio calculated after eliminating X_3 and X_4

Table 3.3: Confidence intervals test for eliminating confounders

	$X_1=0$	$X_1=1$	$X_1=0$	$X_1=1$	$X_1=0$	$X_1=1$	$X_1=0$	$X_1=1$
$X_2=1$	CI_{G^1}	CI_{G^4}	$CI_{x_3^1}$	$CI_{x_3^4}$	$CI_{x_4^1}$	$CI_{x_4^4}$	$CI_{x_{3,4}^1}$	$CI_{x_{3,4}^4}$
$X_2=2$	CI_{G^2}	CI_{G^5}	$CI_{x_3^2}$	$CI_{x_3^5}$	$CI_{x_4^2}$	$CI_{x_4^5}$	$CI_{x_{3,4}^2}$	$CI_{x_{3,4}^5}$
$X_2=3$	CI_{G^3}	CI_{G^6}	$CI_{x_3^3}$	$CI_{x_3^6}$	$CI_{x_4^3}$	$CI_{x_4^6}$	$CI_{x_{3,4}^3}$	$CI_{x_{3,4}^6}$

- CI_{G^i} is the confidence intervals before eliminating any of the confounders
- $CI_{x_3^i}$ is the confidence intervals after eliminating confounder X_3
- $CI_{x_4^i}$ is the confidence intervals after eliminating confounder X_4
- $CI_{x_{3,4}^i}$ is the confidence intervals after eliminating confounders X_3 and X_4

3.2.4 Goodness of Fit Test

We applied goodness of fit (GOF) earlier to compare two models during the interaction elimination process. The goal of that test is to compare between the model before the elimination and after the elimination of an interaction and to choose between them. However, the goodness of fit test in this section is applied only on the final model to check if the model fits or not.

The goal of conducting the GOF test on the final model is to determine how well the model fits the data [148]. The GOF test used in this study is the Hosmer-Lemshow test (HL test). This test is recommended for the logistic regression model [148]. The idea of the test is to measure the differences between observed (Y_i) and predicted ($\hat{Y}_i$) outcomes for all

subjects. Therefor, the perfect fit would be when $Y_i - \hat{Y}_i = 0$. The test first computes all predicted values $\hat{Y}_i$. Second, the test orders all $\hat{Y}_i$ for all i from the largest values. Third, the test divides them into 10 quantiles and puts them in a table with the observed values. Fourth, the test calculates HL statistics and compares the χ^2 with 8 df. The H0 states that there is no evidence that this model is not fit. Therefore, when the χ^2 is low and p-value > 0.05, the H0 is not rejected and the model is considered fit.

3.3 Using a Case-control Study on Eclipse

3.3.1 Case Study 1: Europa

In this section, we illustrate the use of a Case-control study for explaining software fault proneness using Europa's release of Eclipse as a cases study. Eclipse is an open-source software that aids software development by providing development environment (IDEs). Data of this release were extracted in earlier works [2, 38]. We chose to implement studies at the file level of the extracted data because this allows us to have large sample, which makes it easier for matching.

In this section, we follow the steps explained in the previous section. We start with the variable selection and justify the selection using a correlation test and earlier findings. Also, this step involves dividing the dataset into cases and controls, sampling, matching and centering variables. Second, this section explains the process of building the full model, starting with the initial model and multicollinearity diagnostic. Third, this section explain the steps taken to reach the final model after eliminating insignificant interactions and confounders.

Confounder Selection, Sampling, and Matching

The total number of static code and change variables is thirty. Having all of them in a single model would introduce complexity and collinearity. Our selection relies on the correlation test and related studies. There are a limited number of related studies that used the explanatory approach. Therefore, we also consider variables that were proven to be effective by other studies that conducted prediction and statistical analysis. In this sense, we consider variables that showed high correlation with software faults and provided an effective prediction for software faults in earlier research.

The first variable is the LOC, which is the traditional size measurement for any piece of code. The relationship between LOC and program behavior has been explored since the era of machine language [149]. As we write more code, we introduce more errors to the code, and that makes LOC and programming errors highly correlated. In addition, fault density may

introduce negative correlation because large software units may contain a small percentage of bugs. Therefore, we cannot rely on LOC as an explanatory variable. However, LOC provides good information when used as a matching confounder [149]. Conditional logistic regression allows us to perform matching by stratifying samples based on the LOC.

Some of the options to address a confounding factor are to restrict the analysis only to one level (e.g., only large files) or to stratify the sample; that is, carry on the analysis separately for different levels of the confounding variable (e.g., small, medium, and large files). These approaches, however, are either restrictive or become cumbersome, especially if there is more than one confounding variable. Therefore, we propose using matching by LOC.

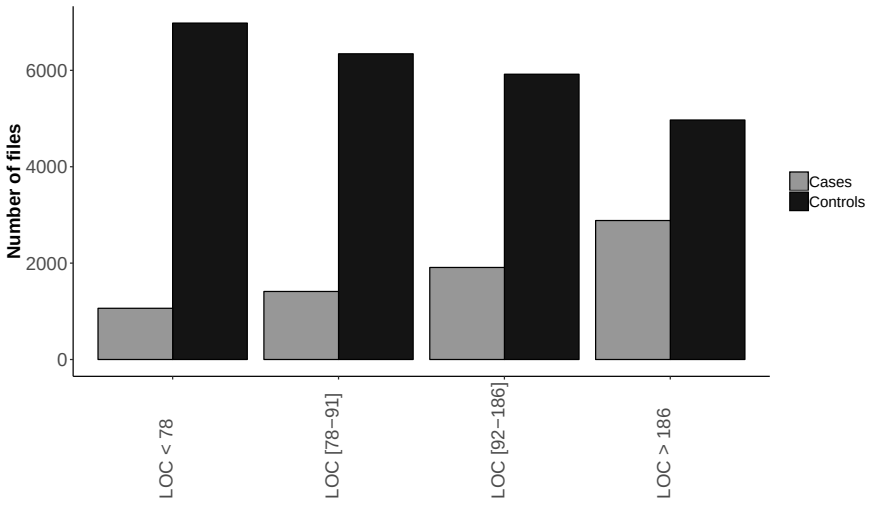


Figure 3.2: Distribution of lines of code on cases and controls of Europa

The primary advantage of matching over random sampling without matching is that matching often leads to tighter CI around the OR than the CI that would be achieved without matching.

At this stage, we divide the population of the release (i.e., Europa) into two groups: cases (i.e., faulty files) and controls (i.e., fault-free files). The total number of files extracted from the Europa release was 32,128 files, 6,896 of which had at least one Postrelease fault, whereas the rest did not.

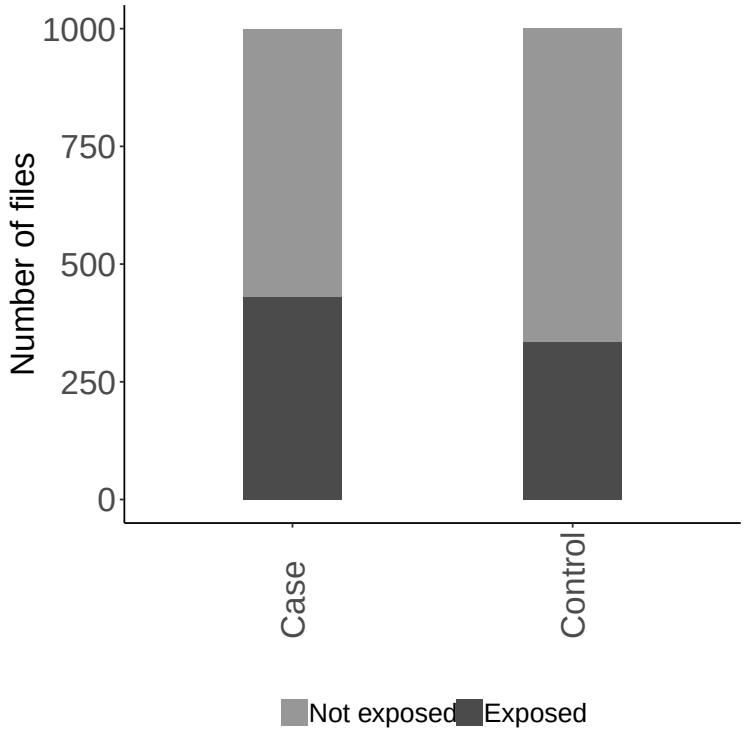


Figure 3.3: Number of exposed files (Bugfixes=1) in cases and controls

Second, we sample the first group (i.e., cases) and then match it with the second group (i.e., controls). The size of the sample is determined by the method of matching (i.e., 1-1 or 1-M). The type of matching is also determined by the distribution of LOC on the two groups, as shown in Figure 3.2, which presents LOC in four quantiles for the two groups. The two bars, of the fourth quantile (i.e., large-size files) are almost equal. In this case, the appropriate method of matching is 1-1 matching, which means for every case, we find one control with a similar size. The method of sampling and matching were automated using R programming, and the optimal sample size we were able to extract from the two groups was 1,000 files (i.e., 500 cases and 500 controls).

Exposure and Confounder Selection

The Case-control study requires assigning the exposure for the model. The exposure is selected from the independent variables, and it should have an effective impact on the response variable. In this study, Bugfixes was selected to be the exposure. Bugfixes confounder counts the number of time that a file experiences prerelease faults. It also tells how many

Table 3.4: Static code and change confounders used in this study

Metric	**Description**
Lines of Code	Total number of lines of code in a file used for matching
Method Calls	All method calls in statements and in logical expressions
Average Complexity	The summation of all method complexity values in a file divided by the total number of methods
Bugfixes	The file was involved in prerelease bug fixing process (1) or not (0)
Developers	Number of Distinct authors who made revisions to the file
Code Churn	The summation of lines of code added and lines of code deleted
Age	New files ($<$ 53 weeks) are coded 1 and old files ($\geq$ 53 weeks) are coded 0
Postrelease bugs	The file contains one or more faults (1) or is free of fault (0)

times a file has gone through the bug-fixing process. The Bugfixes variable, in this study, is represented using a binary representation. If a file (from cases or controls) has experienced at least one Prerelease fault, then the file is coded one, or zero otherwise. Early studies used the Bugfixes variable to predict or explain Postrelease faults [150, 151, 22, 21, 18]. Further, it was among the top five predictors in Eclipse releases 2.0, and 3.6 [22]. It was also among the top five predictors in Eclipse releases 2.0, 2.1, and 3.0 [21]. Figure 3.3 depicts the number of files that experienced bug fixes in the two groups (cases and controls). Around forty percent of cases were exposed to Bugfixes, and thirty percent of controls had Bugfixes.

Then, we selected the independent variables (i.e., confounders) to be included in the initial model. In some studies (e.g., [27]), researchers believed that static code confounders should only be considered for software faults prediction. In other studies (e.g., [21]), researchers believed that change confounders are better at software fault prediction. Others (e.g., [34]) suggest using a combination of both. Our study considers both types of confounders because our attempt is to explain confounders rather than provide higher prediction performance. Our selected confounders are defined in Table 3.4.

From static code confounders, we selected Average Complexity and Method Calls, which have shown a strong association with software faults in [152] and [153]. From the change variables set, we selected three confounders: Developers, Code Churn, and Age. It was found that the more Developers contributed to the file, the more likely it was that this file

have bugs [18]. Moreover, the number of Developers was among the top five predictors in Eclipse releases 3.0, 3.3, 3.5, and 3.6 [22]. Code churn [3] was among the top five predictors in Eclipse releases (2.1, 3.0, and 3.6) [22]. Additionally, the Age of a file was among the top five predictors on Eclipse (releases 2.0 and 3.0) [21]. It also appeared as one of the top five predictors in Eclipse (releases 2.1, 3.0, and 3.4) [22].

Another factor we rely on for our confounder selection is the correlation test results. The non parametric Spearman correlation test is used for our skewed data. We couple the Spearman correlation between the pairs of all thirty confounders. Highly correlated confounders (i.e., above 0.7) were excluded from the initial selection. For example, revision had a high correlation with Bugfixes, with a coefficient equal to 0.98. As a result, revision was excluded and Bugfixes (the exposure) was retained. Because of space limitations, we cannot include the correlation results of all thirty confounders. We present only the pair correlation between the selected confounders in Table 3.5.

Table 3.5: Spearman correlation coefficients for Europa release

LOC							
-0.12	Calls						
0.00	0.00	Complexity					
0.00	0.03***	0.02***	Bugfixes				
-0.01**	0.08***	0.03***	0.31***	Developers			
-0.02***	0.03***	0.04***	0.51***	0.22***	CodeChurn		
0.00	-0.02***	0.01***	0.38***	0.15***	-0.02***	Age	
-0.01***	0.05***	0.01***	0.68***	0.23***	0.28***	-0.05***	Postrelease Bugs
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$							

Basic Statistics

The basic statistics of the cases and controls samples of Europa are shown in Table 3.6, and boxplots of all confounders in the two samples are illustrated in Figure 3.4 [4]. LOC follow the same distribution in cases and controls, as shown in Table 3.6 and in Figure 3.4. Mann-Whitney test results (p-value = 0.24) indicate that there are no statistical differences

[3]Code churn in some studies was considered as the subtraction of the added lines and deleted lines. Our approach summed up the added and deleted lines as in [29, 24, 154].

[4]Postrelease bugs, Bugfixes, and Age are presented in Table 3.6 with their original numerical values. In the model, they are converted to nominal values (0,1).

in terms of the size between the two samples. Bugfixes, meanwhile, are higher in cases than in controls as shown in Table 3.6. Additionally, the results presented for the sample in Figure 3.3, show that more Bugfixes exist in cases than in controls. The boxplot in Figure 3.4 shows that files with a high number of Prerelease faults exist more in cases, and that leads to a higher mean in cases than in controls. Distinct Developers are presented with maximum values of 4 in the two samples. The average number of Developers is higher in cases than in controls, as shown in Figure 3.4, which indicates that more Developers are involved in faulty files than in fault-free files. The Code Churn values are also higher in cases than in controls, which also indicates that more lines are added to or deleted from faulty files than are added to or deleted from fault-free files. In cases, more than 50% of files are considered new files (i.e., fewer than 53 weeks) with median = 34 weeks. In controls, the median Age is 113, which indicates that more than 50% of files in controls are old files (i.e., more than 53 weeks old). Because we divide our files based on the status of the Postrelease bugs, all files in the control sample are fault-free files (i.e., all zeros) and all files in the case sample have at least one fault.

Table 3.6: Basic statistics for cases and controls samples of Europa

	Cases (files with bugs =1000)					Controls (files with no bugs =1000)					Mann-Whitney
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	p-value
LOC	1	993	268	246	242	1	997	266	202	242	0.95
Methods Call Statement	0	805	92	50	121	0	686	74	35	228	$p < 0.001$
Average Complexity	1	27	2.82	2	2.35	1	60	2.28	1.74	3.18	$p < 0.001$
Bugfixes	0	148	2.29	0	8.66	0	20	0.84	0	1.93	$p < 0.001$
Developers	1	4	1.92	2	0.67	0	4	0.73	1	0.69	$p < 0.001$
Code Churn	0	2261	59	8	170.85	0	888	12.69	0	87.32	$p < 0.001$
Age	1.14	321	82	35	87	1.14	321	130	115	81.88	$p < 0.001$
Post Release Bugs	1	20	1.87	1	1.86	0	0	0	0	0	$p < 0.001$

Building the Model

The initial model includes the exposure (i.e., Bugfixes), and five confounders (two static code and three change code confounders). The initial model is given with Equation 3.6. A contribution of this study is that the initial model includes interactions between the exposure and all other confounders and interactions between the confounders themselves.

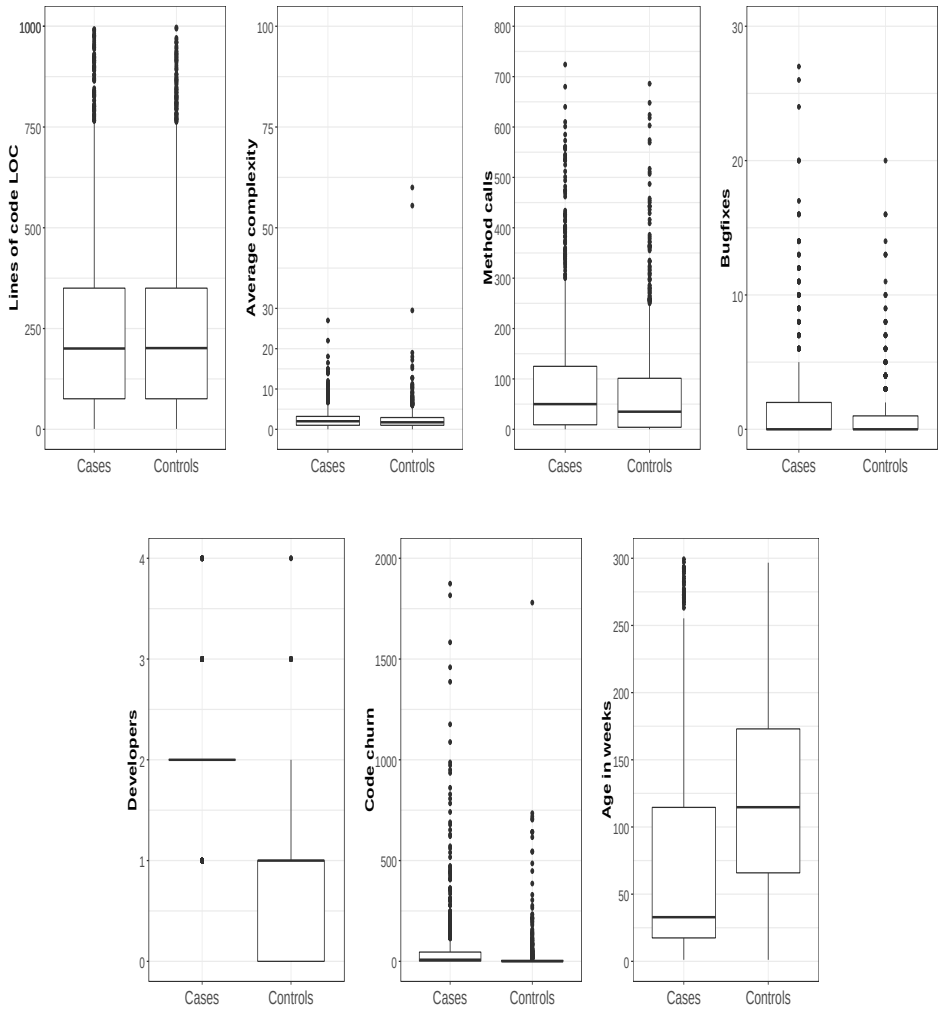


Figure 3.4: Selected confounders' boxplots of cases and controls

The multicollinearity of the initial model Model_0 is diagnosed using the CNI/VDP matrix. The result of the test is presented in Table 3.7. The top CNI ($CNI_1 = 89.3$) indicates that the model suffers from multicollinearity as the value of CNI_i exceeds the cutoff value (i.e., 30). We checked the VDPs associated with CNI_1, and we found that both Code Churn and interaction $Bugfixes \times Codechurn$ have a VDP higher than 0.5. The interaction $Bugfixes \times CodeChurn$ should be dropped from the model, and the test should be again over the reduced model. The second test results indicate that CNI_1 is reduced to 11, which means that the model is free from the multicollinearity issue. As a consequence, the full model (Model_{full}) is produced as shown by Equation 3.7.

Initial Model_0

$$\begin{aligned}\text{Y} = {} & \beta_1 \cdot Bugfixes + \beta_2 \cdot MethodCalls + \beta_3 \cdot AvgComplexity + \beta_4 \cdot Developers + \beta_5 \cdot CodeChurn \\ & + \beta_6 \cdot Age + \beta_7 \cdot Bugfixes \times MethodCalls + \beta_8 \cdot Bugfixes \times AvgComplexity \\ & + \beta_9 \cdot Bugfixes \times Developers + \beta_{10} \cdot Bugfixes \times CodeChurn + \beta_{11} \cdot Bugfixes \times Age \\ & + \beta_{12} \cdot MethodCalls \times AvgComplexity + \beta_{13} \cdot MethodCalls \times Developers \\ & + \beta_{14} \cdot MethodCalls \times CodeChurn + \beta_{15} \cdot MethodCalls \times Age \\ & + \beta_{16} \cdot AvgComplexity \times Developers + \beta_{17} \cdot AvgComplexity \times CodeChurn \\ & + \beta_{18} \cdot AvgComplexity \times Age + \beta_{19} \cdot Developers \times CodeChurn \\ & + \beta_{20} \cdot Developers \times Age + \beta_{21} \cdot CodeChurn \times Age\end{aligned} \tag{3.6}$$

Full Model_{Full}

$$\begin{aligned}\text{Y} = {} & \beta_1 \cdot Bugfixes + \beta_2 \cdot MethodCalls + \beta_3 \cdot AvgComplexity + \beta_4 \cdot Developers + \beta_5 \cdot CodeChurn \\ & + \beta_6 \cdot Age + \beta_7 \cdot Bugfixes \times MethodCalls \\ & + \beta_8 \cdot Bugfixes \times AvgComplexity + \beta_9 \cdot Bugfixes \times Developers + \beta_{11} \cdot Bugfixes \times Age \\ & + \beta_{12} \cdot MethodCalls \times AvgComplexity + \beta_{13} \cdot MethodCalls \times Developers \\ & + \beta_{14} \cdot MethodCalls \times CodeChurn + \beta_{15} \cdot MethodCalls \times Age \\ & + \beta_{16} \cdot AvgComplexity \times Developers + \beta_{17} \cdot AvgComplexity \times CodeChurn \\ & + \beta_{18} \cdot AvgComplexity \times Age + \beta_{19} \cdot Developers \times CodeChurn \\ & + \beta_{20} \cdot Developers \times Age + \beta_{21} \cdot CodeChurn \times Age\end{aligned} \tag{3.7}$$

Eliminate interactions

The full model Model_{full} at this level contains several insignificant interactions, as shown in the first column of Table 3.8. The second column represents Model_2, which eliminates the first interaction *Method Calls* × *Code churn*. The result of the likelihood ratio test is zero, which indicates that the removal did not impact the model. The third column represents Model_3, which eliminates the second interaction *Average Complexity* × *Age*. The likelihood ratio between this model and the preceding model is close to zero. $\text{Model}_1 3$ is considered the final model of this stage, and the outcome of this model is given with Equation 3.8.

Eliminate confounders

Our gold-standard model at this stage is Model_{13}. In this model, we have three significant interactions $Bugfixes \times Developers$, $hbox Bugfixes \times hbox Age$, and $Developers \times hbox Age$. Therefore, Bugfixes, Developers, and Age should remain in the final model. Other confounders (i.e. Method Calls, Average Complexity and Code Churn) can be eliminated since they do not have interactions and are not significant in the model. Hence, we have eight possible scenarios as shown in Table 3.9. Every confounder has the status to either be removed (NO) or retained in the model (YES). The first scenario (scenario 1) is to keep all confounders, which is the same as the gold-standard model. The last scenario (scenario 8) is to eliminate them all. The outcome of this process is the reduced model (final model).

Because space is limited, we present a comparison between scenario 1 (the gold standard model) and scenario 8. As explained in the methodology section, it is always better to start with the scenario that suggests to eliminating all confounders as long as no significant discrepancies exist in OR and CI between the gold-standard model and reduced model.

Equations 3.9 and 3.10 show the calculations for OR of scenarios 1 and 8. The results of the two equations are presented in Tables 3.10 and 3.11. We need to first compare OR of the two models (scenario 1 and 8) from Table 3.10. If the comparison shows no significant differences, then we compare their CI values in Table 3.11. If the CI results of the two scenarios show no meaningful difference, then we can decide to remove all confounders and pick scenario 8 to be the final model.

$$\begin{aligned} &\text{Gold-standard Model}_G \\ \text{Y} = {} & 1.48 \cdot Bugfixes + 0.06 \cdot MethodCalls + 0.002 \cdot AvgComplexity + 0.69 \cdot Developers \\ & + 0.87 \cdot CodeChurn + 1.36 \cdot Age - 0.73 \cdot Bugfixes \times Developers \\ & - 1.23 \cdot Bugfixes \times Age + 0.45 \cdot Developers \times Age \end{aligned} \tag{3.8}$$

$$\text{Gold Standard Odd Ratio } (OddRatio_{scenario1}) = \text{EXP}(0.70 \cdot Bugfixes - 0.78 \cdot Bugfixes \times Developers - 1.21 \cdot Bugfixes \times Age) \tag{3.9}$$

Table 3.7: CNI/VDP collinearity diagnoses for Europa's initial model Model_0

CNI ▷▷	87.3	20.6	16.3	15.4	13.5	12.2	10.8	10.2	8.5	7.7	7.3	6.1	5.6	4.5	4.3	3.9	3.6	2.8	2.2	1.5	1	△ VDP ▽
Bugfixes	0.00	0.30	0.30	0.30	0.00	0.02	0.00	0.00	0.00	0.05	0.01	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.00	
Method calls	0.00	0.02	0.42	0.07	0.04	0.21	0.01	0.06	0.00	0.12	0.00	0.00	0.01	0.00	0.00	0.01	0.01	0.02	0.00	0.00	0.00	
Average Complexity	0.00	0.00	0.18	0.28	0.35	0.02	0.00	0.13	0.00	0.01	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Developers	0.03	0.83	0.06	0.01	0.00	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	
Code churn	0.99	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Age	0.01	0.14	0.07	0.22	0.41	0.03	0.00	0.01	0.03	0.04	0.02	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Bugfixes × Method Calls	0.00	0.40	0.14	0.01	0.00	0.01	0.01	0.11	0.18	0.01	0.06	0.00	0.02	0.00	0.02	0.02	0.00	0.00	0.01	0.00	0.00	
Bugfixes × Avg complexity	0.01	0.02	0.00	0.00	0.04	0.11	0.02	0.33	0.00	0.07	0.04	0.20	0.12	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.00	
Bugfixes × Developers	0.05	0.04	0.22	0.48	0.01	0.04	0.04	0.02	0.03	0.01	0.03	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	
Bugfixes × Code churn	0.99	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Bugfixes × Age	0.00	0.09	0.00	0.00	0.03	0.18	0.12	0.02	0.05	0.18	0.01	0.13	0.00	0.09	0.00	0.04	0.00	0.04	0.00	0.00	0.00	
Method Calls × Avg complexity	0.00	0.01	0.14	0.20	0.19	0.00	0.00	0.12	0.01	0.16	0.02	0.09	0.03	0.00	0.00	0.02	0.01	0.00	0.01	0.00	0.00	
Method Calls × Developers	0.00	0.67	0.00	0.00	0.02	0.19	0.04	0.00	0.00	0.01	0.00	0.01	0.00	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	
Method Calls × Code churn	0.00	0.01	0.01	0.00	0.00	0.01	0.05	0.00	0.53	0.02	0.34	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	
Method Calls × Age	0.00	0.17	0.13	0.04	0.13	0.00	0.00	0.08	0.11	0.02	0.17	0.10	0.01	0.00	0.01	0.02	0.01	0.00	0.01	0.00	0.00	
Average Complexity × Developers	0.00	0.04	0.03	0.11	0.26	0.20	0.01	0.24	0.00	0.07	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	
Average Complexity × Code churn	0.00	0.00	0.00	0.06	0.07	0.00	0.07	0.02	0.00	0.03	0.01	0.00	0.39	0.02	0.16	0.06	0.10	0.00	0.01	0.01	0.00	
Average Complexity × Age	0.00	0.02	0.03	0.01	0.00	0.03	0.00	0.39	0.00	0.00	0.00	0.03	0.27	0.02	0.13	0.03	0.00	0.01	0.03	0.00	0.00	
Developers × Code churn	0.00	0.00	0.02	0.08	0.00	0.12	0.61	0.01	0.06	0.03	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	
Developers × Age	0.00	0.04	0.01	0.07	0.35	0.32	0.03	0.04	0.05	0.00	0.03	0.03	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	
Code churn × Age	0.01	0.00	0.00	0.01	0.02	0.03	0.17	0.02	0.01	0.00	0.00	0.00	0.03	0.22	0.05	0.02	0.34	0.04	0.00	0.01	0.00	

Table 3.8: Europa release model reduction using backward hierarchal elimination for interactions

Metrics	$Model_{full}$	$Model_2$	$Model_3$	.....	$Model_{11}$	$Model_{12}$	$Model_{13}$
Bugfixes	1.88**	1.88**	1.88**		1.88**	1.91**	2.01**
Method Calls	0.99+	0.99+	0.99+		0.99	0.99	0.99
Average Complexity	0.99	0.99	1.00		0.99	0.99	0.99
Developers	1.88**	1.88**	1.88**		1.90**	1.91**	2.03***
Code Churn	1.00	1.00+	1.00+		1.00+	1.00	1.00
Age	6.74***	6.74***	6.73***		6.61***	6.61***	6.63***
Bugfixes × Method Calls	1.21	1.21	1.22				
Bugfixes × Average Complexity	1.09	1.09	1.08				
Bugfixes × Developers	0.48***	0.48***	0.48***		0.48***	0.48***	0.44***
Bugfixes × Age	0.28***	0.28***	0.28***		0.29***	0.29***	0.29***
Method Calls × Average Complexity	0.93	0.93	0.93				
Method Calls × Developers	0.84+	0.84+	0.84+		0.89+		
Method Calls × Code Churn	1.00						
Method Calls × Age	1.15	1.15	1.15				
Average Complexity × Developers	0.90	0.90	0.91				
Average Complexity × Code Churn	0.85	0.86	0.87				
Average Complexity × Age	1.04	1.04					
Developers × Code Churn	0.73	0.73+	0.73+		0.74+	0.77	
Developers × Age	1.54*	1.54*	1.54*		1.56*	1.55*	1.54*
Code Churn × Age	1.13	1.13	1.13				
χ^2	324***	324***	323.9***		318.4***	315.7***	313.3**
Likelihood Ratio Test = $\Delta\chi^2$		0	0.05		1.9	2.70	2.43
R^2	0.29	0.29	0.29		0.28	0.28	0.28
Deviance Explained	%44.07	%44.07	%44.11		%44.40	% 44.28	% 44.30
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$							

$$OddRatio_{scenario8} = \text{EXP}(0.63 \cdot Bugfixes - 0.73 \cdot Bugfixes \times Developers - 1.23 \cdot Bugfixes \times Age) \tag{3.10}$$

Table 3.10 presents OR of the two models (gold-standard model and scenario 8). Age and Developers are the two variables on the table because they are the only confounders interacting with the exposure. The two columns in the left of the table are OR of the gold-standard model. Gray cells in the table indicate that these OR are exactly like their

Table 3.9: The eight scenarios to consider in eliminating the confounders

	Method Calls	Average Complexity	Code Churn
Scenario 1	YES	YES	YES
Scenario 2	YES	YES	NO
Scenario 3	YES	NO	YES
Scenario 4	YES	NO	NO
Scenario 5	NO	YES	YES
Scenario 6	NO	YES	NO
Scenario 7	NO	NO	YES
Scenario 8	NO	NO	NO

YES: To keep the confounder in the model

NO: To eliminate the confounder from the model

Table 3.10: Golden standard odds ratio and odds ratios of scenario 8

		Age		Age	
		0	1	0	1
Developers	1	0.96	0.72	0.96	0.72
	2	0.77	0.66	0.77	0.66
	3	0.68	0.63	0.69	0.63
	4	0.64	0.62	0.65	0.62
		$OR_{scenario1}$		$OR_{scenario8}$	

Table 3.11: Confidence interval assessment of Europa

		Age		Age	
		0	1	0	1
Developers	1	0.00	0.03	0.00	0.03
	2	0.02	0.04	0.02	0.03
	3	0.03	0.04	0.03	0.04
	4	0.04	0.04	0.04	0.04
		$CI_{scenario1}$		$CI_{scenario8}$	

equivalent OR in the gold-standard model. Similarly, CI (Upper 95% - Lower 95%) results are presented in Table 3.11. The highlighted cells indicate that those cells have either exact values or smaller values than the cells on the gold-standard model. Both tables indicate that differences are not meaningful, which means removing the two confounders (i.e., Method

Calls, Average Complexity, and Code Churn) from the model is not harmful to the model. The final model Model_{final} is as shown by Equation 3.11 and the final results are presented in Figure 3.5. The goodness of fit according to HL for Europa indicates that the final model is fit at 95% confidence level.

$$\begin{aligned} &\text{Model}_{final} \\ Y =& 0.63 \cdot \mathit{Bugfixes} + 0.47 \cdot \mathit{Developers} + 1.88 \cdot \mathit{Age} - 0.73 \cdot \mathit{Bugfixes} \times \mathit{Developers} - 1.23 \cdot \mathit{Bugfixes} \times \mathit{Age} \\ &+ 0.45 \cdot \mathit{Developers} \times \ \mathit{Age} \end{aligned} \tag{3.11}$$

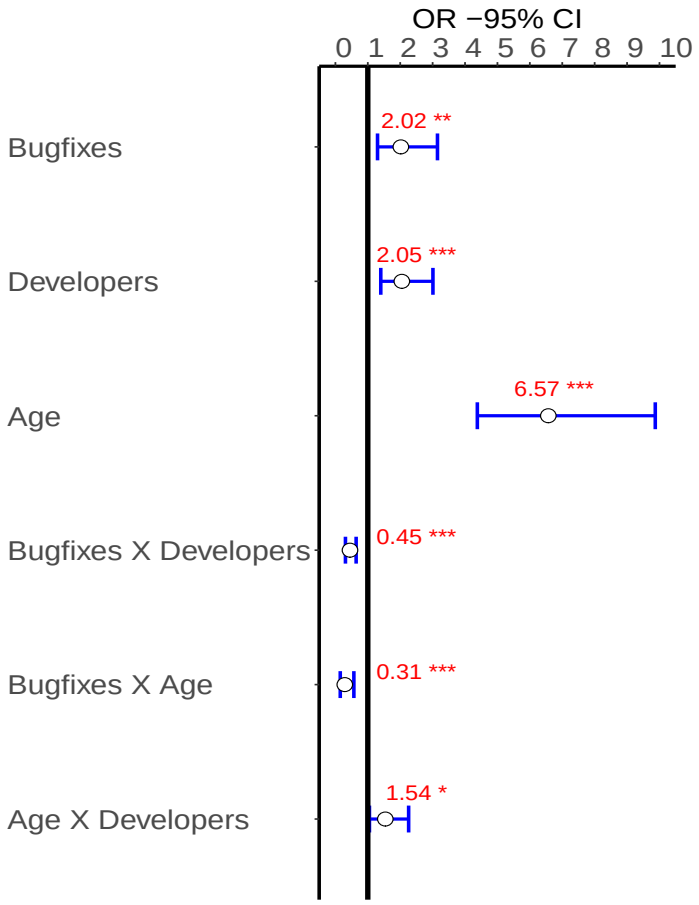


Figure 3.5: Europa final model odd ratios and confidence intervals

Discussion of the results

In our analysis, we use OR to explain the amount that the confounder or interaction contribute to Postrelease faults. In the Case-control model, the OR is defined as the probability of cases (event to occur) over the probability of controls (event do not occur). The value of the OR can be around one when both events (cases and controls) are equal or statistically very close. If the probability of cases is statistically higher than the probability of controls, the OR becomes greater than one. If the probability of controls is statistically higher than the probability of cases, the OR becomes less than one but higher than or close to zero.

The following list presents main individual confounders that can be used to explain Postrelease fault proneness of Europa files:

- Bugfixes (OR = 2.02) : Files with prerelease bugs have two times higher chance to experience Postrelease bugs.
- Age (OR = 6.57): New files operating in the current release are more than six times likely to have Postrelease bugs than old files operating in the previous release.
- Developers (OR = 2.05): More Developers contributing the same file increases chances of Postrelease faults.

This study shows that prerelease bugs contribute to Postrelease bugs with OR = 2.02. This result is consistent with what was found in [5]. The explanatory study [5] used releases 2.0, 2.1, and 3.0 from Eclipse. In release 2.0, the prerelease bugs had OR = 1.9. The OR for prerelease bugs was even higher in releases 2.1 (OR = 3.27) and 3.0 (OR = 3.28).

Age has the highest OR compared to other confounders with OR = 6.57. This result indicates that new files have a higher probability (more than six times) to have Postrelease faults than old files. Many prediction studies suggest that Age is one of the best confounders for predicting faults [22, 21]. One study [18] used correlation test to indicates the existence of an association between new files and faults.

The Developers has an OR = 2.05, which means that files with more Developers working on them are two times more likely to experience faults than files with of Developers. While some studies (e.g., [155]) observed a correlation between Developers and software faults, other studies did not observe such a correlation [35].

The following list presents the main interactions that can be used to explain Postrelease fault proneness of Europa files:

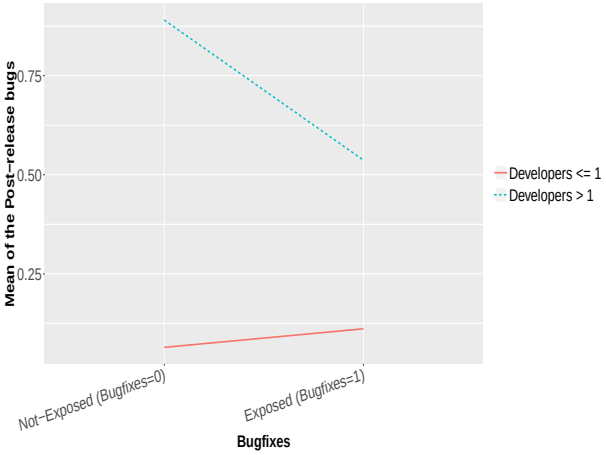
- Bugfixes with Developers (OR = 0.45): Developers working on fixing software bugs will reduce the chance (by 55%) of having Postrelease bugs.
- Bugfixes with Age (OR = 0.29): New files exposed to the bugfixing process have a 71% less chance to have Postrelease bugs.
- Age with Developers (OR = 1.54): More Developers working on new files will increase the chance (by 54%) of having Postrelease bugs.

The interactions of the three confounders, Bugfixes, Developers and Age are presented in Figures 3.6a, 3.6b, and 3.6c. The figures help to understand the relationship between the two confounders with the mean of the Postrelease bugs. To simplify the figures, we categorized the Developers into two levels using the median value (i.e., median of Developers = 1) as a cut-off.

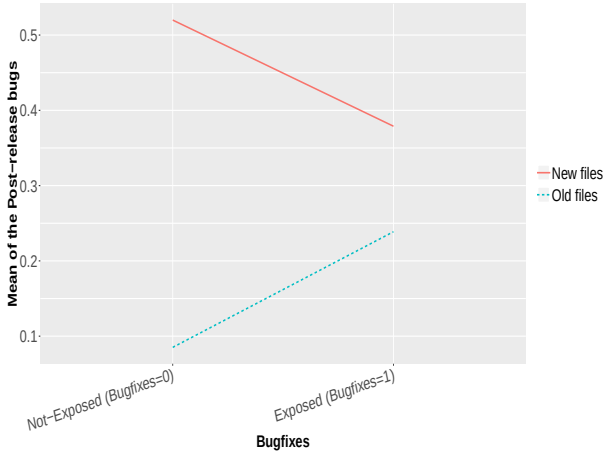
Bugfixes with Developers had an OR smaller than 1 (OR = 0.45). When analyzing the main confounder of Bugfixes, we found that a file has a higher chance to be faulty if it was previously exposed to the bugfixing process. The interaction between Bugfixes and Developers tells us that having more Developers working in fixing bugs reduces the chance of Postrelease bugs compare to more Developers working on files that were not exposed to Bugfixes. As shown in Figure 3.6a, fewer Developers are always better whether or not files are exposed to the bugfixing process. However, with a high number of Developers, the mean of Postrelease bugs reduces when it interacts with the bugfixing process.

The second significant interaction is between Bugfixes and Age, which has an OR lower than one (OR = 0.29). The interaction plot in Figure 3.6b illustrates that new files always have a higher mean of Postrelease bugs. However, the mean of Postrelease bugs reduces when new files are exposed to the bugfixing process.

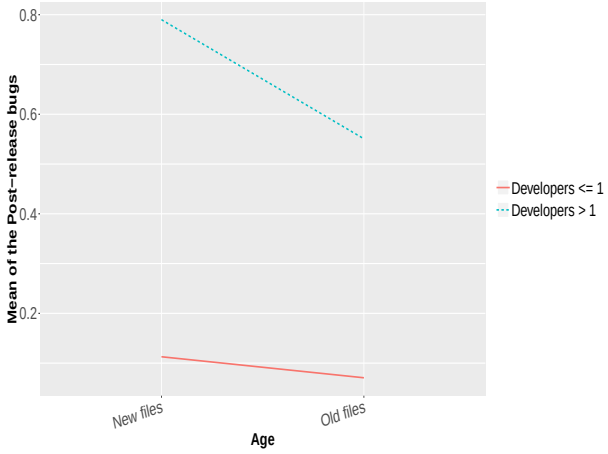
The interaction between Age and Developers has an OR above 1 (OR = 1.54). The mean of the Postrelease bugs is at the lowest level when the number of Developers is low whether the files are old or new, as shown in Figure 3.6c. However, the mean of Postrelease bugs increases when a large number of Developers are involved in new files.



(a) Interaction plot between Bugfixes and Developers in Europa



(b) Interaction plot between Bugfixes and Age in Europa



(c) Interaction plot between Developers and Age in Europa

Figure 3.6: Significant interaction plots of Europa

It is important to point out that other confounders were considered in the initial model, but they were eliminated with their interactions in the final model. Some studies (e.g., [27]) recognized static code confounders as good predictors. However, we found that static code confounders (i.e., Method Calls, and Average Complexity) were not statistically significant, and they were excluded along with their interactions from the final model. Regarding change confounders, some studies (e.g., [22]) suggest that Code Churn is a good predictor. However, Code Churn and its interactions were excluded from our final model. These confounders were good predictors in some studies, but we were not able to use them to explain software fault proneness for Europa. This indicates that some confounders can be good predictors for software faults but may not be helpful in explanation.

3.3.2 Case Study 2:Ganymede

As the second study for this research, we used the Ganymede dataset, which is another release of Eclipse. The total number of extracted files was 32,648. Faulty files (cases) numbered 5,391 and fault-free files (controls) numbered 27,257 files. The distribution of the LOCs among cases and controls is shown in Figure 3.7. We used the same approach we did with Europa in sampling and matching. Our sample size contained 1,000 files from cases and the same number from controls.

The exposure of the model is the Bugfixes, and Figure 3.8 illustrates the exposure and the number of occurrences in the two groups. Fifty percent of cases were exposed to Bugfixes, and 20% of files in controls had prerelease faults. Table 3.12 shows the basic statistics of the selected confounders of Ganymede. The data were clearly well matched in terms of the LOC according to the Mann-Whitney test. Controls (i.e., fault-free files) has less Code Churn, fewer Developers, fewer prerelease bugs, and older files as shown in Table 3.12. Boxplots of selected confounders are shown in the Figure 3.9. LOC, Average Complexity, and Method Calls confounders are clearly matched in the two samples. Medians of Bugfixes, Developers, and Code Churn are higher in cases than in the controls group. Files in the controls group are older in Age in than in the cases group.

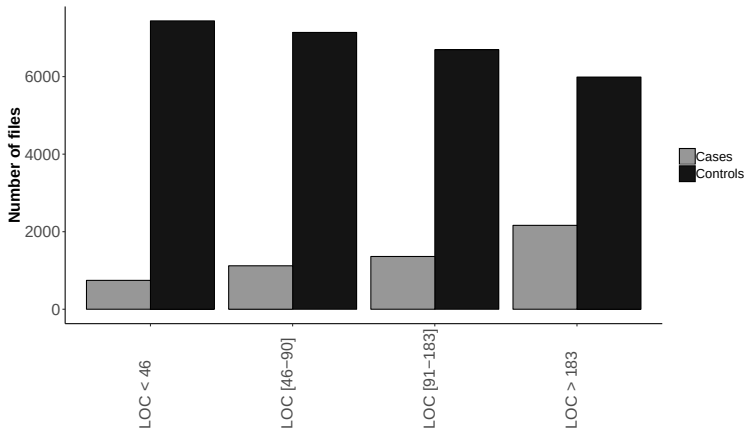


Figure 3.7: Distribution of lines of code on cases and controls of Ganymede

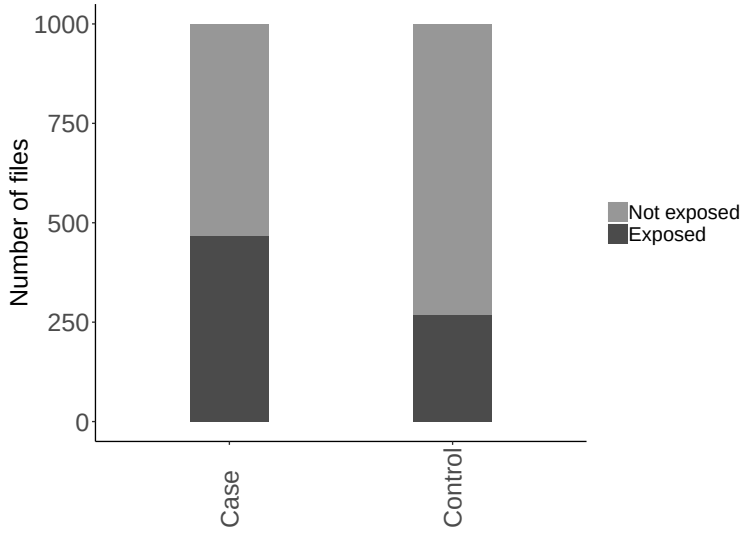


Figure 3.8: Number of exposed files (Bugfixes=1) in cases and controls

Table 3.12: Basic statistics for cases and controls samples of Ganymede

	Cases (files with bugs = 1,000)					Controls (files with no bugs = 1,000)					Mann-Whitney
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	p-value
LOC	10	995	217	139.05	208.05	8	999	217	139.5	208.11	0.99
Methods Calls	0	667	64.53	30	91.09	0	796	59.83	27	85.03	0.35
Average Complexity	0	19	2.2	1.83	1.96	0	45.8	2.14	1.82	2.26	0.27
Bugfixes	0	96	2.83	0	7.31	0	16	0.53	0	1.18	$p < 0.001$
Developers	1	5	1.65	1	0.79	0	5	0.68	0	0.84	$p < 0.001$
Code Churn	0	1487	50.08	9	133.43	0	1821	9.45	0	85.13	$p < 0.001$
Age	2.42	373.71	117.61	91.57	95.84	1.71	373.71	166.71	156.92	93.01	$p < 0.001$
Postrelease bugs	1	36	2.67	2	2.91	0	0	0	0	0	$p < 0.001$

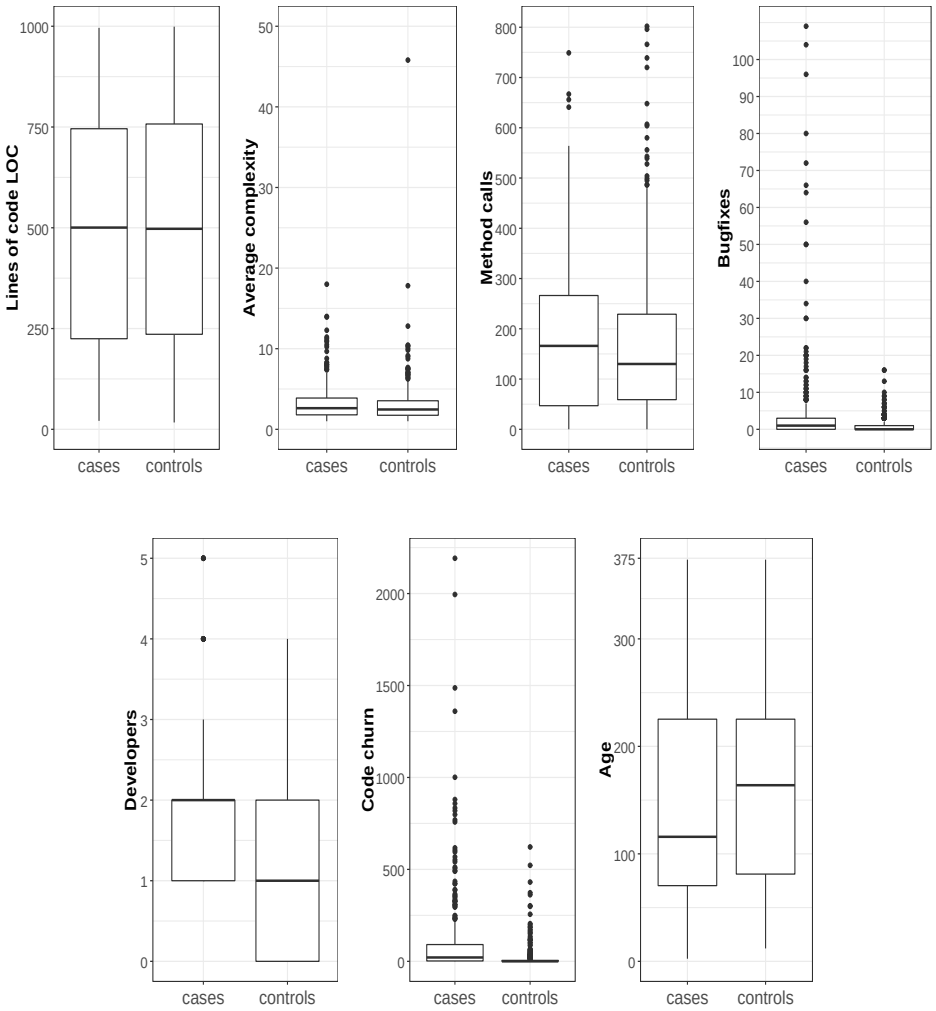


Figure 3.9: Selected confounders' boxplots of cases and controls in Ganymede

Table 3.13: Spearman correlation coefficients for the Ganymede release

LOC							
0.83***	Calls						
0.31***	0.35***	Complexity					
0.16***	0.19***	0.08***	Bugfixes				
0.21***	0.22***	0.11***	0.43***	Developers			
0.20***	0.18***	0.05***	0.51***	0.24***	Code churn		
0.08***	0.05***	-0.02***	-0.16***	-0.17***	-0.11***	Age	
0.14***	0.18***	0.09***	0.58***	0.44***	0.29***	-0.16***	Postrelease Bugs
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$							

Spearman test as conducted to test the collinearity of the initial model confounders. The correlation results are presented in Table 3.13, which contains pair-wise correlation of only the selected confounders. All the coefficients show low correlations except in two cases: high correlation between Method Calls and LOCs (0.83), and medium correlation (0.58) between Bugfixes and Postrelease bugs. The two cases are not a problem because they did not occur between explanatory variables (i.e., independent variables).

We start with the same initial model as in Europa (see Equation 3.6). The CNI/VDP for the initial model had high CNI and high VDP associated with the interactions of Bugfixes and Code Churn. This interaction is dropped from the initial model, and the CNI/VDP for the reduced model gives low CNI ($CNI_1 = 11$), which indicates that multicollinearity is solved. The full model is similar to the full model for Europa as shown by Equation 3.7.

Next, we applied the same method as we did in Europa to eliminate the interactions and confounders shown in Table 3.14. The result of this process was that eight insignificant interactions were eliminated that had no significant impact on the model based on the likelihood ratio test conducted between every reduced model and the model before the elimination. Model_9 is the result of this process, which contains six significant interactions, four significant confounders, and two insignificant confounders. The two insignificant confounders cannot be eliminated from the model because they have significant interactions that retained in the model. Hence, the process should stop at this point and use the model produced from this step (i.e., Model_9) as the final model (i.e., Model_{final}). The final model is shown in Equation 3.12. The HL goodness of fit test of Ganymede's final model indicates that predicted values are not statistically different than actual values at the 95% confidence level.

$$
\begin{aligned}
& Model_{final} \\
Y = & 1.08 \cdot Bugfixes - 0.07 \cdot MethodCalls + 0.18 \cdot AvgComplexity + 1.99 \cdot Developers \\
& + 1.52 \cdot CodeChurn + 0.16 \cdot Age - 1.73 \cdot Bugfixes \times Developers + 0.001 \cdot Bugfixes \times Age \\
& - 0.00 \cdot MethodCalls \times CodeChurn + 0.003 \cdot AvgComplexity \times CodeChurn \\
& - 0.20 \cdot AvgComplexity \times Age + 0.97 \cdot Developers \times Age
\end{aligned}
\tag{3.12}
$$

Discussion of the results

Unlike Europa, none of the main confounders in Ganymede were eliminated. The following list presents the main individual confounders that can be used to explain Postrelease fault proneness of Ganymede files:

Table 3.14: Ganymede release model reduction using backward hierarchal elimination for interactions

Metrics	Model_{full}	Model_2	Model_3	...	Model_7	Model_8	Model_9
Bugfixes	2.35+	2.37*	2.37*		2.70**	2.75**	2.97**
Method Calls	0.77	0.77	0.77		0.90	0.94	0.92
Average Complexity	1.06	1.06	1.06		1.19+	1.18+	1.2+
Developers	7.04***	7.00***	6.99***		7.27***	7.36***	7.33***
Code Churn	1.07**	1.07**	1.07**		8.14**	5.44**	4.60**
Age	1.11	1.11	1.10		1.11	1.09	1.18
Bugfixes × Method Calls	1.00						
Bugfixes × Average Complexity	1.06	1.06	1.06				
Bugfixes × Developers	0.18***	0.18***	0.18***		0.18***	0.18***	0.17***
Bugfixes × Age	1.00*	1.00*	1.00**		1.00*	1.00*	1.00*
Method Calls × Average Complexity	1.00	1.00	1.00				
Method Calls × Developers	0.99	0.99					
Method Calls × Code Churn	1.00*	1.00*	1.00*		1.00*	1.00**	0.99***
Method Calls × Age	1.00	1.00	1.00				
Average Complexity × Developers	1.01	1.01	1.01				
Average Complexity × Code Churn	1.00*	1.00*	1.00*		1.00**	1.00**	1.00**
Average Complexity × Age	0.76*	0.76**	0.76**		0.80*	0.80*	0.81*
Developers × Code Churn	0.99	0.99	0.99		0.99	0.99	
Developers × Age	3.07***	3.06***	3.06***		3.05***	2.95***	2.64**
Code churn × Age	0.99	0.99	0.99		0.99		
χ^2	643.3***	643.3***	643.3***		640.9***	639.9***	638.5***
Likelihood Ratio Test = $\Delta\chi^2$		0.01	0.00		0.72	1.07	1.40
R^2	0.50	0.50	0.50		0.50	0.50	0.50
Deviance Explained	%60.19	%60.15	%60.15		%60.43	%61.26	%61.47
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$							

- Bugfixes (OR = 2.97): Files that had prerelease bugs are three times more likely to experience Postrelease bugs.
- Average Complexity (OR = 1.2): Complex files are 20% more likely to experience Postrelease bugs.
- Developers (OR = 7.33): Files with more Developers have seven times higher chance to have Postrelease faults.

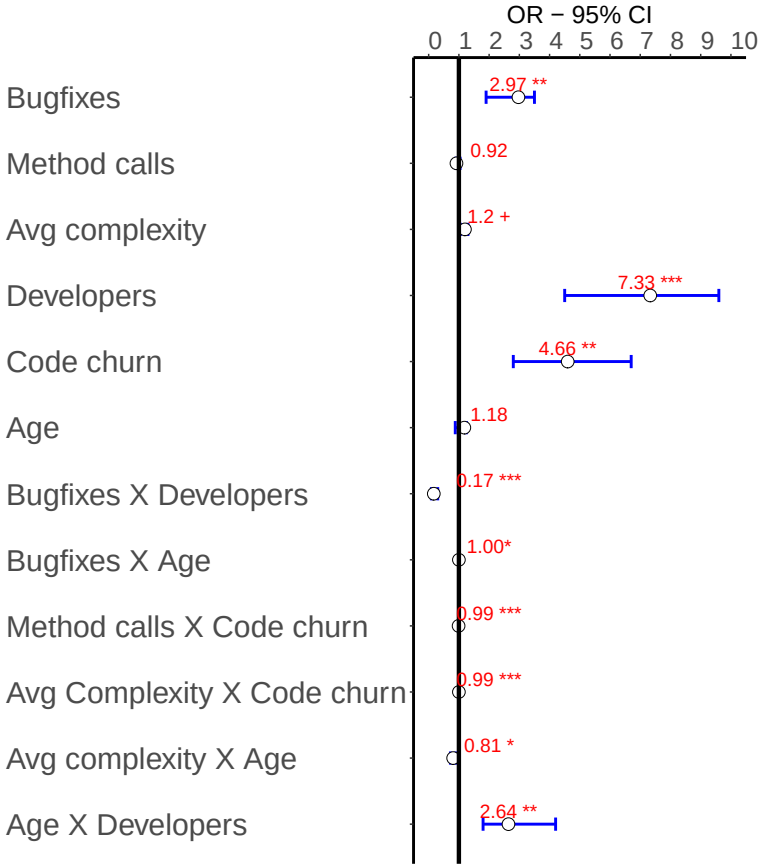


Figure 3.10: Ganymede final model odd ratios and confidence intervals

- Code churn (OR = 4.60): Files that experience cod churn (added+deleted lines) are four times more likely to experience Postrelease faults.

Bugfixes confounder OR is consistent with Europa. Both the Europa and Ganymede models show that the files involved in the bugfixing process have a two to three times higher chance to experience Postrelease faults than files that did not have Bugfixes. In addition, Developers confounder results in the two models are consistent. The Ganymede model demonstrates that having more Developers involved in a file increases the chances for Postrelease bugs by more than seven times. While the Code Churn confounder was not significant with Europa, it was significant in the Ganymede model. Code churn on a file increases the risk (by 4.6 times) for Postrelease bugs. Average Complexity is significant at 90% confidence level in the Ganymede model and it shows that complex files have 20% more chances for Postrelease bugs. Method Calls and Age of files are not statistically significant in the Ganymede model.

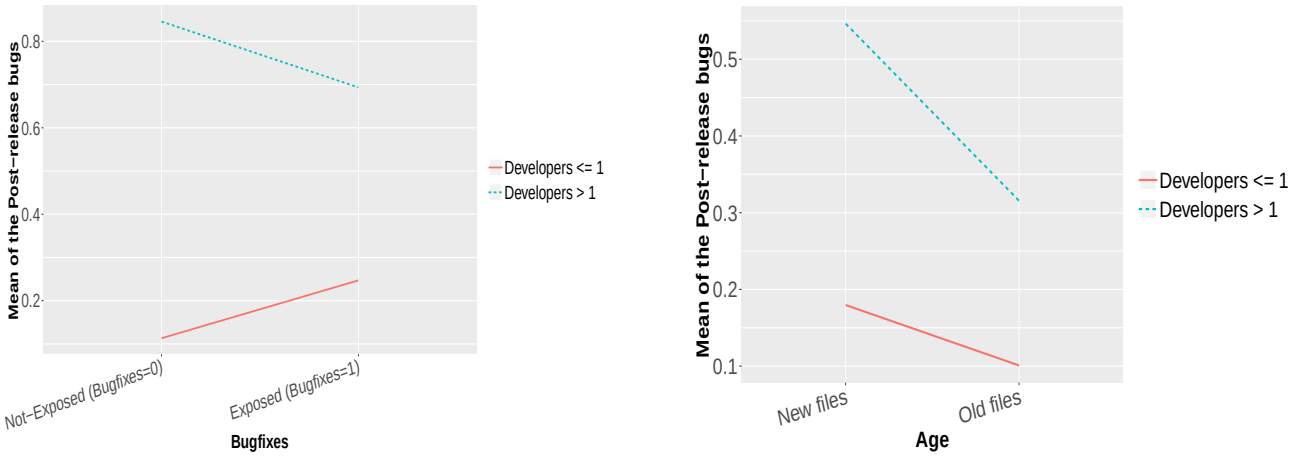
The following list presents the main interactions that can be used to explain Postrelease fault proneness of Europa files:

- Bugfixes with Developers (OR = 0.17): Fewer Developers working on bugfixing reduces the probability of Post-release faults by 83%.
- Developers with Age (OR = 2.64): More Developers working on new files increases the probability of Postrelease bugs by 2.6 times.
- Average Complexity with Age (OR = 0.81): New files with low complexity have 19% less chance to have Postrelease bugs.

The interaction between Bugfixes and Developers showed a low OR (OR=0.17). This is also consistent with the results between the same confounders in the Europa release. Although Bugfixes as a confounder increases the risk of software faults, when Bugfixes interacts with a low number of Developers there is a low chance of software faults as shown in Figure 3.11a. The same figure indicates that a high number of Developers have a high risk of software faults. However, the risk for Postrelease bugs decreases when the high number of developers are working on files exposed to bugfixes.

The interaction between Developer and Age has an OR higher than one (OR=2.64), which is consistent with the Europa model. We learned from Europa model that old files have a lower risk of software faults than new files. However, old files, in both Europa and in Ganymede, have high risk of software faults when there is high number of Developers involved, as shown in Figure 3.11b. Also, a high number of Developers has a higher risk to Postrelease bugs when they work on new files compared to a high number of Developers working on old files.

The interaction between Average Complexity and Age is a new phenomenon in this model. The OR of this interaction is below one (OR = 0.81). It is clear from the Europa model that new files are more likely to have faults. In this model, the results indicate that complex new files have less chance of getting faults than new files with less complexity. Reducing complexity is always recommended at all levels. However, we might need to pay more attention new files with less complexity.



(a) Bugfixes and Developers

(b) Developers and Age

Figure 3.11: Significant interactions of Ganymede releases

3.4 Replicated Study: Using a Case-control Study on Apache Projects

In this section, our case-control approach is applied to explain confounders and their interactions and how they affect software fault proneness on the following Apache projects: Derby, Ant, and Xalan. We follow the same processes explained in Section 3.2 to quantify Odds Ratios OR of all confounders and their interactions, which explains the probability of a specific confounder or interaction of findings Postrelease bugs. Our main motivation in this section is to explore the generalizability by applying the same methodology to different projects. Our results demonstrated similarities and differences among confounders and their interactions across different projects.

The selection of confounders in this section starts with a similar set of confounders from static code and change confounders, including the selection of the exposure (prerelease bugs) and Lines of code (LOC) as a matching confounder. Datasets used in this study are from Apache foundation software, specifically Derby, Ant, and Xalan projects [156, 157, 158].

These datasets were extracted by Mohammad Ahmad, a fellow in the lab, and he used them for another research study [159]. The static code confounders were based on Chidamber and Kemerer Java Metrics CKJM definitions [160]. Confounders of this type have more detailed features related to the complexity, and coupling of codes. To make our work

consistent, we selected confounders similar to those we used in Sections 3.3 and 3.3.2 (i.e., Average Complexity and Method Calls). The closest to these confounders, based on definitions of the confounders [160], are the average methods complexity AMC, and the number of public methods NPM. Note that the change confounders the same as the change confounders we used in the sections 3.3 and 3.3.2 of this chapter.

We applied the correlation test to avoid high-correlated confounders on the same model. We sampled cases from the faulty files group of every release, and we matched those files (i.e., based on LOC) with a similar number of files from the fault-free files. We used one-to-one matching for consistency with the earlier work. However, in some releases we were forced to increase the number of files from the controls because we had an extremely low number of fault-prone files. Then, Condition Indeces/Variance Decomposition Properties CNI/VDP test was applied to diagnose the multicollinearity of the initial model. The normal model is $CNI_1 < 30$ and variances of all confounders and interactions are $VDP < 0.5$. Next, eliminating insignificant interactions is applied following the hierarchical backward fashion. This step was followed by eliminating the confounders step as described in Section 3.2. The outcome of the two processes is the final model (Model_{final}). The last step is to conduct Hosmer Lemshaw HL test for the final models for goodness of fit.

Our primary goal is to find the impact of the exposure and other confounders on Postrelease bugs. Additionally, as we did in the previous work, we intend to observe how the interaction between confounders could significantly affect the Postrelease bugs. We used conditional logistic regression to match our files based on the lines of code. Additionally, the regression quantifies the odds ratios of all confounders and interactions included in our model. The two main research questions to be identified from the work are listed below.

- **RQ1- What are the main confounders that cause an increase or decrease in the Postrelease bugs?**
- **RQ2- What are the main interactions that cause an increase or decrease in the Postrelease bugs?**

3.4.1 Derby Project

Derby is one of the Apache open source software projects [157]. Derby is implemented in Java and used as relational database software. It was first released in 2004 and has evolved with many other releases (see table 3.15). A total of nine releases were extracted from the Derby project, but two releases were excluded from this study, as explained in the next section.

Table 3.15: Release date of Derby releases

Release	Date
Derby 10.1.2.1	18 November 2005
Derby 10.1.3.1	5 July 2006
Derby 10.4.1.3	26 April 2008
Derby 10.5.1.1	01 May 2009
Derby 10.6.1.0	19 May 2010
Derby 10.8.1.2	29 April 2011
Derby 10.8.3.0	29 January 2013

Inclusion and Exclusion

The numbers of faulty files and fault-free files in all Derby releases are shown in Figure 3.12. In some releases, we have much less faulty files compared to fault-free files (e.g., 10.5.1.1). Therefore, we may increase the size of control group to increase the sample size. Figure 3.13 shows that the number of files that were exposed to the bug fix process in the two groups (i.e., case and control) of every release. The 10.1.2.1 and 10.10.1.1 releases were excluded from this study because we cannot use the release with an unbalanced distribution of Bugfixes, that is should have enough files from both the case and control group that have enough observations of the two events (exposed or not exposed to Bugfixes). Next, we present the case-control studies for Derby's release: 10.1.3.1, 10.4.1.3, 10.5.1.1, 10.6.1.0, 10.8.1.2, and 10.8.3.0.

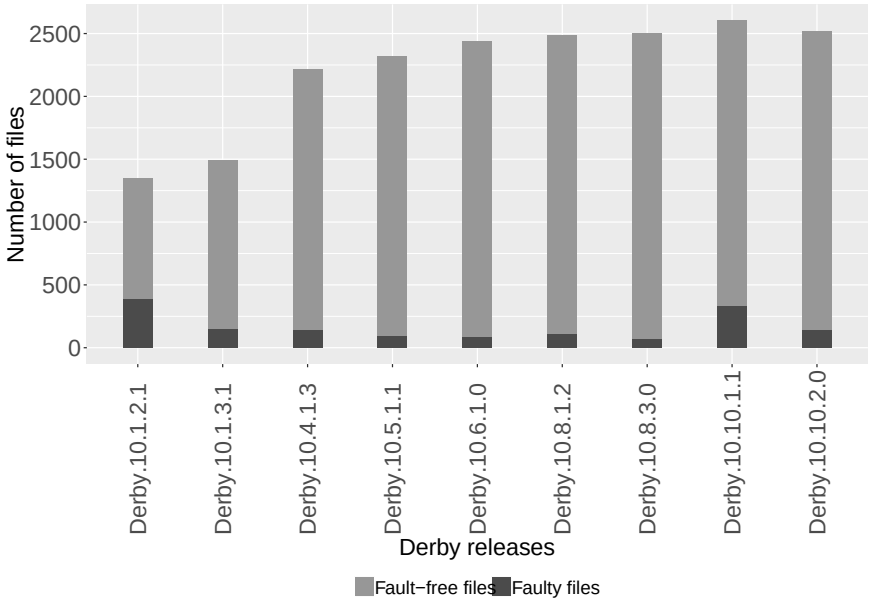


Figure 3.12: The number of faulty files and fault-free files in every release of Derby

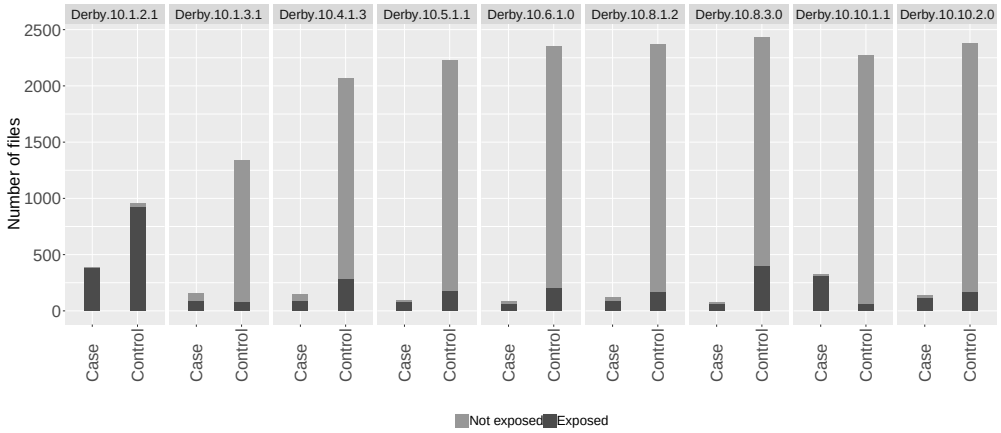


Figure 3.13: Number of files exposed to Bugfixes from cases and controls for every release of Derby

Derby 10.1.3.1

The distribution of the lines of code in the two samples, case and control, is shown in Figure 3.14. We matched every case with one control of the same size. The results of the matches are presented in Table 3.16, as shown by the p-value of the Mann Whitney test (i.e., > 0.05), which indicated a good match. The other basic statistics for the selected confounders are also shown in Table 3.16. As shown, the number of public methods was

higher in the control group than in the case group. The complexity AMC has the same level in two samples. The number of Developers was higher in the case versus control group. The mean of the Bugfixes is slightly higher in the case sample. Age cannot be included because all case samples were new files, and we did not have any old files.

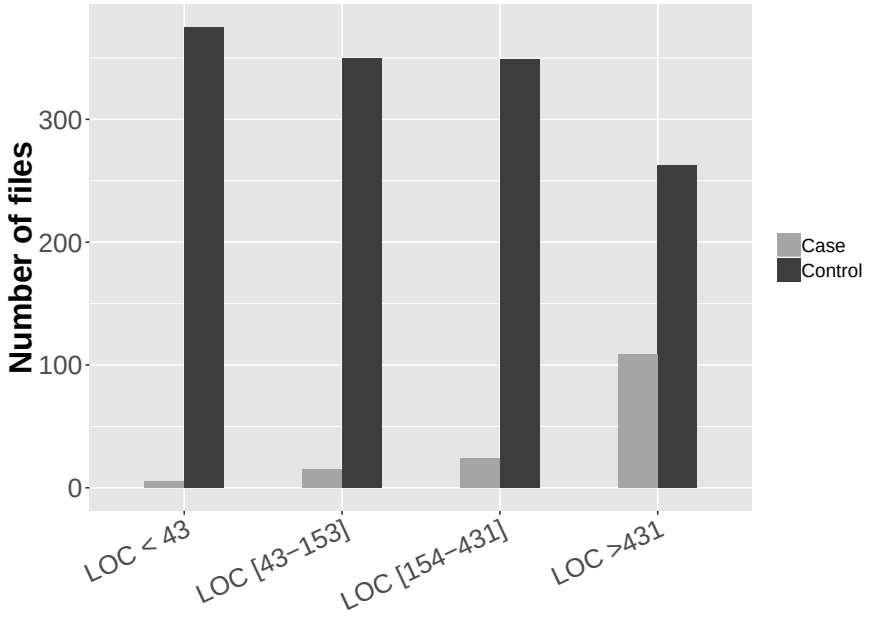


Figure 3.14: Distribution of lines of code in cases and controls of Derby.10.1.3.1

Table 3.16: Descriptive data of case and control groups of Derby 10.1.3.1

	Cases (153 files)					Controls (153 files)					Wilcoxon Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-value
LOC	12	12,120	1,656	885	2,165.2	1	2,1340	1,484	852	2,307.73	0.41
AMC	0	1,840	74.65	35.38	182.33	0	564.6	70.45	40.68	89.24	0.29
NPM	0	178	24.24	10	32.27	0	166	15.32	7.5	21.17	< 0.05
Bugfixes	0	1	0.58	1	0.49	0	1	0.1	0	0.3	< 0.001
Developers	1	7	1.72	1	1.23	0	4	0.34	0	0.74	< 0.001

The correlation test identified a high correlation between the Developers, Age, and Code Churn (see Figure 3.15). Age was excluded because all of the files were new, and this was the reason for the negative correlation of the other confounder. The Developers and Code Churn were tested individually against the response confounders. The Developers showed more significant results than the Code Churn. Therefore, we excluded Code Churn from the initial model. The initial model started with the exposure, three confounders, and six

interactions, as in Equation 3.13. This model was tested for multicollinearity using the CNI/VDP test. Table 3.17 showed the highest condition index CNI was 15.24, which means there was no sign of collinearity. In this case, the full model is the same as the initial model without any change. We can start eliminating interactions from this model.

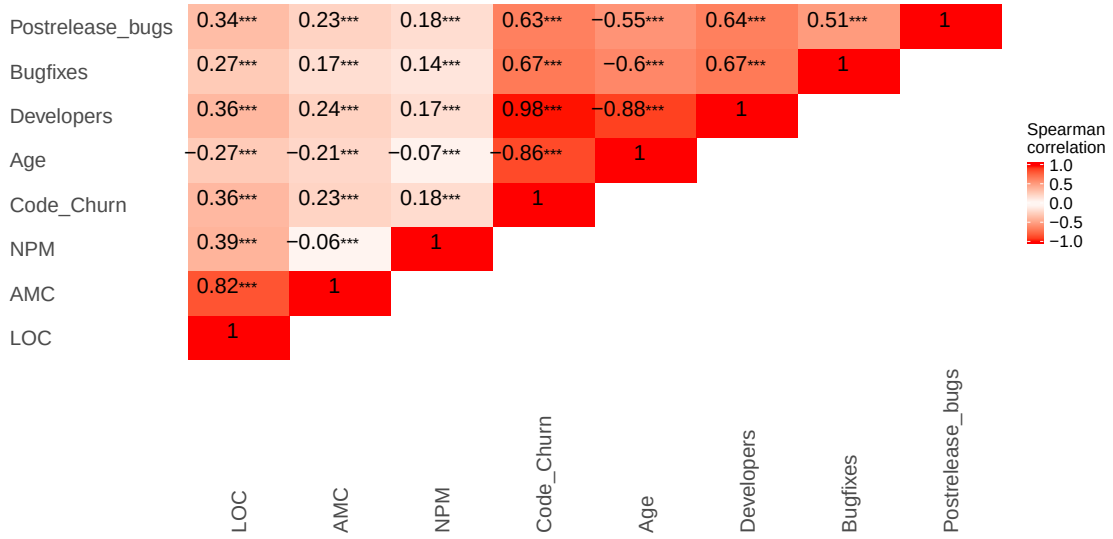


Figure 3.15: The pair-wise correlation test on Derby 10.1.3.1 using the Spearman correlation

Table 3.17: The CNI/VDP collinearity diagnosed for model Derby.10.1.3.1$_0$

CNI ▷▷		15.24	4.13	3.58	2.96	2.50	2.36	2.00	1.57	1.22	1.00	
Bugfixes	β_1	0.00	0.13	0.01	0.27	0.29	0.07	0.03	0.18	0.00	0.01	
NPM	β_2	0.87	0.00	0.03	0.05	0.03	0.00	0.01	0.00	0.00	0.00	
AMC	β_3	0.97	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.01	0.00	
Developers	β_4	0.00	0.23	0.40	0.14	0.10	0.01	0.01	0.09	0.00	0.02	
$Bugfixes \times NPM$	β_5	0.06	0.57	0.01	0.28	0.00	0.02	0.02	0.02	0.00	0.02	
$Bugfixes \times AMC$	β_6	0.13	0.04	0.00	0.12	0.02	0.61	0.00	0.00	0.07	0.00	
$Bugfixes \times Developers$	β_7	0.01	0.00	0.75	0.00	0.11	0.02	0.00	0.01	0.00	0.02	
$NPM \times AMC$	β_8	0.98	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	
$NPM \times Developers$	β_9	0.12	0.69	0.10	0.02	0.03	0.01	0.00	0.01	0.00	0.02	
$AMC \times Developers$	β_{10}	0.02	0.04	0.02	0.02	0.01	0.01	0.77	0.05	0.02	0.01	

△ VDP △

$$
\begin{aligned}
&\text{Derby.10.1.3.1}_0 \\
Y \; &= \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC + \beta_4 \cdot Developers \\
&+ \beta_5 \cdot Bugfixes \times NPM + \beta_6 \cdot Bugfixes \times AMC + \beta_7 \cdot Bugfixes \times Developers \\
&+ \beta_8 \cdot NPM \times AMC + \beta_9 \cdot NPM \times Developers + \beta_{10} \cdot AMC \times Developers
\end{aligned} \tag{3.13}
$$

Three insignificant interactions were eliminated from the full model, as depicted in Table 3.18. The results of $\Delta\chi^2$ at every reduction showed no significant change in the model. The outcome model after eliminating interactions, was shown by Equation 3.14. All confounders had significant interactions except AMC. As a result, we had two possible scenarios: model without change (Derby.10.1.3.1$_G$) or model without AMC (Derby.10.1.3.1$_{Scen2}$). The odds ratios OR of the first and second scenarios were calculated by Equations 3.15 and 3.16, respectively.

Table 3.18: The Derby.10.1.3.1 release model reduction using backward hierarchal elimination for the interactions

Variables	Derby.10.1.3.1$_{full}$	Derby.10.1.3.1$_2$	Derby.10.1.3.1$_3$	Derby.10.1.3.1$_4$
Bugfixes	1.76*	1.81*	1.92**	1.66*
NPM	2.47	2.49	1.19	1.16
AMC	6.89	6.77	2.14	1.55
Developers	18.45***	16.80***	15.81***	15.66***
Bugfixes × NPM	1.70*	1.73*	1.75*	1.51+
Bugfixes × AMC	2.07	2.39	2.67	
Bugfixes × Developers	0.33**	0.33**	0.32***	0.33***
NPM × AMC	6.02	6.16		
NPM × Developers	0.62*	0.60*	0.63*	0.65*
AMC × Developers	1.72			
χ^2	194***	193.7***	193***	189.1***
Likelihood Ratio Test = $\Delta\chi^2$		0.23	0.76	3.83
R^2	0.67	0.67	0.67	0.66
Deviance Explained	93.17%	93.23%	93.11%	83.09%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$				

$$\begin{aligned} &\text{Derby.10.1.3.1}_{G} \\ Y \ &= 0.50 \cdot Bugfixes + 0.02 \cdot NPM + 0.44 \cdot AMC + 2.75 \cdot Developers \\ &+ 0.41 \cdot Bugfixes \times NPM - 1.08 \cdot Bugfixes \times Developers - 0.41 \cdot NPM \times Developers \end{aligned} \tag{3.14}$$

$$\begin{aligned} \text{Derby.10.1.3.1}_{OR*} \ &= EXP(0.50 \cdot Bugfixes + 0.41 \cdot Bugfixes \times NPM \\ &- 1.08 \cdot Bugfixes \times Developers) \end{aligned} \tag{3.15}$$

$$\begin{aligned} \text{Derby.10.1.3.1}_{OR2} \ &= EXP(0.53 \cdot Bugfixes + 0.38 \cdot Bugfixes \times NPM \\ &- 1.11 \cdot Bugfixes \times Developers) \end{aligned} \tag{3.16}$$

Table 3.19 presents the odds ratios and confidence intervals for the two scenarios. We had a total of 306 observations, and we selected 15 observations. The odds ratios and confidence intervals of the two models were not significantly different. The Wilcoxon test was used between the two models concerning the odds ratios or confidence intervals. The Wilcoxon test was used to see whether the distribution resulting from OR and if the CI equations were significant. Wilcox test showed no significant discrepancies between the two models concerning ORs or CIs. The final model was given in Equation 3.17.

Table 3.19: The odds ratio comparison between the Derby.10.1.3.1$_{OR*}$ model and the Derby.10.1.3.1$_{OR2}$ model

Observations	50	51	52	53	54	160	161	162	163	164	250	251	252	253	254
The OR Assessment for the Gold Standard Model and the Model Without AMC															
Derby 10.1.3.1$_{OR*}$	0.80	1.45	0.76	0.83	0.75	0.42	0.41	0.41	0.41	0.41	0.43	0.43	0.41	0.36	0.43
Derby 10.1.3.1$_{OR2}$	0.78	1.55	0.74	0.80	0.73	0.40	0.39	0.39	0.39	0.39	0.41	0.41	0.39	0.35	0.41
Percent difference	3	7	3	4	3	5	5	5	4	5	5	5	5	4	5
Wilcoxon Test	There is no significant difference in the odds ratio between the two models (p-value = 0.2).														
The CI Assessment for the Gold Standard Model and the Model Without AMC															
Derby 10.1.3.1$_{CI*}$	0.16	0.15	0.37	0.23	0.09	0.26	0.23	0.23	0.22	0.21	0.27	0.29	0.21	0.09	0.29
Derby 10.1.3.1$_{CI2}$	0.16	0.14	0.40	0.22	0.09	0.25	0.22	0.22	0.21	0.20	0.26	0.27	0.20	0.09	0.27
Percent difference %	3	2	8	3	1	7	5	5	5	5	6	6	5	3	6
Wilcoxon Test	There is no significant difference between the confidence intervals of the two models (p-value = 0.78)														

$$\begin{aligned} &\text{Derby.10.1.3.1}_{final} \\ Y \ &= 0.50 \cdot Bugfixes + 0.15 \cdot NPM + 2.75 \cdot Developers \\ &\quad + 0.41 \cdot Bugfixes \times NPM - 1.08 \cdot Bugfixes \times Developers - 0.41 \cdot NPM \times Developers \end{aligned} \tag{3.17}$$

Derby 10.4.1.3

The distribution of the LOC in the case and control groups of this release was shown in Figure 3.16. The size of the case group, as shown in Table 3.20, was 146 files, and the number of files in the control group was 292. The results of the Wilcoxon test indicated that LOC was well matched in the two samples. Also, the values of the Bugfixes and Developers were higher in case group versus the control group. Age was not considered in this release because all of the files in case group were new files, that is, old files did not exist. The complexity of control group was higher (both the median and mean) than the case group. The median and the mean of the number of public methods were higher in the case versus the control group. The number of Developers (i.e., maximum, mean, and median) was higher in the case group compared to the control group.

Table 3.20: Descriptive data of cases and controls of Derby 10.4.1.3

	Case (146 files)					Control (292 files)					Wilcox Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-value
LOC	6	14,600	1,543	655	2,263.2	2	16,560	1,528	647	2,343.77	0.94
AMC	0	474.2	46.32	28.65	67.79	0	1,908	82.97	37.64	206.01	< 0.05
NPM	0	150	29.13	12	33.81	0	206	15.94	7	25.64	< 0.001
Bugfixes	0	1	0.83	1	0.36	0	1	0.1	0	0.3	< 0.001
Developers	1	5	0.83	1	0.71	0	3	0.15	0	0.45	< 0.001

The correlation between the Code Churn and Developers was high ($\beta = 0.94$), as shown in Figure 3.17. We treated it as we did in release 10.1.3.1, that is, we excluded Code Churn from the initial model. Additionally, Age was negatively correlated with the Code Churn ($\beta = -0.82$). Another high correlation was detected between LOC and AMC ($\beta = 0.81$).

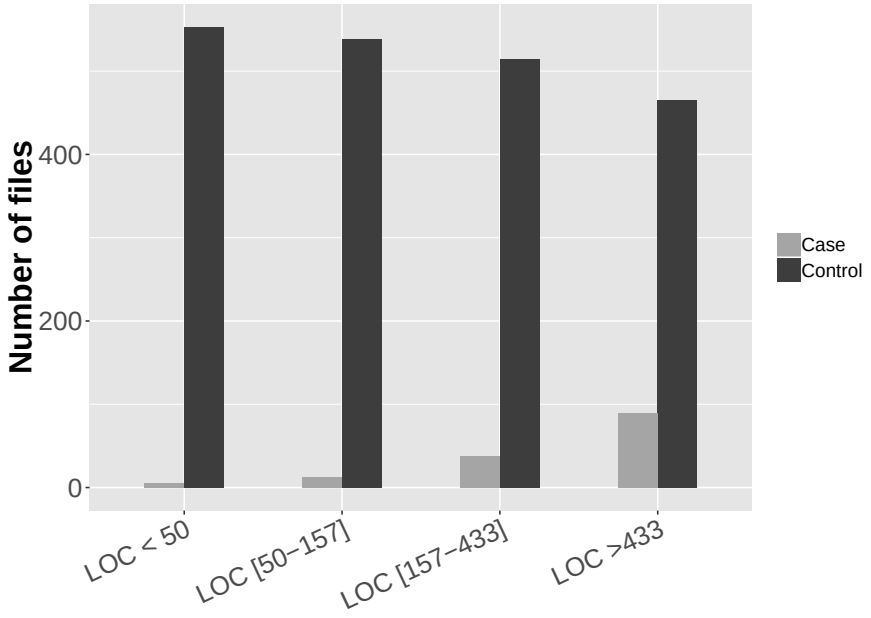
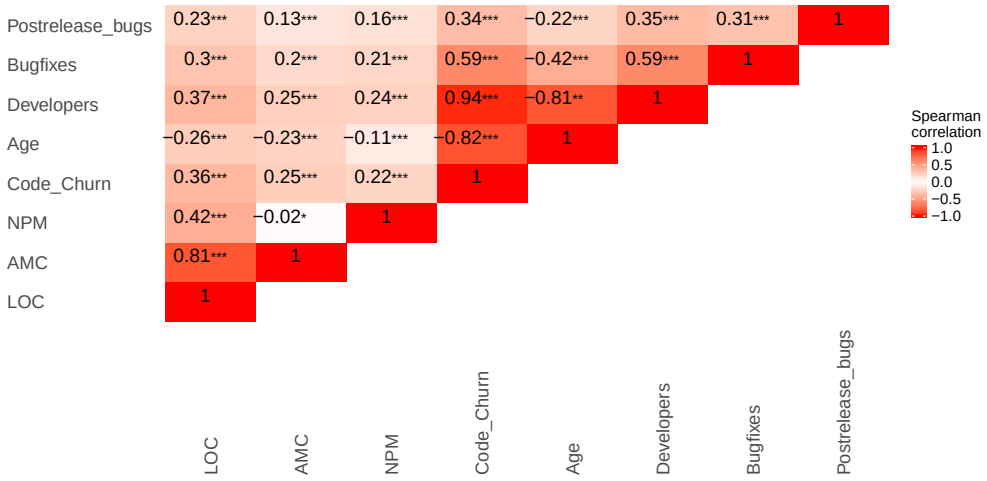


Figure 3.16: The distribution of lines of code in the case and control groups of Derby 10.4.1.3



*** p-value $<$ 0.001 , ** p-value $<$ 0.01 , * p-value $<$ 0.05 , + p-value $<$ 0.1

Figure 3.17: The pair-wise correlation test on Derby 10.4.1.3 using the Spearman test.

The initial model of this release was given by Equation 3.13. Selected confounders for the initial model were Bugfixes, AMC, NPM, and Developers. The multicollinearity test of the initial model (see Table 3.21) showed that no collinearity existed among any of the confounders or interactions.

Table 3.21: The CNI/VDP collinearity diagnosed for model Derby.10.4.1.3$_0$

CNI ▷▷		7.16	5.19	4.37	3.55	2.62	2.38	1.74	1.51	1.13	1.00	
Bugfixes	β_1	0.00	0.00	0.02	0.10	0.67	0.00	0.02	0.16	0.00	0.01	
NPM	β_2	0.54	0.03	0.08	0.03	0.00	0.26	0.01	0.01	0.00	0.01	
AMC	β_3	0.92	0.00	0.02	0.00	0.00	0.01	0.02	0.00	0.02	0.00	
Developers	β_4	0.00	0.10	0.04	0.58	0.17	0.00	0.01	0.07	0.00	0.02	△ VDP △
$Bugfixes \times NPM$	β_5	0.02	0.40	0.37	0.02	0.01	0.14	0.01	0.02	0.00	0.02	
$Bugfixes \times AMC$	β_6	0.10	0.46	0.25	0.00	0.00	0.00	0.16	0.01	0.02	0.00	
$Bugfixes \times Developers$	β_7	0.01	0.11	0.02	0.58	0.18	0.00	0.00	0.02	0.00	0.02	
$NPM \times AMC$	β_8	0.79	0.14	0.02	0.00	0.00	0.01	0.02	0.01	0.01	0.00	
$NPM \times Developers$	β_9	0.05	0.40	0.40	0.08	0.01	0.02	0.01	0.01	0.00	0.02	
$AMC \times Developers$	β_{10}	0.17	0.47	0.31	0.01	0.00	0.00	0.01	0.00	0.03	0.00	

The elimination of the insignificant interactions was presented in Table 3.22. The NPM with AMC was the first interaction dropped from the model, and $\Delta\chi^2 = 0.2$ is not significant. The process continued until we dropped the fifth interaction which was between AMC and Developers. Only one interaction was retained in the model: Bugfixes with Developers. The model (Derby.10.4.1.3$_6$) was the outcome of the previous process, and we called it the gold standard model (Derby.10.4.1.3$_G$) (see Equation 3.18).

Table 3.22: The Derby 10.4.1.3 release model reduction using backward hierarchal elimination for the interactions

Variables	Derby.10.4.1.3$_{full}$	Derby.10.4.1.3$_2$	.....	Derby.10.4.1.3$_5$	Derby.10.4.1.3$_6$
Bugfixes	1.56**	1.55**		1.67***	1.70***
NPM	1.34	1.20		1.42*	1.47*
AMC	0.55	0.45		0.70	1.02
Developers	8.51***	8.59***		7.94***	7.55***
Bugfixes × NPM	0.91	0.90			
Bugfixes × AMC	0.67	0.66			
Bugfixes × Developers	0.26***	0.26***		0.27***	0.28***
NPM × AMC	1.40				
NPM × Developers	1.25	1.28			
AMC × Developers	2.53	2.60		1.40	
χ^2	175.4***	175.2***		172.5***	170.3***
$\Delta\chi^2$		0.20		1.37	2.19
R^2	0.49	0.49		0.48	0.48
Deviance Explained	91.19%	91.16%		91.12%	90.82%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$					

The only insignificant confounder in the model was the AMC. This confounder had no significant interaction in the model. Therefore, there were two possible scenarios. The first was to keep the model without any change (i.e., the gold standard model). The second scenario was to eliminate the AMC. To examine the differences between the gold standard model and the second scenario model, we calculated the odds ratios and confidence intervals of the two models using Equations 3.19 and 3.20.

$$\begin{aligned} &\text{Derby.10.4.1.3}_{G} \\ Y \ &= 0.53 \cdot Bugfixes + 0.38 \cdot NPM + 0.02 \cdot AMC + 2.02 \cdot Developers \\ &\quad - 1.24 \cdot Bugfixes \times Developers \end{aligned} \tag{3.18}$$

The results of the OR and CI assessment were presented in Table 3.23. Most of the ORs had an exact match, and some cells had differences of 3%. The Wilcoxon test between the two samples of OR indicated that there are not statistically significant. Similarly, the confidence intervals had either an exact match or lower values in the reduced model. The Wilcoxon test showed of the CI samples were not significantly different. Therefore, AMC was eliminated without causing any discrepancy in the model. So, the final model was given by Equation 3.21.

$$\text{Derby.10.4.1.3}_{OR*} = EXP(0.53 \cdot Bugfixes - 1.24 \cdot Bugfixes \times Developers) \tag{3.19}$$

$$\text{Derby.10.4.1.3}_{OR2} = EXP(0.53 \cdot Bugfixes - 1.25 \cdot Bugfixes \times Developers) \tag{3.20}$$

$$\begin{aligned} &\text{Derby.10.4.1.3}_{Final} \\ Y \ &= 0.53 \cdot Bugfixes + 0.38 \cdot NPM + 2.02 \cdot Developers - 1.25 \cdot Bugfixes \times Developers \end{aligned} \tag{3.21}$$

Table 3.23: Odds ratio comparison between the Derby.10.4.1.3$_{OR*}$ model and the Derby.10.4.1.3$_{OR2}$ model

Observations	50	51	52	53	54	160	161	162	163	164	250	251	252	253	254
The OR Assessment of the Gold Standard Model and the Model Without AMC															
Derby 10.4.1.3$_{OR*}$	0.56	3.07	3.07	3.07	0.97	0.33	0.33	0.33	0.33	0.56	0.97	0.56	0.56	0.56	0.56
Derby 10.4.1.3$_{OR2}$	0.56	3.08	3.08	3.08	0.97	0.32	0.32	0.32	0.32	0.56	0.97	0.56	0.56	0.56	0.56
Percent difference %	0	0	0	0	0	3	3	3	3	0	0	0	0	0	0
Wilcox Test	There is no significant difference in the odds ratio between the two models (p-value = 0.45)														
The CI Assessment of the Gold Standard Model and the Model Without AMC															
Derby 10.4.1.3$_{CI*}$	0.16	1.70	1.70	1.70	0.65	0.03	0.03	0.03	0.03	0.16	0.65	0.15	0.15	0.15	0.15
Derby 10.4.1.3$_{CI2}$	0.15	1.65	1.65	1.65	0.64	0.03	0.03	0.03	0.03	0.15	0.64	0.15	0.15	0.15	0.15
Percent difference %	6	3	3	3	3	2	0	0	0	0	6	2	0	0	0
Wilcox Test	There is no significant difference between the confidence intervals of the two models (p-value = 0.45)														

Derby 10.5.1.1

The faulty files sample had 93 files, and the fault-free files sample had 186 files in this release (Table 3.24). The amount of data we extracted allowed us to sample for every file from case group two matched files from the control group. Lines of code were well matched in the two samples as shown by the Wilcoxon test result. The complexity of codes was higher in the control group (median = 37.64) compared to the case group (median = 28.65). The number of public methods was higher for the case group than for the control group. The files in the case group had more exposure to the Bugfixes than the files in the control group. Age still had the same issue, new files only existed in the case group, so we did not add it to the initial model.

Table 3.24: The descriptive data of the case and control groups of Derby 10.5.1.1

	Case (93 files)					Control (186 files)					Wilcox Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-Value
LOC	6	14,600	1,543	655	2,263.2	2	16,560	1,528	647	2,343.77	0.94
AMC	0	474.2	46.32	28.65	67.79	0	1,908	82.97	37.64	206.01	< 0.05
NPM	0	150	29.13	12	33.81	0	206	15.94	7	25.64	< 0.001
Bugfixes	0	1	0.83	1	0.36	0	1	0.1	0	0.3	< 0.001
Age	0.42	26.14	16.35	18.14	9.11	0.42	194.8	132.6	170.4	62.92	< 0.001

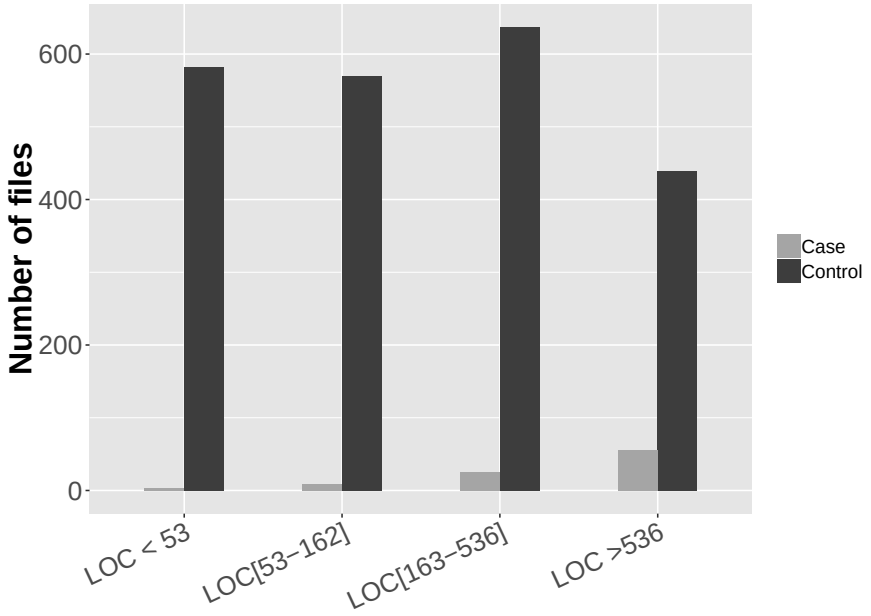


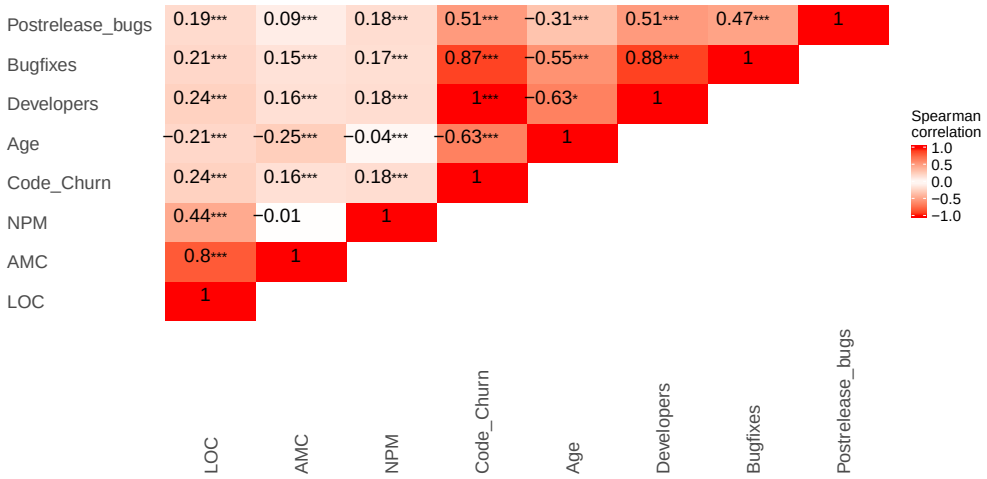
Figure 3.18: The distribution of the lines of code in the case and control groups of Derby 10.5.1.1

Figure 3.19 showed that Age was negatively correlated with all other variables. The maximum number of weeks in the case sample was 26 weeks, whereas most of the files in the control group were older than 53 weeks. Code churn and Developers were highly correlated with the exposure, as shown in Figure 3.19. In the model, the Age of the case group was coded as 1, which may introduce errors in fitting the model. Therefore, the three confounders were excluded from the initial model.

The initial model (Equation 3.22) included the exposure, NPM, and AMC with their interactions. The first step was to run the initial model, Derby.10.5.1.1$_0$, in the CNI-VDP to diagnose the collinearity in the model. The results in Table 3.25 showed that the highest conditioning index was 6.87, which indicated no collinearity was detected. Therefore, we took the same model to the next step, the full model Derby.10.5.1.1$_{full}$.

Derby.10.5.1.1$_0$

$$\begin{aligned} Y &= \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC + \beta_4 \cdot Bugfixes \times NPM \\ &\quad + \beta_5 \cdot Bugfixes \times AMC + \beta_6 \cdot NPM \times AMC \end{aligned} \tag{3.22}$$



*** p-value < 0.001 , ** p-value < 0.01 , * p-value < 0.05 , + p-value < 0.1

Figure 3.19: The pair-wise correlation test on Derby 10.5.1.1 using the Spearman test

Table 3.25: CNI/VDP collinearity diagnosed for the model Derby.10.5.1.1$_0$

CNI ▷▷		6.87	3.83	1.86	1.75	1.38	1.00	
Bugfixes	β_1	0.02	0.09	0.02	0.59	0.14	0.00	△ VDP △
NPM	β_2	0.64	0.02	0.06	0.14	0.10	0.00	
AMC	β_3	0.80	0.18	0.00	0.00	0.01	0.02	
$Bugfixes \times NPM$	β_4	0.02	0.03	0.68	0.01	0.19	0.00	
$Bugfixes \times AMC$	β_5	0.04	0.92	0.00	0.01	0.00	0.03	
$NPM \times AMC$	β_6	0.94	0.04	0.00	0.00	0.00	0.01	

The first eliminated interaction was Bugfixes with AMC (see Table 3.26), with no significant change ($\Delta\chi^2 = 0$). Additionally, the second elimination was Bugfixes and NPM, with no significant change ($\Delta\chi^2 = 0.52$). As a result of eliminating the interactions process, we had the third model (Derby.10.5.1.1$_3$) as the final model. In this model, NPM was the only insignificant confounder in the model. This confounder was not eliminated because it was significantly interacted with AMC. The final model is represented in Equation 3.23.

Derby.10.5.1.1$_{Final}$

$$Y = 1.82 \cdot Bugfixes - 0.17 \cdot NPM - 1.36 \cdot AMC - 1.99 \cdot NPM \times AMC \tag{3.23}$$

Table 3.26: Derby 10.5.1.1 release model reduction using backward hierarchal elimination for the interactions

Variables	Derby.10.5.1.1$_{full}$	Derby.10.5.1.1$_2$	Derby.10.5.1.1$_3$
Bugfixes	6.23***	6.22***	6.19***
NPM	0.80	0.81	0.84
AMC	0.21+	0.22+	0.25+
Bugfixes × NPM	1.16	1.15	
Bugfixes × AMC	1.01		
NPM × AMC	0.10+	0.10+	0.13+
χ^2	160.8***	160.8***	160.2***
$\Delta\chi^2$		0.00	0.52
R^2	0.65	0.65	0.65
Deviance explained	52.58%	52.47%	45.71%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$			

Derby 10.6.1.0

The distribution of the LOC of the files in the case and control groups of this release is shown in Figure 3.20. The sample size of the case group in Derby 10.6.1.0 is 84 files and there is twice this number in the control group (see Table 3.27). The Wilcoxon test showed the two samples are matched well in terms of the LOC. The number of public methods was higher in the case group with a median = 19.5 compared to NPM median = 10 in the control group. The average method complexity AMC was higher in the control group with a median = 54 versus the AMC of the case group (median = 35.61). The Bugfixes occurred more in the case group (mean = 0.76) than in the control group (mean = 0.19). More Developers were shown in the case group (mean = 1.6) higher than in the control group (mean = 0.41). All faulty files were new (< 53 weeks). Age is excluded and cannot be used as a confounder in the initial model.

Code churn is still highly correlated with exposure in Derby 10.6.1.0 as shown in Figure 3.21. The Developers correlation with exposure is moderate. Code churn is excluded from the initial model because of the high correlation with exposure. The initial model includes confounders and interactions as shown in Equation 3.24.

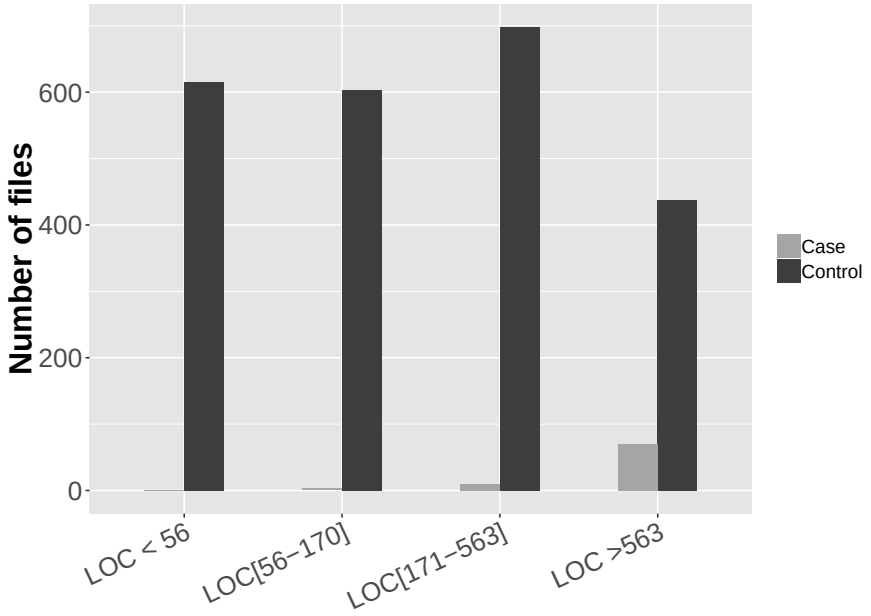


Figure 3.20: The distribution of the lines of code in the case and control groups Derby 10.6.1.0

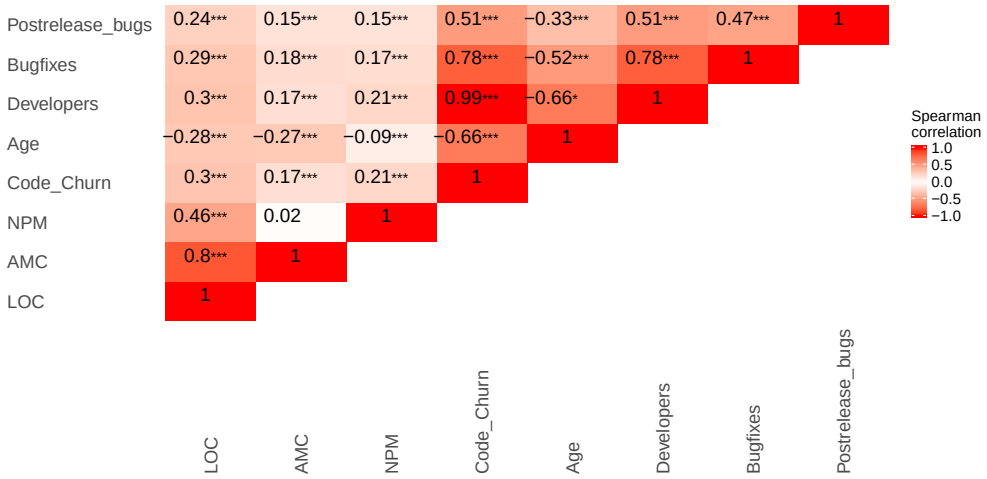
Table 3.27: The descriptive data of the case and control groups of Derby 10.6.1.0

	Cases (84 files)					Controls (168 files)					Wilcox Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-value
LOC	44	17,390	3,197	1,656	3,905.24	1	21,100	2,780	1,392	3,458.97	0.63
NPM	0	206	34.71	19.5	41.12	0	855	26.08	10	69.73	< 0.001
AMC	0	1,662	84.49	35.61	201.3	0	3,956	135.4	54	363.53	< 0.05
Bugfixes	0	1	0.76	1	0.42	0	1	0.19	0	0.39	< 0.001
Developers	1	6	1.6	1	0.87	0	3	0.41	0	0.68	< 0.001
Age	0.71	27.14	17.53	17.71	7.47	0.71	249.8	137.6	165.2	96.23	< 0.001

Derby.10.6.1.0_0

$$Y = \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC + \beta_4 \cdot Developers + \beta_5 \cdot Bugfixes \times NPM + \beta_6 \cdot Bugfixes \times AMC + \beta_7 \cdot Bugfixes \times Developers \quad (3.24)$$

The CNI/VDP test for collinearity was implemented on the initial model (Derby.10.6.1.0_0). The result of the test is presented in Table 3.28. The highest CNI is 3.47, which is considered low. Based on this result, no interaction should be eliminated, and this model is treated as the full model (Derby.10.6.1.0_{full}).



*** p-value $<$ 0.001 , ** p-value $<$ 0.01 , * p-value $<$ 0.05 , + p-value $<$ 0.1

Figure 3.21: The pair-wise correlation test on Derby 10.6.1.0 using the Spearman correlation

Table 3.28: The CNI/VDP collinearity diagnosed for model Derby.10.6.1.0_0

CNI ▷▷		3.47	2.80	2.32	2.04	1.28	1.10	1.00	
Bugfixes	β_1	0.23	0.42	0.21	0.02	0.07	0.01	0.06	△ VDP △
NPM	β_2	0.00	0.13	0.42	0.23	0.02	0.08	0.00	
AMC	β_3	0.01	0.37	0.09	0.34	0.01	0.08	0.00	
Developers	β_4	0.70	0.15	0.04	0.02	0.03	0.01	0.05	
$Bugfixes \times NPM$	β_5	0.00	0.14	0.43	0.22	0.02	0.08	0.00	
$Bugfixes \times AMC$	β_6	0.00	0.40	0.09	0.32	0.01	0.08	0.00	
$Bugfixes \times Developers$	β_7	0.70	0.13	0.05	0.03	0.04	0.01	0.05	

We started with eliminating interaction of Bugfixes and NPM. The $\Delta\chi^2$ result between the full model and the second model indicates there is no significant change. The third model, Derby.10.6.1.0_3, eliminates the second interaction between the exposure and AMC. The elimination did not cause any significant difference between Derby.10.6.1.0_3 and Derby.10.6.1.0_2 ($\Delta\chi^2 = 2.04$). The only remaining interaction is between the exposure and Developers, which is significant. The model Derby.10.6.1.0_3 is considered as the gold standard model, which has one significant interaction retained as shown in Equation 3.25. We have two insignificant confounders: NPM and AMC. This means there are four possible scenarios: eliminate none of them, eliminate both of them, eliminate NPM, or eliminate AMC.

We start to compare the second and the first scenario. The first scenario is the gold standard model, and odds ratios of this model is represented by Equation 3.26, and the second scenario is represented by Equation 3.27. The coefficients of the two scenarios are equal, which means that their outcomes are equal.

Table 3.29: Derby.10.6.1.0 release model reduction using backward hierarchal elimination for the interactions

Terms	Derby.10.6.1.0$_{full}$	Derby.10.6.1.0$_2$	Derby.10.6.1.0$_3$
Bugfixes	2.41**	2.40**	1.96**
NPM	1.03	0.98	1.02
AMC	0.25	0.24	0.97
Developers	7.92***	7.94***	8.30***
Bugfixes × NPM	1.11		
Bugfixes × AMC	3.35	3.29	
Bugfixes × Developers	0.33**	0.33***	0.32***
χ^2	128.6***	128.3***	126.3***
Likelihood Ratio Test = $\Delta\chi^2$		0.25	2.04
R^2	0.60	0.60	0.59
Deviance explained	73.11%	73.35%	73.41%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$			

Derby.10.6.1.0$_G$

$$\begin{aligned} Y = \; & 0.67 \cdot Bugfixes + 0.02 \cdot NPM - 0.02 \cdot AMC + 2.11 \cdot Developers \\ & - 1.11 \cdot Bugfixes \times Developers \end{aligned} \tag{3.25}$$

Derby.10.6.1.0$_{OR*}$

$$Y = \; exp(0.67 \cdot Bugfixes - 1.11 \cdot Bugfixes \times Developers) \tag{3.26}$$

Derby.10.6.1.0$_{OR2}$

$$Y = \; exp(0.67 \cdot Bugfixes - 1.11 \cdot Bugfixes \times Developers) \tag{3.27}$$

Table 3.30 presents OR and CI assessments between the gold standard model and the model without NPM and AMC. The ORs have an exact match in the two rows, with 0% differences for all observations. The Wilcoxon test between the two samples of OR values have a p-value = 1, which indicates an exact match of the two samples. CIs values have

either exact match or less value (i.e., better) for the reduced model. The Wilcox test p-value indicates no significant difference at a 95% confidence level. Therefore, confounders NPM and AMC can be eliminated without causing any significant difference. The final model of release 10.6.1.0 is represented by Equation 3.28.

Table 3.30: The OR and CI assessments between Derby.10.6.1.0$_{OR*}$ and Derby.10.6.1.0$_{OR2}$

Observations	1	2	3	4	5	101	102	103	104	105	201	202	203	204	205
The OR Assessment Between the Gold Standard Model and the Model Without NPM and AMC															
Derby.10.6.1.0$_{OR*}$	1.76	0.40	0.70	0.40	1.76	0.70	0.28	0.28	0.70	0.28	0.28	1.76	0.28	1.76	0.28
Derby.10.6.1.0$_{OR2}$	1.76	0.40	0.70	0.40	1.76	0.70	0.28	0.28	0.70	0.28	0.28	1.76	0.28	1.76	0.28
Percent difference %	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Wilcoxon Test	There is no significant difference in the odds ratio between the two models (p-value = 1).														
The CI Assessment Between the Gold Standard Model and the Model Without NPM and AMC															
Derby.10.6.1.0$_{CI*}$	2.77	1.90	0.65	1.90	2.77	0.65	0.06	0.06	0.65	0.06	0.06	2.77	0.06	2.77	0.06
Derby.10.6.1.0$_{CI2}$	2.76	1.87	0.65	1.87	2.76	0.65	0.05	0.05	0.65	0.05	0.05	2.76	0.05	2.76	0.05
Percent difference	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Wilcoxon Test	There is no significant difference between the confidence intervals of the two models (p-value = 0.96).														

Derby.10.6.1.0$_{final}$

$$Y = 0.67 \cdot Bugfixes + 2.12 \cdot Developers - 1.11 \cdot Bugfixes \times Developers \tag{3.28}$$

Derby 10.8.1.2

In this release, there were 74 files in the case group and 148 in the control group, as shown in Table 3.31. LOC, AMC, and NPM were well matched as presented by the Wilcoxon test (p-value $>$ 0.05). The distribution was similar in the two samples in terms of the LOC, AMC, and NPM. There were more files exposed to bug fixes in the control sample versus the case sample. All of the files in the case sample were new (i.e., $<$ 53 weeks), and this resulted in eliminating Age from the initial model. Developers and Code Churn were both highly correlated with exposure and each other as shown in Figure 3.23. Therefore, we excluded them both from the initial model.

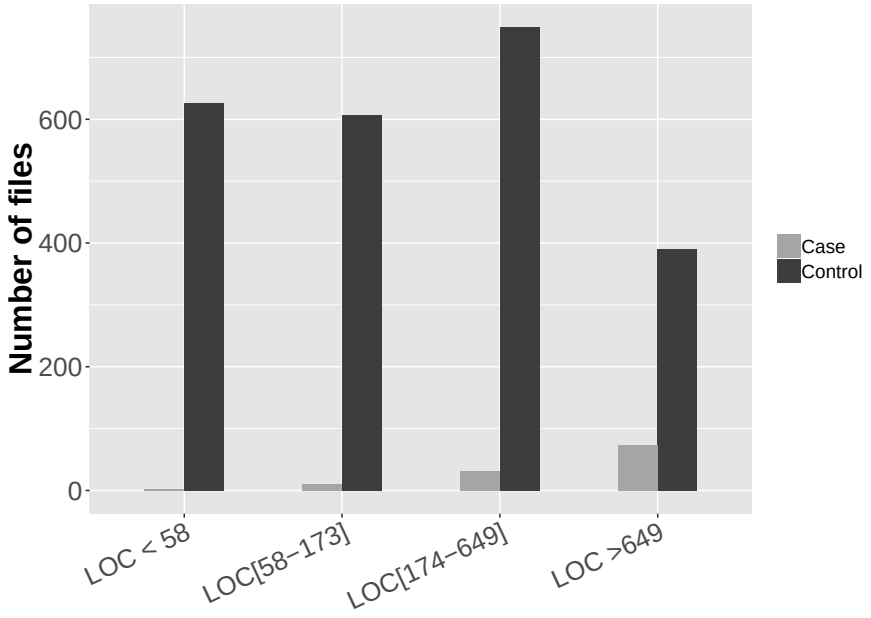


Figure 3.22: The distribution of lines of code in the case and control groups of Derby 10.8.1.2.

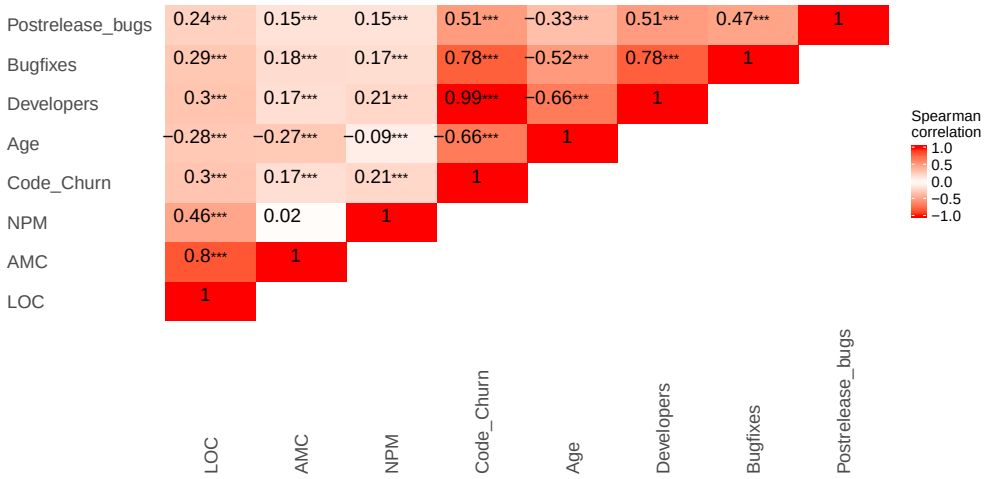
Table 3.31: Descriptive data of cases and controls groups of Derby.10.8.1.2

	Case (116 files)					Control (232 files)					Wilcoxon Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-Value
LOC	31	20,160	2,362	959	3,633.33	1	21,100	2,112	940	3,179.16	0.77
AMC	0.06	1,668	71.55	35.67	175.33	0	3,956	110	47.7	313.18	< 0.05
NPM	0	206	29.97	11.5	40.07	0	1103	22.12	8.5	75.25	< 0.05
Bugfixes	0	1	0.75	1	0.43	0	1	0.15	0	0.35	< 0.001
Age	2.14	24.57	14.97	15.28	6.27	2.57	298.6	189	221.2	105.86	< 0.001

The initial model consisted of the exposure, two static code confounders, and their interactions with the exposure. The initial model needed to be diagnosed from the collinearity using the CNI/VDP test. The results of this test were presented in Table 3.32. It was clear that there were no collinearity issue detected in this model. Therefore, we accepted this model as the full model (Derby.10.6.1.0_{full}) and started the next step.

Derby.10.8.1.2_0

$$\begin{aligned} Y = \; & \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC \\ & + \beta_4 \cdot Bugfixes \times NPM + \beta_5 \cdot Bugfixes \times AMC \end{aligned} \tag{3.29}$$



*** p-value < 0.001 , ** p-value < 0.01 , * p-value < 0.05 , + p-value < 0.1

Figure 3.23: A pair-wise correlation test on Derby 10.8.1.2 using the Spearman correlation.

Table 3.32: The CNI/VDP collinearity diagnosed for model Derby $10.8.1.2_0$

CNI ▷▷		4.21	1.56	1.42	1.27	1.00	
Bugfixes	β_1	0.05	0.00	0.21	0.00	0.02	△ VDP △
NPM	β_2	0.94	0.00	0.00	0.00	0.05	
AMC	β_3	0.01	0.59	0.00	0.40	0.00	
$Bugfixes \times NPM$	β_4	0.94	0.00	0.01	0.00	0.05	
$Bugfixes \times AMC$	β_5	0.00	0.60	0.00	0.39	0.00	

The interactions reduction process started with the Bugfixes and NPM. After eliminating the first interaction, the second interaction became significant, as shown in Table 3.33. The likelihood ratio test ($\Delta\chi^2$) was conducted between the full model and the second model. The result of the test suggested that we can eliminate the interaction without causing any major change to the model. The removal of the second interaction from the second model was not recommended by the same test because it caused a significant change in the likelihood ratio of the model. The final result of this step was one interaction removed from the full model. The second model became the gold standard model ($Derby.10.8.1.2_G$), as shown in Equation 3.30.

NPM was the only confounder that had the chance to be eliminated from the model. So, we had two possible scenarios: one was the model with NPM and the second was the model without NPM. To calculate the odds ratios of the two scenarios, we needed to use the Equations 3.31 and 3.32. The results of OR and CI were presented in Table 3.34. The

Table 3.33: Derby.10.8.1.2 release model reduction using backward hierarchal elimination for the interactions

Variables	Derby 10.8.1.2$_{full}$	Derby 10.8.1.2$_2$
Bugfixes	5.41***	5.69***
NPM	1.15	0.92
AMC	0.14	0.10+
Bugfixes × NPM	0.83	
Bugfixes × AMC	3.94	5.09+
χ^2	137.2***	136.7***
$\Delta\chi^2$		0.45
R^2	0.48	0.48
Deviance explained	59.09%	58.93%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$		

ORs showed an exact match in 9 of the 15 observations. The differences in the ORs did not exceed 2%. The Wilcoxon test result showed that there were no significant differences between them. In all observations, the CIs of the reduced model were either equal to or smaller than the confidence intervals of the gold standard model. This result indicated that the model without NPM was better than the gold standard model. As a result, we eliminated the confounder NPM, and the final model was given as in Equation 3.33.

Derby.10.8.1.2$_G$

$$Y = 1.73 \cdot Bugfixes - 0.08 \cdot NPM - 2.27 \cdot AMC + 1.62 \cdot Bugfixes \times AMC \tag{3.30}$$

Derby.10.8.1.2$_{OR*}$

$$Y = EXP(1.73 \cdot Bugfixes + 1.62 \cdot Bugfixes \times AMC) \tag{3.31}$$

Derby.10.8.1.2$_{OR2}$

$$Y = EXP(1.72 \cdot Bugfixes + 1.55 \cdot Bugfixes \times AMC) \tag{3.32}$$

Derby.10.8.1.2$_{final}$

$$Y = 1.72 \cdot Bugfixes - 2.14 \cdot AMC + 1.55 \cdot Bugfixes \times AMC \tag{3.33}$$

Table 3.34: The OR and CI assessments between *Derby*.10.8.1.2$_{OR*}$ and *Derby*.10.8.1.2$_{OR2}$

Observations	1	2	3	4	5	101	102	103	104	105	201	202	203	204	205
The OR Assessment for the Gold Standard Model and the Model Without NPM															
Derby.10.8.1.2$_{OR*}$	0.39	6.86	0.41	5.42	5.85	0.39	6.85	7.30	7.62	6.54	0.33	0.34	0.35	0.34	0.41
Derby.10.8.1.2$_{OR2}$	0.39	6.89	0.40	5.50	5.92	0.38	6.88	7.31	7.62	6.59	0.33	0.34	0.35	0.34	0.41
Percent difference	0	0	1	2	1	1	1	0	0	1	0	0	0	0	0
Wilcoxon Test	There is no significant difference in the odds ratio between the two models (p-value = 0.95).														
The CI Assessment for the Gold Standard Model and the Model Without NPM															
Derby.10.8.1.2$_{CI*}$	0.03	2.10	0.06	1.33	0.40	0.01	2.06	3.31	4.25	1.28	0.11	0.08	0.06	0.08	0.07
Derby.10.8.1.2$_{CI2}$	0.03	2.04	0.06	1.30	0.39	0.01	2.00	3.20	4.10	1.24	0.11	0.08	0.06	0.08	0.07
Percent difference	0	3	0	2	3	0	3	3	3	3	0	0	0	0	0
Wilcoxon Test	There is no significant difference between the confidence intervals of the two model (p-value = 0.99).														

Derby 10.8.3.0

In this release, the total number of files was 222 files. The total number of faulty files in this release was 74. Therefore, we used them all, and we matched them with 148 files from the control group based on the distribution of the LOC shown in 3.24. The basic statistics of the other confounders were presented in Table 3.35. LOC, AMC, and NPM were all matched and had the same distribution in the two samples with a p-value > 0.05. All the case sample files had the Age below 53 weeks. Therefore, Age was excluded from the initial model of this release.

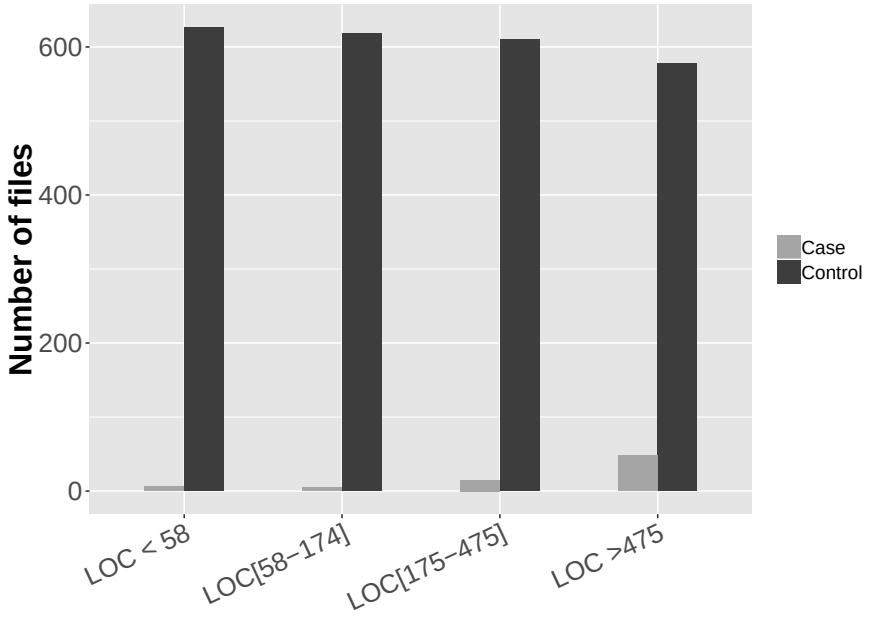


Figure 3.24: The distribution of lines of code in the case and control groups of Derby 10.8.3.0.

Table 3.35: The descriptive data of case and control groups of Derby.10.8.3.0

	Case (74 files)					Control (148 files)					Wilcox Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-value
LOC	9	16,350	1920	924.5	2,835.8	1	189,400	3,270	921.5	15,743.27	0.88
AMC	0	297.1	47.01	34.62	51.35	0	15,780	223.3	40.63	1,341.73	0.11
NPM	0	163	23.27	12	28.41	0	206	20.18	10	29.99	0.38
Bugfixes	0	1	0.85	1	0.35	0	1	0.3	0	0.46	< 0.001
Age	3.14	40.42	21.15	22.36	11.26	0.28	376.6	193.2	232.2	152.59	< 0.001

Developers and Code Churn are still highly correlated with exposure in this release (see Figure 3.25). The initial model is the same one as the previous release (Equation 3.29). The CNI/VDP for the initial model has a collinearity issue caused by the interaction of AMC and NPM. The test is repeated after eliminating AMC × NPM, and the results are shown normally in Table 3.36. Then, the full model is reduced by one interaction, as shown in Equation 3.34.

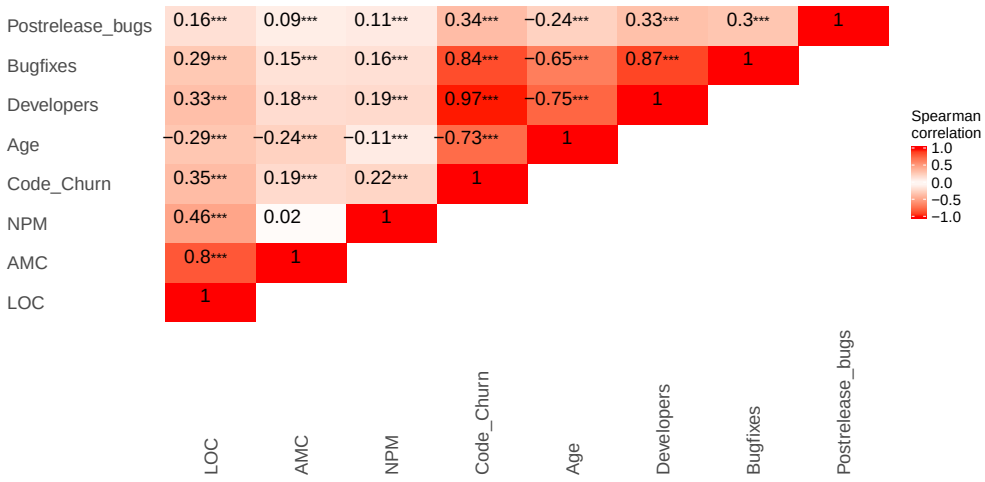


Figure 3.25: A pair-wise correlation test on Derby 10.8.3.0 using the Spearman correlation

$$\text{Derby.10.8.3.0}_0$$

$$Y = \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC + \beta_4 \cdot Bugfixes \times NPM + \beta_5 \cdot Bugfixes \times AMC \quad (3.34)$$

Table 3.36: The CNI/VDP collinearity diagnosed for model Derby.10.8.3.0$_{full}$

CNI ▷▷		10.30	1.93	1.49	1.16	1.00	
Bugfixes	β_1	0.15	0.12	0.38	0.29	0.00	△ VDP △
NPM	β_2	0.01	0.63	0.08	0.06	0.00	
AMC	β_3	0.99	0.00	0.00	0.00	0.01	
$Bugfixes \times NPM$	β_4	0.01	0.62	0.09	0.06	0.00	
$Bugfixes \times AMC$	β_5	0.99	0.00	0.00	0.00	0.01	

In Table 3.37, two interactions were eliminated without affecting the model. The hypothesis was not rejected due to this change in the model. The three remaining confounders in the model are all significant and should be retained in the model. The final model with associated coefficients is shown in Equation 3.35.

Table 3.37: Derby.10.8.3.0 release model reduction using backward hierarchal elimination for the interactions

Variables	Derby.10.8.3.0$_{full}$	Derby.10.8.3.0$_2$	Derby.10.8.3.0$_3$
Bugfixes	4.94***	3.56***	3.66***
NPM	0.79	0.81	0.66+
AMC	0.01	0.01*	0.01*
Bugfixes × NPM	0.74	0.73	
Bugfixes × AMC	19.89		
χ^2	75.52***	75.16***	73.13***
$\Delta\chi^2$		0.36	2.02
R^2	0.43	0.43	0.42
Deviance explained	67.47%	67.31%	68.67%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$			

Derby.10.8.3.0$_{final}$

$$Y = 1.3 \cdot Bugfixes - 0.40 \cdot NPM - 1.06 \cdot AMC \tag{3.35}$$

Goodness of Fit Test for Derby Models and a Discussion of the Results

The results of the HL [147] goodness of fit test for the final models of Derby are presented in Table 3.38. The results of χ^2 of all models indicated good fit with 95% confidence level. Therefore, null hypotheses of all models were not rejected, which means the predicted observations ($\hat{Y}_i$) were not statistically significantly different from the actual values (Y_i).

Table 3.38: Results of goodness of fit test according to Hosmer Lemoshow

Model	HL result χ^2	df	P-Value	H_0	Good fit
Derby.10.1.3.1$_{final}$	0	8	1	Do not reject	Yes
Derby.10.4.1.3$_{final}$	10.45	8	0.23	Do not reject	Yes
Derby.10.5.1.1$_{final}$	2.96	8	0.93	Do not reject	Yes
Derby.10.6.1.0$_{final}$	8.37	8	0.39	Do not reject	Yes
Derby.10.8.1.2$_{final}$	0.45	8	0.99	Do not reject	Yes
Derby.10.8.3.0$_{final}$	0.95	8	0.99	Do not reject	Yes

H_0: The distribution of expected values ($\hat{Y}_i$) is not significantly different from the real values (Y_i).

Figure 3.27 illustrates the final odds ratios of the final models from all Derby releases used in this study. The odds ratios are represented in the figures by white small circles surrounded by lines that correlate lower and upper confidence intervals. The long black vertical lines, that pass at 1 in all graphs, are for distinguishing how the odds ratios deviate above or under the 1.

The main observations for the main confounders of Derby are as follows:

- More files exposed to Bugfixes increase the chances of software faults in all releases used from the Derby project.
- More Developers working on a file increase the chances of software faults in Derby 10.1.3.1, 10.4.1.3, and 10.6.1.0.
- More public methods in a file increases software faults in release 10.4.1.3 and decreases software faults in release 10.8.3.0.

The main observation for the interactions of confounders of Derby is:

- The interaction between Bugfixes and the number of Developers reduces the chance of software faults in Derby releases 10.1.3.1, 10.4.1.3, and 10.6.1.0.

The Bugfixes showed consistent results in all releases of Derby, The results showed that the probability of Bugfixes occurring in the case group was higher than the control group but at different levels. In releases 10.1.3.1, 10.4.1.3, and 10.6.1.0, the probability was between 70% and 96%. In the other releases, the probability was even higher for seeing Bugfixes in

the case group compared to the control group (more than six times in release 10.5.1.1 (OR = 6.19), more than five times in release 10.8.1.2 (OR = 5.59), and more than three times in release 10.8.3.0 (OR = 3.66)). The Bugfixes in Derby is also consistent with what was found in Eclipse Europa (OR = 2.02), and Ganymede (OR = 2.97) releases.

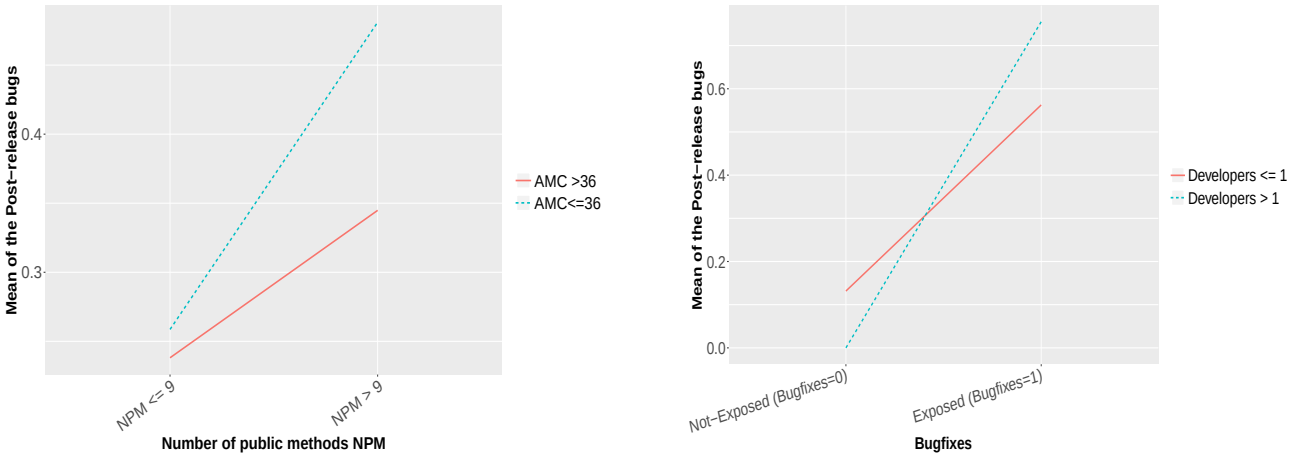
The Developers confounder effect on the Postrelease faults was also consistent for some Derby releases and Eclipse. Specifically, in releases 10.1.3.1, 10.4.1.3, and 10.6.1.0 (as shown in Figures 3.27a, 3.27b, and 3.27d) more Developers worked on the files in case group than in the control group. For example, Developers are eight times more likely to exist in the case group than in the control group in release 10.6.1.0.

The interaction between Bugfixes and Developers was also consistent with our findings in the previous study. In releases 10.1.3.1, 10.4.1.3, and 10.6.1.0, the odds ratios at this interaction were all below one. As shown in Figure 3.26b, more Developers work on files with Bugfixes, the files have a higher chance of Postrelease bugs. When Developers work with files that are not exposed to Bugfixess, the results showed the opposite effect. The risk for Postrelease bugs reduces when a low or high number of developers are working on files that are not exposed to Bugfixes.

Unlike the Eclipse project, static code confounders significantly affected the fault proneness of Derby project. NPM had a higher odds ratio than one (OR = 1.46) in release 10.4.1.3 (see Figure 3.27b). In release 10.8.3.0, the NPM odds ratio was below one (OR = 0.66), as shown in Figure 3.27f. The first result suggests that files with more public methods were 46% more likely to be seen in faulty files. The second result indicated that files with more NPM are 36% less likely to be seen in faulty files. The AMC results were consistent in releases 10.8.1.2 and 10.8.3.0, with odds ratios below one (OR = 0.01 and 0.11). Both suggest that complex files are less likely to have Postrelease faults.

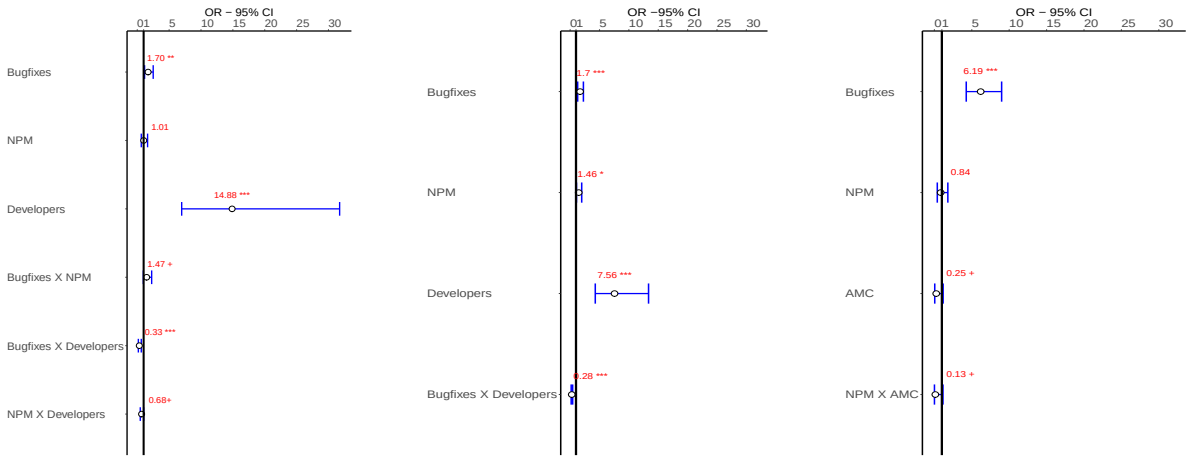
The interaction between AMC and NPM appeared in the 10.5.1.1 model (OR = 0.13), which means their interaction reduced the probability of Postrelease faults. Figure 3.26a explains how they interact with the Postrelease faults in the sample. When NPM is low, the chance of high Postrelease bugs was low, regardless of whether complexity (AMC) was low

or high. However, when the NPM was high (above the median), the chance for Postrelease bugs was high for low complex methods. In other words, when complexity was high, reducing the number of public methods was recommended. Additionally, low number methods was always a good practice whether complexity was low or high.

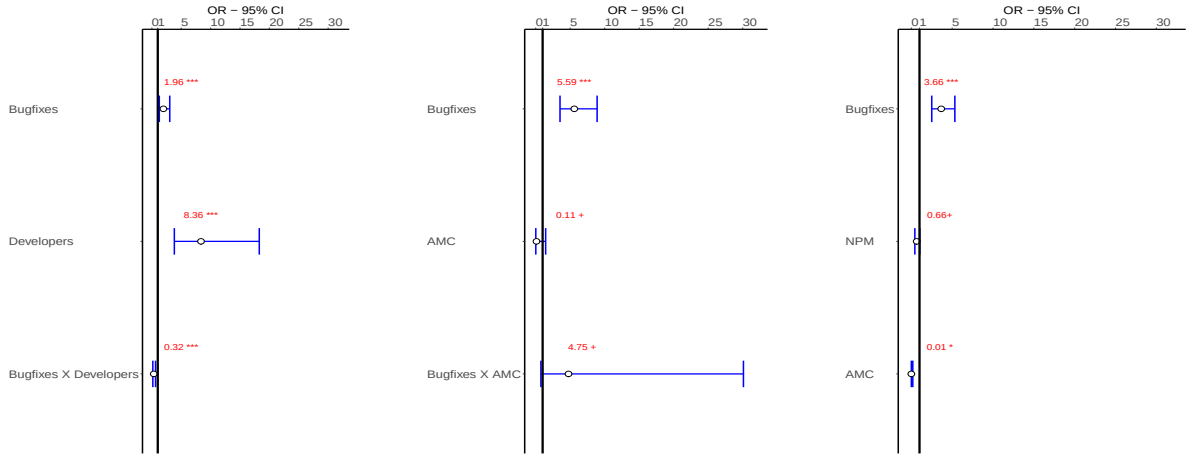


(a) NPM and AMC in 10.5.1.1 (b) Bugfixes and Developers in 10.6.1.0

Figure 3.26: Significant interactions of Derby releases.



(a) Release 10.1.3.1 (b) Release 10.4.1.3 (c) Release 10.5.1.1



(d) Release 10.6.1.0 (e) Release 10.8.1.2 (f) Release 10.8.3.0

Figure 3.27: The final OR of models for Derby releases

3.4.2 Ant Project

The Ant project is a part of the Apache software foundation. It is a Java library and command-line tool used to compile source files and deploy software projects [161] [156]. Ant software been recently praised for its simplicity, portability, and power [161]. Out of the total of seven releases [159], we used only two for our study (i.e., Ant16 and Ant18).

Inclusion and Exclusion

To include a release from the Ant project, we needed to ensure that we had enough faulty files to create the case group and enough of fault-free files to match for the second group (i.e., the control group). Additionally, we had to ensure that the events of the exposure (i.e., Bugfixes) are not severely imbalanced between the case and control groups.

Ant13, 17, and 19 provided too few cases (see Figure 3.28), which were insufficient to create samples. However, Ant14 had more faulty files than fault-free files, which made it difficult to do one-to-one matching. Therefore, Ant13, 14, 17, and 19 were excluded from this study. Next, we examined Bugfixes in the case and control groups for the remaining releases. All files from the control group of Ant 15 were not exposed to Bugfixes, and only a few files from the case group were exposed to Bugfixes, as shown in Figure 3.29. Therefore, Ant 15 was also excluded from this study.

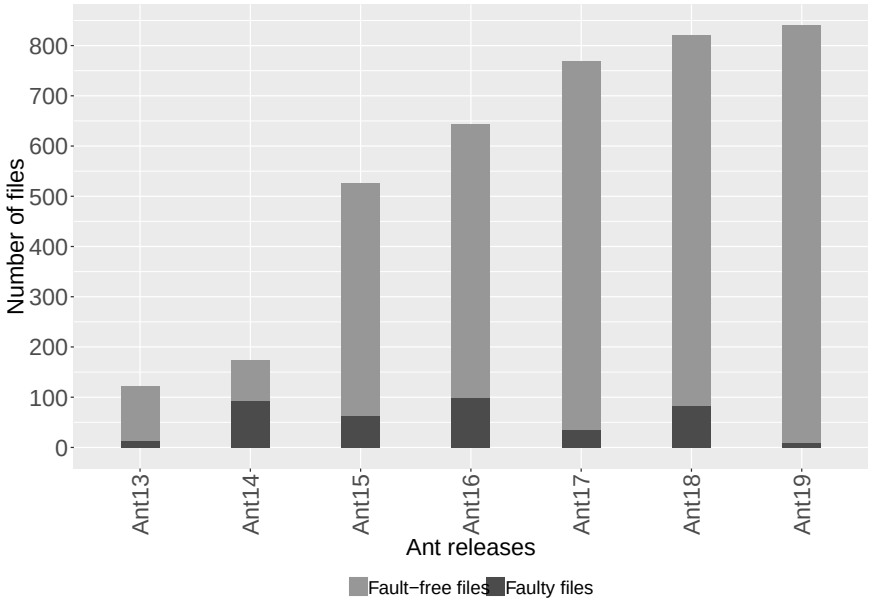


Figure 3.28: The distribution of faulty and fault-free files in the Ant releases

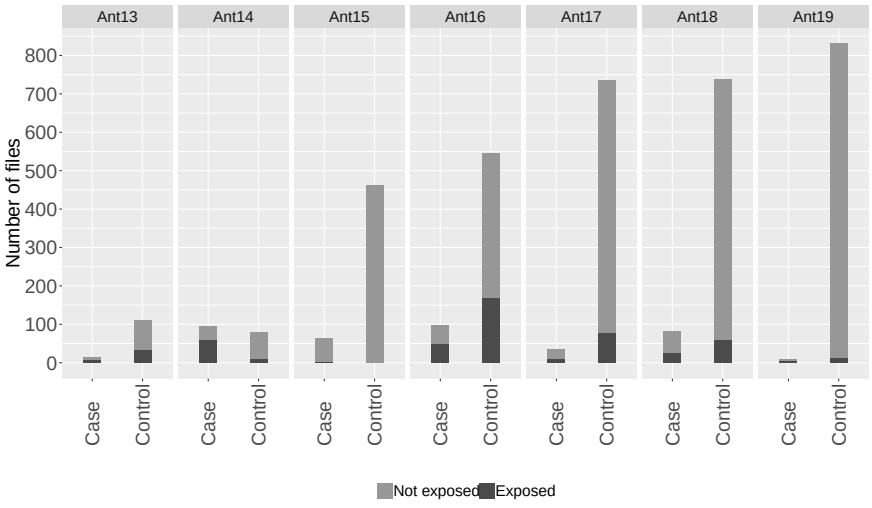


Figure 3.29: The distribution of Bugfixes in files of Ant releases

Ant 16

The matching between the case and control groups for Ant 16 is one to one (i.e., 100 files in cases and 100 in controls). Distribution of the LOC of the two groups is shown in Figure 3.30. The results of the Mann Whitney test of the LOC between the two samples is presented in Table 3.39. The two groups were matched in terms of the LOC, with P-value > 0.05. Bugfixes were also matched in the two samples according to the test result. However, the mean and median of the Bugfixes in the case group were higher than in the control group,

which indicated more files experienced bugfixing the case group than in the control group.All of the files in the case sample are considered new (less than 53 weeks), which means all of the values will be coded as 1. Therefore, Age cannot be included in the initial model of Ant 16.

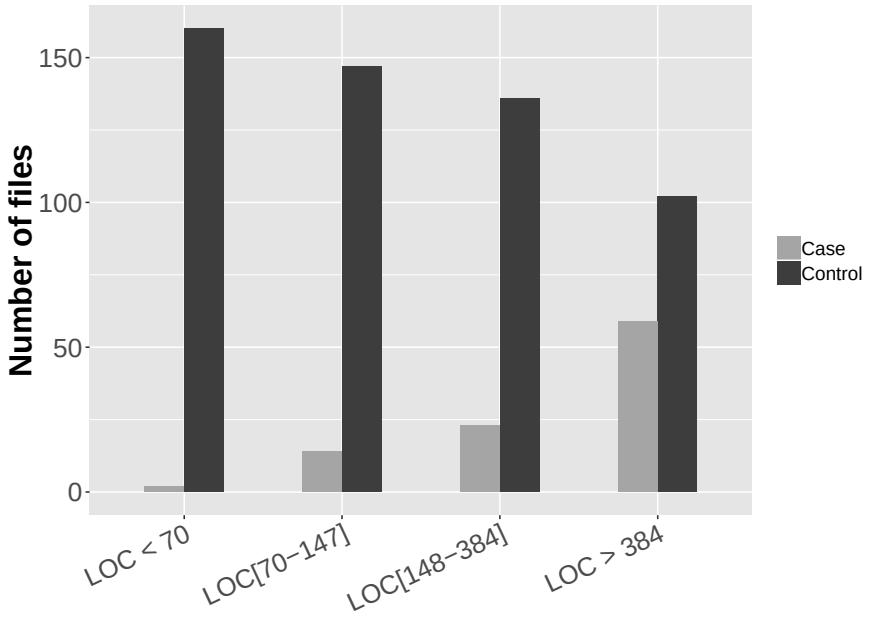
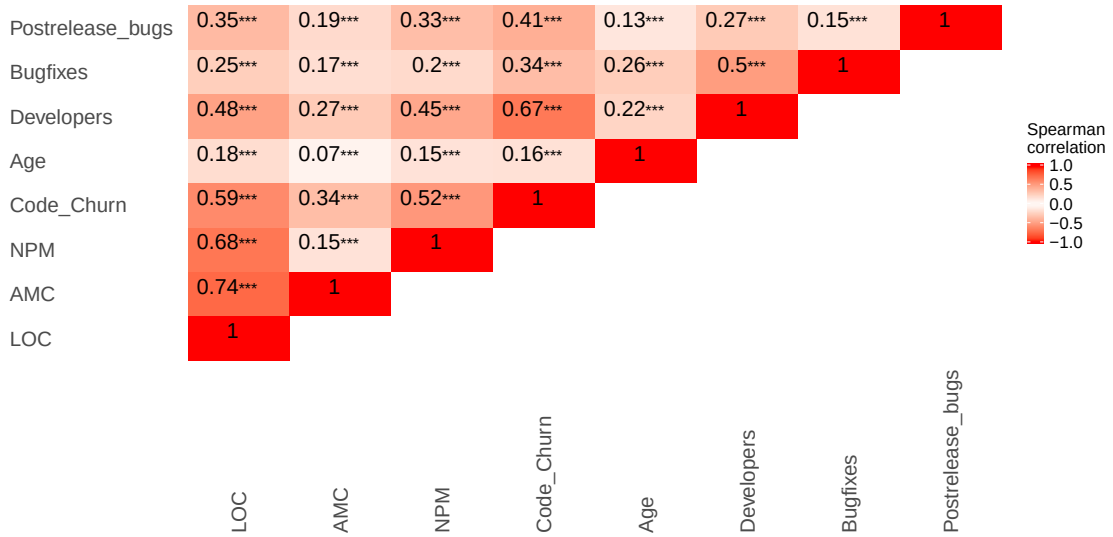


Figure 3.30: Distribution of the Lines of code confounder in cases and controls in Ant16

Table 3.39: The distribution of the lines of code confounder in the case and control groups in Ant 16

	Cases (100 files)					Controls (100 files)					Wilcox Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-value
LOC	24	3,928	663.7	503	625.24	1	4,238	629.2	502	595.31	0.82
AMC	63.43	110	30.14	26.59	18.36	0	2,052	58.18	30.5	204.62	< 0.05
NPM	0	87	15.61	13.5	13.23	0	44	10.84	9	8.62	< 0.01
Bugfixes	0	1	0.51	1	0.5	0	1	0.46	0	0.5	0.52
Developers	1	7	2.77	3	1.38	0	6	2.16	2	1.07	< 0.01
Age	13.71	37.28	29.09	30.78	6.84	20.85	163.7	31.97	24.42	20.49	0.17
Code churn	8	1347	276.7	208	281.81	0	934	103.7	55	149.57	< 0.001

The Spearman correlation test results are shown in Figure 3.31. The high correlation coefficients between LOC and AMC ($\beta > 0.7$) was not a problem because LOC was used in the model as a matching confounder and not as an explanatory confounder. A moderate correlation existed between Code Churn and Developers ($\beta = 0.58$). As a result, no additional confounders were dropped at this stage.



*** p-value < 0.001 , ** p-value < 0.01 , * p-value < 0.05 , + p-value < 0.1

Figure 3.31: A pair-wise correlation test on Ant 16 using the Spearman correlation

The initial model included all confounders except Age. Additionally, the initial model included interactions with exposure only. The number of files in the case sample was 100. The maximum recommended number of terms to include in the initial model was ten terms. So, we decided to keep the interactions with the exposure only, which means our initial model had nine terms, as shown in Equation 3.36. We tested the multicollinearity of the initial model using CNI/VDP. The results showed that the initial model did not have any multicollinearity issues as shown in Table 3.40. The highest CNI ($CNI_1 = 9.45$), which was far below the cut off value (i.e., thirty). All VDPs were below 0.5 which indicated no high variances of all confounders and interactions. As a result, no change was needed for the initial model, and this model was moved to the next step as a full model.

Ant16Model$_0$

$$\begin{aligned} Y \ &= \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC + \beta_4 \cdot Developers + \beta_5 \cdot CodeChurn \\ &+ \beta_6 \cdot Bugfixes \times NPM + \beta_7 \cdot Bugfixes \times AMC + \beta_8 \cdot Bugfixes \times Developers \\ &+ \beta_9 \cdot Bugfixes \times CodeChurn \end{aligned} \tag{3.36}$$

Table 3.40: The CNI/VDP collinearity diagnosed for model Ant16Model$_0$

CNI ▷▷		9.45	3.24	2.79	2.49	2.31	1.75	1.69	1.34	1.00	
Bugfixes	β_1	0.02	0.02	0.16	0.25	0.25	0.00	0.19	0.09	0.01	
NPM	β_2	0.03	0.00	0.26	0.41	0.05	0.16	0.05	0.01	0.03	
AMC	β_3	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
Developers	β_4	0.01	0.15	0.38	0.04	0.29	0.01	0.05	0.04	0.03	△ VDP △
Code churn	β_5	0.00	0.75	0.03	0.00	0.00	0.13	0.04	0.01	0.03	
$Bugfixes \times NPM$	β_6	0.03	0.00	0.26	0.41	0.04	0.16	0.05	0.01	0.03	
$Bugfixes \times AMC$	β_7	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
$Bugfixes \times Developers$	β_8	0.01	0.15	0.38	0.04	0.30	0.01	0.05	0.04	0.03	
$Bugfixes \times Codechurn$	β_9	0.00	0.75	0.03	0.00	0.00	0.13	0.04	0.01	0.03	

Table 3.41: Ant16 release model reduction using backward hierarchal elimination for the interactions

Metrics	Ant16Model$_{full}$	Ant16Model$_2$	Ant16Model$_3$	Ant16Model$_4$
Bugfixes	0.50*	0.50*	0.49*	0.44**I
NPM	1.60	1.60	1.62	1.67
AMC	0.04	0.04	0.04	0.13
Developers	1.55+	1.54+	1.55+	1.51+
Code churn	14.22***	14.32***	14.12***	10.40***
Bugfixes × NPM	1.08	1.07		
Bugfixes × AMC	15.54	15.02	13.09	
Bugfixes × Developers	0.96			
Bugfixes × Code churn	0.13***	0.13***	0.14***	0.18***
χ^2	67.27***	67.25***	67.15***	64.7***
Likelihood Ratio Test = $\Delta\chi^2$		0.01	0.09	2.44
p-value		0.24	0.11	0.12
R^2	0.41	0.41	0.41	0.40
Deviance explained	82.94%	82.94%	82.86%	82.44%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$				

Eliminating interactions started the process to eliminate the Bugfixes with Developers from the full model, as shown in Table 3.41. The removal did not cause any significant change per the result of the likelihood ratio test $\Delta\chi^2 = 0.01$, with a 95% confidence level. The second interaction to be removed without a significant change is Bugfixes with NPM ($\Delta\chi^2 = 0.09$). The final interaction we removed is Bugfixes and AMC ($\Delta\chi^2 = 2.24$).

The outcome of the elimination of interactions is the gold standard model, represented by Equation 3.37. This model contained one interaction between Bugfixes and Code Churn and five confounders. NPM and AMC were the only insignificant confounders, and both had no significant interactions. The four possible outcome scenarios for the final mode were: the gold standard model, model without both NPM and AMC, model without NPM, and model without AMC.

We started the comparison between the first and the second scenario. ORs of the two scenarios were calculated using Equations 3.38 and 3.39. The ORs and CIs of the two scenarios were presented in Table 3.42. All of the observations had differences of less than 10% except for one observation. The differences in the CIs between the two models were also insignificant, and some observations some of the observations showed the same CI between the two models. Based on that, NPM and AMC were eliminated, which produces the final model, represented in Equation 3.40.

$$\begin{aligned}\text{Ant16Model}_G \\ Y = &- 0.81 \cdot Bugfixes + 0.51 \cdot NPM - 2 \cdot AMC + 0.41 \cdot Developers \\ &+ 2.34 \cdot CodeChurn - 1.67 \cdot Bugfixes \times CodeChurn\end{aligned} \tag{3.37}$$

$$\text{Ant16Model}_{OR*} = EXP(-0.81 \cdot Bugfixes - 1.67 \cdot Bugfixes \times CodeChurn) \tag{3.38}$$

$$\text{Ant16Model}_{OR2} = EXP(-0.83 \cdot Bugfixes - 1.56 \cdot Bugfixes \times CodeChurn) \tag{3.39}$$

Table 3.42: The odds ratios comparison between the gold standard model and the first scenario model

Observations	1	2	3	4	5	61	62	63	64	65	121	122	123	124	125
The OR Assessment of the Gold Standard Model and the Model Without NPM and AMC															
Ant16Model$_{OR*}$	1.21	0.74	0.76	0.83	1.18	0.54	6.88	1.92	52.73	0.27	0.73	1.22	0.85	0.69	0.89
Ant16Model$_{OR1}$	1.28	0.7	0.72	0.9	1.25	0.52	6.51	1.98	43.61	0.27	0.69	1.30	0.92	0.76	0.97
Percent difference	6	5	5	9	6	3	5	3	17	1	5	6	9	10	8
The CI Assessment of the Gold Standard Model and the Model Without NPM and AMC															
Ant16Model$_{CI*}$	0.50	0.42	0.40	0.01	0.45	0.51	21.21	1.91	0.53	0.49	0.42	0.53	0.01	0.16	0.05
Ant16Model$_{CI1}$	0.57	0.41	0.40	0.01	0.51	0.51	20.67	2.03	0.54	0.49	0.42	0.59	0.01	0.17	0.06
Percent difference	12	-2	2	0	13	1	-3	7	3	0	0	12	0	3	10

Ant16Model$_{final}$

$$Y = -0.83 \cdot Bugfixes + 0.48 \cdot Developers + 2.32 \cdot CodeChurn - 1.56 \cdot Bugfixes \times CodeChurn \quad (3.40)$$

Ant 18

In this release, one-to-one matching method was applied based on the distribution of the LOC, as seen in Figure 3.32. The LOC was matched in the two samples, as indicated by the Wilcoxon test (p-value = 0.55) of the LOC in Table 3.43. The number of public methods was higher in the case group with a slightly higher median in the case group than in the control group. In general, Bugfixes, NPM, and AMC were matched, with very close mean and median values in the two samples. More Developers were witnessed in the case group than in the control group with a higher maximum value and slightly higher mean.

Table 3.43: Descriptive data of case and control groups for Ant 18

	Case (80 files)					Control (80 files)					Wilcoxon Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-Value
LOC	29	2817	866.3	596	748.69	2	4,569	745.1	636	648.3	0.55
NPM	0	106	19.24	13	19.5	0	62	13.81	10.5	11.5	0.16
AMC	8.5	216	32.97	27.32	26.68	0	2,052	58.03	29.85	221.83	0.35
Bugfixes	0	1	0.31	0	0.46	0	1	0.27	0	0.44	0.54
Developers	1	4	1.43	1	0.66	0	3	0.88	1	0.75	< 0.001
Age	1	82	56.35	64	22.79	1	499.42	164.84	75.71	174.73	< 0.001

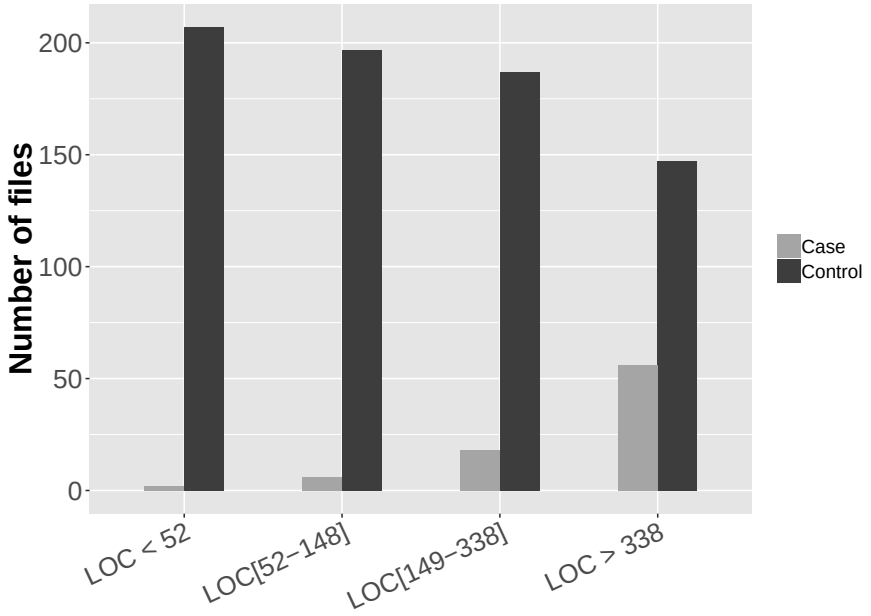
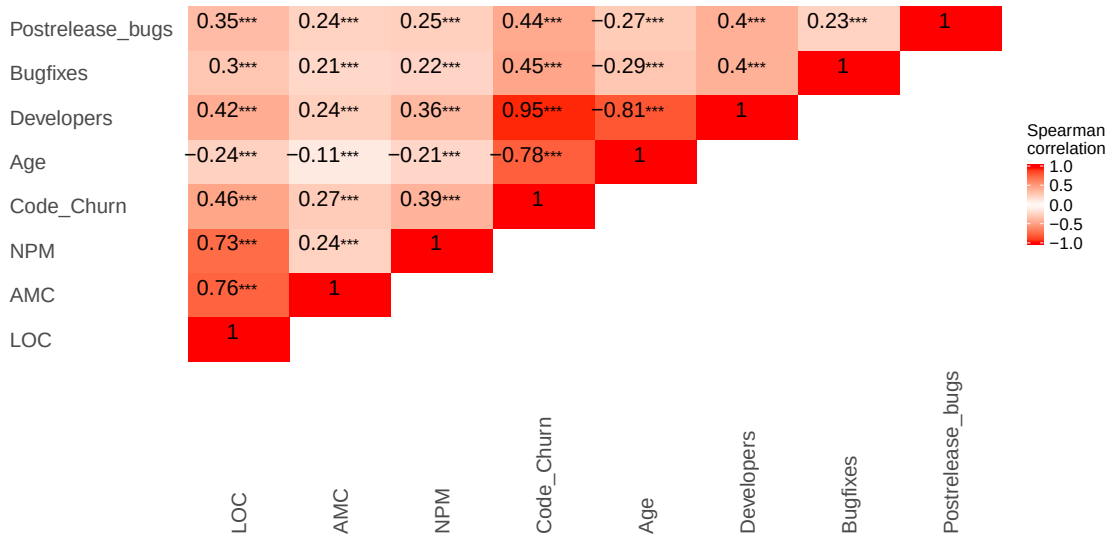


Figure 3.32: The distribution of the Lines of code confounder in the case and control groups in Ant18

The correlation test indicated that Developers was very highly correlated with Code Churn ($\beta = 0.95$) (see Figure 3.33). Therefore, one should be dropped from the initial model. Code churn was also negatively and highly correlated with Age ($\beta = -0.78$). To solve this, we only dropped Code churn since it is correlated both Age and Developers confounders. Age was also excluded because of the imbalanced distribution between old and new files in the two samples.

The initial model (Ant18Model_0) of this release contained the exposure, three confounders, and the interactions between the exposure and other confounders (Equation 3.41). The CNI/VDP test results indicated that no multicollinearity was detected because the highest CNI was below 30 (i.e., $CNI_1 = 10.32$). Even with the high VDPs of AMC and AMC $\times$ Bugfixes, the model was still free of multicollinearity (see Table 3.44). The full model, Ant18Model_{full}, was the same as the initial model.



*** p-value < 0.001 , ** p-value < 0.01 , * p-value < 0.05 , + p-value < 0.1

Figure 3.33: A pair-wise correlation test on Ant18 using the Spearman correlation

$$
\begin{aligned}
& Ant18Model_0 \\
Y \; &= \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC + \beta_4 \cdot Developers \\
&\quad + \beta_5 \cdot Bugfixes \times NPM + \beta_8 \cdot Bugfixes \times AMC + \beta_9 \cdot Bugfixes \times Developers
\end{aligned} \tag{3.41}
$$

Table 3.44: The CNI/VDP collinearity diagnosed for model Ant18Model$_0$

CNI ▷▷		10.32	2.05	1.97	1.54	1.51	1.06	1.00	
Bugfixes	β_1	0.24	0.22	0.05	0.02	0.39	0.07	0.00	△ VDP △
NPM	β_2	0.02	0.00	0.58	0.02	0.21	0.08	0.01	
AMC	β_3	0.99	0.00	0.00	0.00	0.00	0.00	0.01	
Developers	β_4	0.00	0.13	0.55	0.08	0.01	0.07	0.01	
$Bugfixes \times NPM$	β_5	0.06	0.27	0.23	0.28	0.03	0.09	0.01	
$Bugfixes \times AMC$	β_6	0.99	0.00	0.00	0.00	0.00	0.00	0.01	
$Bugfixes \times Developers$	β_7	0.01	0.73	0.02	0.01	0.04	0.08	0.01	

The process to eliminate interactions started with the interaction of Bugfixes and AMC from Ant18Model$_{full}$ as shown in Table 3.45. The second interaction to be removed was Bugfixes and NPM. The interactions did not cause any significant change to the model, as indicated by the likelihood ratio test and a 95% confidence level. The outcome of this process was the gold standard model, presented in Equation 3.42.

Table 3.45: Ant18 release model reduction using backward hierarchal elimination for the interactions

Metrics	Ant18Model$_{full}$	Ant18Model$_2$	Ant18Model$_3$
Bugfixes	0.81	0.92	0.92
NPM	1.33	1.41	1.36
AMC	0.36	0.92	0.89
Developers	2.99**	2.99***	3.04***
Bugfixes × NPM	0.80	0.84	
Bugfixes × AMC	0.25		
Bugfixes × Developers	0.61*	0.59*	0.58*
χ^2	37.01***	36.47***	35.88***
Likelihood ratio test = $\Delta\chi^2$		0.54	0.58
p-value		0.54	0.58
R^2	0.29	0.29	0.28
Deviance explained	89.85%	89.92%	89.10%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$			

$$
\begin{aligned}
&Ant18Model_G \\
Y = &-0.08 \cdot Bugfixes + 0.31 \cdot NPM - 0.10 \cdot AMC + 1.11 \cdot Developers \\
&- 0.54 \cdot Bugfixes \times Developers
\end{aligned}
$$

Two possible confounders can be eliminated from the gold standard model, which left us with four possible scenarios for our final model: no change, remove NPM and AMC, remove NPM, and remove AMC. We started our comparison between the first and the second scenario, and the OR and CI were calculated using Equations 3.42 and 3.43. The OR and CI results were presented in Table 3.46. The ORs and CIs had no meaningful differences between the two scenarios. Most observations had an exact match or less than a 3% difference. The outcome of this process indicated that there was no meaningful difference between the gold standard model and the model without NPM and AMC. Both NPM and AMC should be excluded from the gold standard model, and the final model is shown in Equation 3.44.

$$Ant18_{OR*} = EXP(-0.08 \cdot Bugfixes - 0.54 \cdot Bugfixes \times Developers) \tag{3.42}$$

Table 3.46: The odds ratios and CI assessment between the gold standard model and the model without NPM and AMC in Ant18

Observations	1	2	3	4	5	61	62	63	64	65	121	122	123	124	125
The OR Assessment of the Gold Standard Model and the Model Without NPM and AMC															
$Ant18_{OR*}$	0.98	0.98	0.98	0.98	1.55	0.35	1.55	0.98	1.05	1.55	0.62	0.98	0.62	1.05	0.35
$Ant18_{OR1}$	0.97	0.97	0.97	0.97	1.55	0.35	1.55	0.97	1.07	1.55	0.61	0.97	0.61	1.07	0.35
Percent difference	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
The CI Assessment of the Gold Standard Model and the Model Without NPM and AMC															
$Ant18_{CI*}$	0.36	0.36	0.36	0.36	1.85	1.29	1.85	0.36	0.95	1.85	0.25	0.36	0.25	0.95	1.29
$Ant18_{CI1}$	0.36	0.36	0.36	0.36	1.85	1.27	1.85	0.36	0.98	1.85	0.24	0.36	0.24	0.98	1.27
Percent difference	0	0	0	0	0	-1	0	0	2	0	-1	0	-1	1	-2

$$Ant18_{OR2} = EXP(-0.07 \cdot Bugfixes - 0.55 \cdot Bugfixes \times Developers) \tag{3.43}$$

$$\begin{aligned} &Ant18Model_{final} \\ Y = &-0.07 \cdot Bugfixes + 1.15 \cdot Developers - 0.55 \cdot Bugfixes \times Developers \end{aligned} \tag{3.44}$$

Goodness of Fit Test of Ant Models and Discussion of the Results

The goodness of fit results are presented in Table 3.47 for releases Ant 16 and Ant 18. We conducted the HL test to compare between actual and predicted data using the whole sample. The results of the HL test on the whole sample indicated that both models, Ant18Model$_{final}$ and Ant18Model$_{final}$, were a good fit.

Table 3.47: Goodness of fit test for Ant16Model$_{final}$ and Ant16Model$_{final}$ models according to Hosmer Lemoshow

Model	HL result χ^2	df	p-value	H_0	good fit?
Ant16Model$_{final}$	10.75	8	0.21	Do not reject	Yes
Ant16Model$_{final}$	11.43	8	0.17	Do not reject	Yes

H_0: The distribution between the expected values ($\hat{Y}_i$) and real values (Y_i) are not statistically different.

The main observations for the main confounders of Ant are as follows:

- Files exposed to Bugfixes are less likely to experience Postrelease bugs by 57%.

- More Developers working on a file increases the chance for Postrelease bugs by 61% in Ant16 and by 316% in Ant18.
- More lines added and deleted to files increases the risk for Postrelease bugs by ten times.

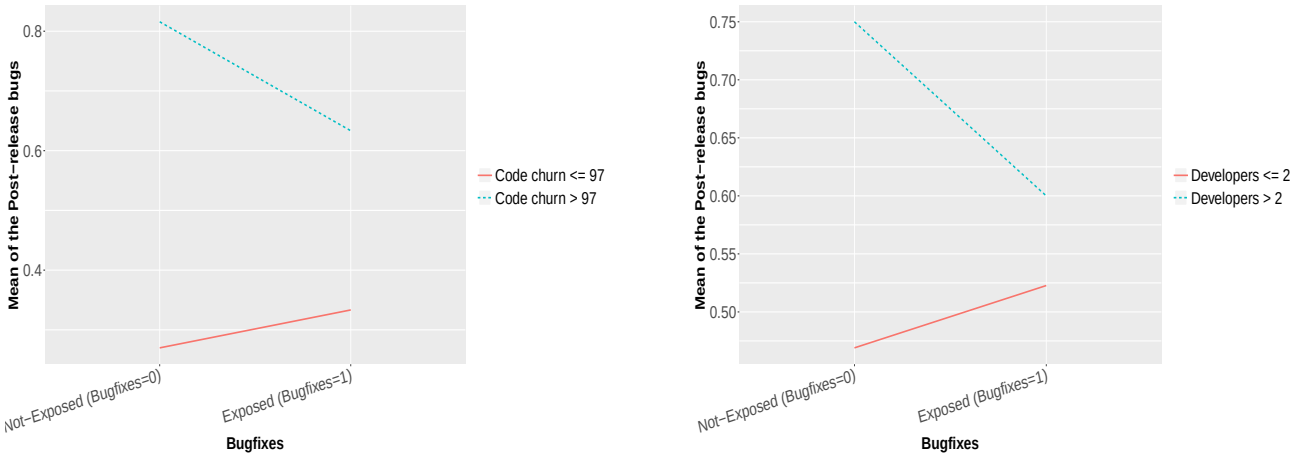
The main observations for the interactions of confounders of Ant are as follows:

- The interaction between Bugfixes and Code Churn decreases the risk of Postrelease bugs by 79% in Ant16.
- The interaction between Bugfixes and the number of Developers decreases the risk of Postrelease bugs by 43% in Ant18.

The final OR and CI of the final models of Ant16 and Ant18 were shown in Figures 3.35a and 3.35b. The exposure was significant in Ant16 but not in Ant18. Bugfixes in Ant16 had an OR below one (OR = 0.43), which means files with prerelease bugs reduce the risk of Postrelease by 57% in Ant16. This was different than the results of the exposure effect on Postrelease bugs found in Derby, Europa, and Ganymede. The Developers OR in Ant 16 was higher (OR = 1.62),which indicated the risk of Postrelease bugs increases by 62% when more Developers worked on a file. In Ant18, more Developers working on a file increases the Postrelease bugs by more than 300%. The Developers results in Ant16 and Ant18 were consistent with the results of Europa, Ganymede and Derby. In Ant16, Code Churn showed a high odds ratio (OR = 10.20). This indicated that files with more modification (lines added and deleted) are ten times more likely to experience Postrelease bugs.

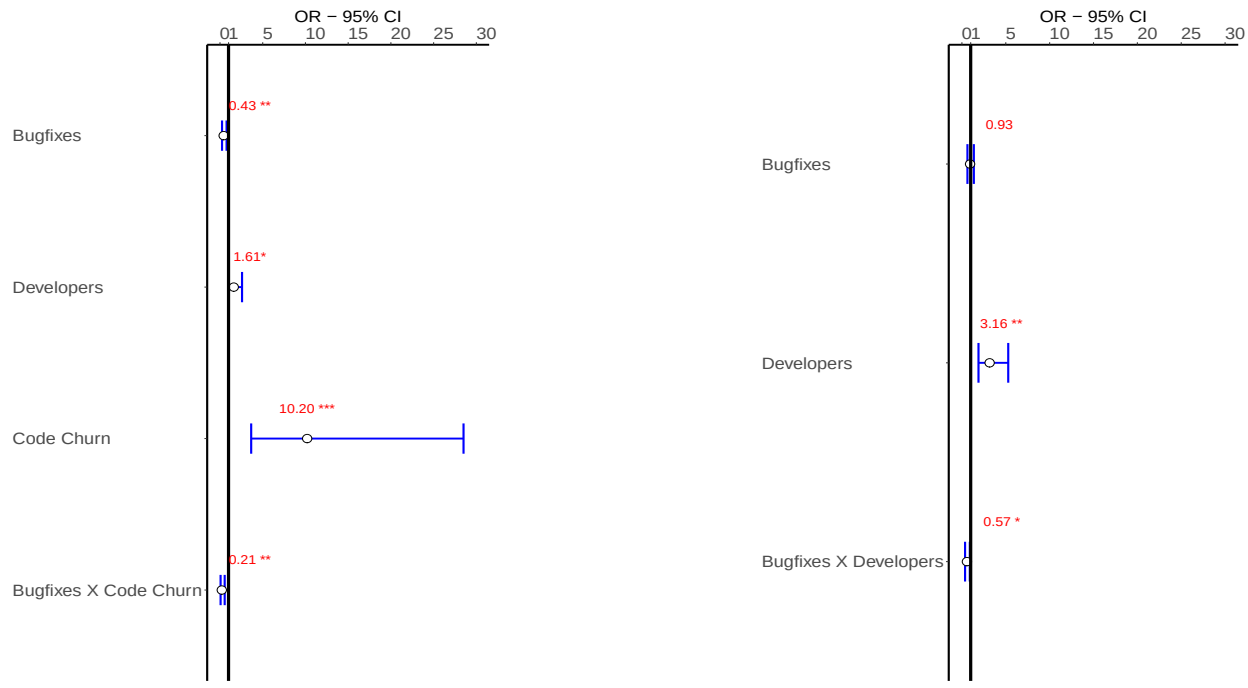
Bugfixes interacted significantly with the Code Churn in Ant 16, which has an OR below one (OR = 0.21). This indicates the interaction decreases Postrelease bugs. Figure 3.34a illustrates the relationship between the two confounders with Postrelease bugs in the sample. More Code Churn (i.e., Code Churn > 97) with Bugfixes is less than a high number for Code Churn without Bugfixes. The figure also shows that less Code Churn (i.e., Code Churn $\leq$ 97) is always associated with low Postrelease bugs.

The interaction between Bugfixes and Developers is consistent with Europa and an OR lower than one. Figure 3.34b shows that files with a high number of Developers significantly decrease Postrelease bugs when files are exposed to Bugfixes compared to the same level of Developers that are not exposed to Bugfixes.



(a) Bugfixess and Code Churn in Ant 16 (b) Bugfixes and Developers in Ant 18

Figure 3.34: Significant interactions in Ant releases



(a) Ant16 (b) Ant18

Figure 3.35: The final results for Ant releases

3.4.3 Xalan Project

Xalan is an Extensible Stylesheet Language Transformations XSLT processor for transforming XML documents into HTML, text, or other XML document types [158]. Four releases were extracted from this project: Xalan 24, 25, 26, and 27 [159]. The numbers of files in the releases were 750, 775, 880, and 920, respectively.

Inclusion and Exclusion

To exclude any release, we checked the number of faulty files in every release. A release was excluded if the release did not have enough faulty files to create a sample. The number of faulty files (case samples) to the whole number of files for every release were shown in Figure 3.36. We had 128, 105, 77, and 144, respectively.

We analyzed the exposure distribution of the two samples in every release. Xalan25 had fewer files exposed to the bug-fixing process (i.e., most Bugfixes are zeros). Further, all faulty files of Xalan27 were exposed to the bug-fixing process (i.e., all Bugfixes are 1). Therefore, we decided to exclude Xalan25 and Xalan27, and included Xalan24 and Xalan26.

Xalan 24

Table 3.48 presents the descriptive statistics of all confounders of Xalan 24, which helped the initial selection of confounders and whether some confounders need to be excluded. For example, Age in some releases of Derby was excluded because those samples contained only new files or only old files. As shown in Table 3.48, there were 128 files from in case group and 128 files from in control group (i.e., we used one-to-one matching). The distribution of the lines of code in the case and the control groups was depicted in Figure 3.38. The Wilcoxon test determined whether the confounders in the two samples were matched well. We did matching using the LOC, which was well matched (p-value > 0.05), along with NPM, AMC, and Age. In terms of Age, the median of the case and control groups were 64 weeks, which means more than 50% of the files from both samples were considered old. Higher complex files were shown in the case group than in the control group. However, the mean and the

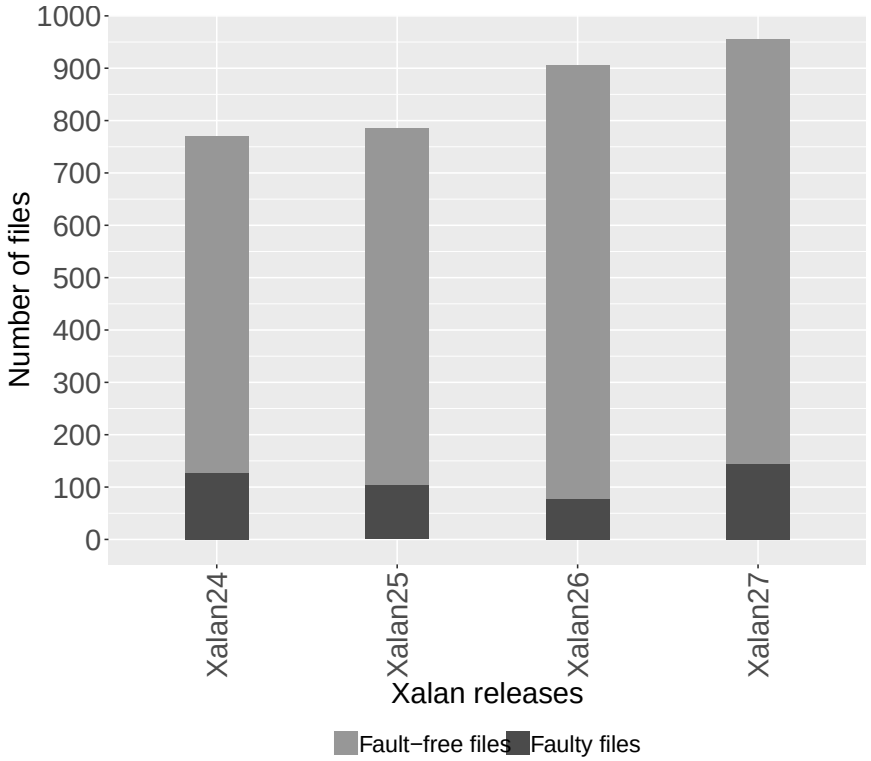


Figure 3.36: The number of faulty files and fault-free files in every release of Xalan

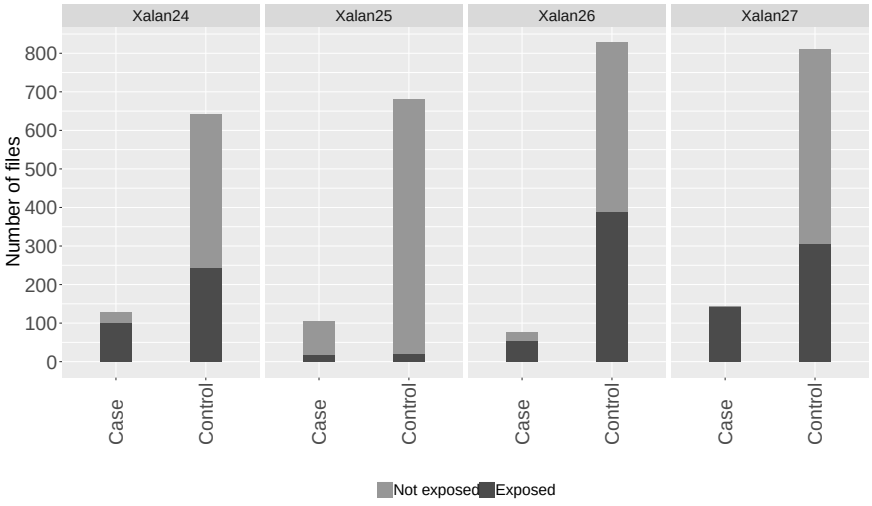


Figure 3.37: The number of faulty files and fault-free files in the case and control groups in every release of Xalan

median of AMC were close to each other in the two samples. The NPMs were almost equal in the two samples. The Bugfixes mean was slightly higher in the case group than in the control group. The Developers median in the case group was higher than the control group, which means more Developers were involved in faulty files than fault-free files.

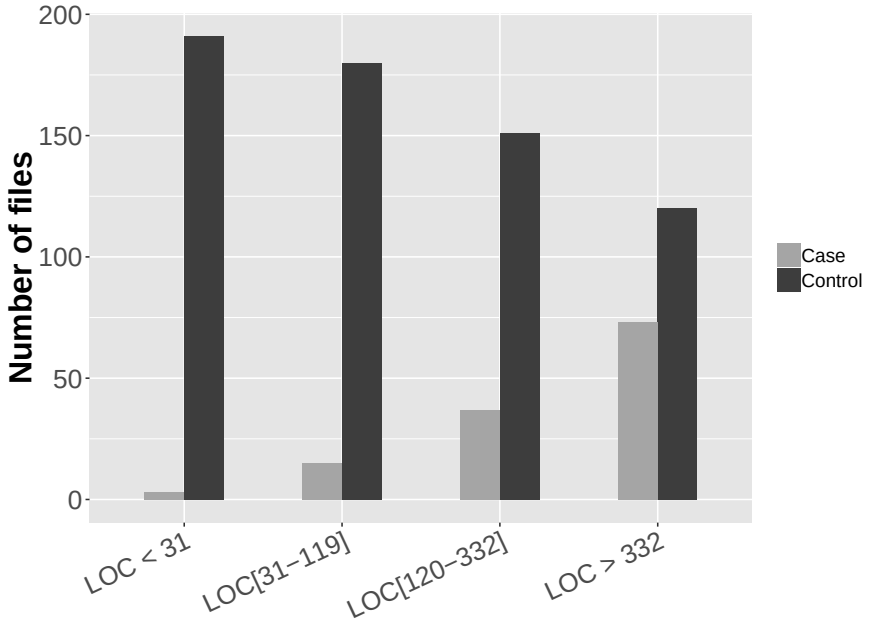
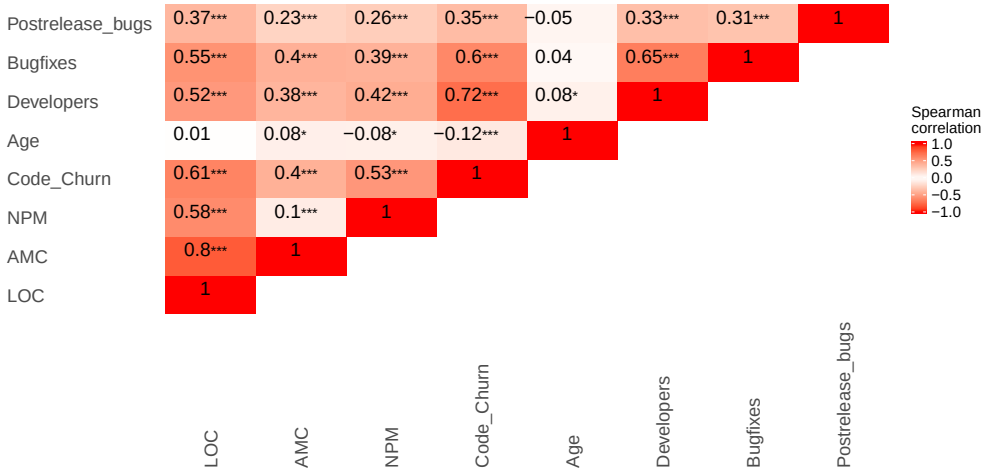


Figure 3.38: The distribution of the Lines of code confounder in the case and control groups in Xalan24

Table 3.48: Descriptive data of the case and control groups in Xalan24

	Case (128 files)					Control (128 files)					Wilcox Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-value
LOC	3	3,472	648.9	383	783.9	7	3,198	589	360	644.62	0.54
NPM	0	115	16.24	9	18.52	0	96	14.24	9	16.34	0.36
AMC	0	436	39.74	23.55	54.86	0	203.33	42.14	28.6	40.19	0.22
Bugfixes	0	1	0.79	1	0.4	0	1	0.62	1	0.48	< 0.01
Developers	1	6	2.93	3	1.36	0	5	2.34	2	1.3	< 0.01
Age	3.14	73.14	58.87	64	18.91	7.85	115.14	62.56	64	16.54	0.46

The correlation coefficient results are shown in Table 3.39. A medium correlation was detected between the exposure and Code Churn ($\beta_i = 0.60$), between the exposure and Developers ($\beta_i = 0.65$), and between NPM and Code Churn ($\beta_i = 0.53$). There was a high correlation between the LOC and AMC ($\beta_i = 0.80$), but this should not affect the model because the LOC was used for matching. Another high correlation is detected between Code Churn and Developers ($\beta_i = 0.72$), which means one of them should be dropped from the initial model. Both show the same level of correlation with the Postrelease bugs. When we implemented the individual regression analysis, Developers confounder was more significant than the Code Churn. Hence, we excluded Code Churn from the initial model, Xalan24$_0$.



*** p-value $<$ 0.001 , ** p-value $<$ 0.01 , * p-value $<$ 0.05 , + p-value $<$ 0.1

Figure 3.39: A pair-wise correlation test on Xalan24 using the Spearman correlation

The initial model Xalan24$_0$, as shown in Equation 3.45, consisted of all confounders except the Code Churn. The model also included interactions between exposures and independent confounders and confounders with themselves.

The first test for the initial model was the multicollinearity test using CNIs and VDPs as presented in Table 3.49. The results indicated that no multicollinearity was detected because the highest CNI is 15.24. Thus, no change was needed for the initial model Xalan24$_0$, and now the model is called the full model, Xalan24$_f$.

$Xalan24_0$

$$\begin{aligned} Y \;=\; & \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC + \beta_4 \cdot Developers + \beta_5 \cdot Age \\ & + \beta_6 \cdot Bugfixes \times NPM + \beta_7 \cdot Bugfixes \times AMC + \beta_8 \cdot Bugfixes \times Developers \\ & + \beta_9 \cdot Bugfixes \times Age + \beta_{10} \cdot NPM \times AMC + \beta_{11} \cdot NPM \times Developers \\ & + \beta_{12} \cdot NPM \times Age + \beta_{12} \cdot AMC \times Developers + \beta_{13} \cdot AMC \times Age \\ & + \beta_{14} \cdot AMC \times Developers + \beta_{15} \cdot Developers \times Age \end{aligned} \tag{3.45}$$

Table 3.49: The CNI/VDP collinearity diagnose for model $Xalan24_0$

CNI ▷▷		7.09	6.32	4.59	3.96	3.28	2.74	2.64	2.18	2.11	1.83	1.73	1.62	1.18	1.09	1.00	
Bugfixes	β_1	0.04	0.05	0.30	0.27	0.17	0.01	0.00	0.03	0.03	0.00	0.04	0.00	0.05	0.00	0.00	
NPM	β_2	0.63	0.11	0.01	0.00	0.00	0.01	0.00	0.17	0.03	0.01	0.01	0.00	0.01	0.00	0.00	
AMC	β_3	0.83	0.06	0.01	0.00	0.01	0.00	0.00	0.01	0.05	0.01	0.00	0.02	0.00	0.01	0.00	
Developrs	β_4	0.01	0.55	0.12	0.00	0.06	0.09	0.04	0.01	0.04	0.03	0.01	0.03	0.04	0.00	0.00	
Age	β_5	0.04	0.76	0.02	0.01	0.01	0.04	0.05	0.00	0.01	0.00	0.00	0.04	0.00	0.01	0.00	
$Bugfixes \times NPM$	β_6	0.01	0.00	0.02	0.26	0.05	0.19	0.35	0.06	0.00	0.00	0.01	0.00	0.00	0.00	0.01	
$Bugfixes \times AMC$	β_7	0.02	0.06	0.34	0.30	0.08	0.04	0.00	0.00	0.00	0.01	0.12	0.01	0.00	0.00	0.00	△ VDP △
$Bugfixes \times Developers$	β_8	0.02	0.00	0.70	0.09	0.01	0.09	0.00	0.01	0.02	0.00	0.00	0.03	0.02	0.01	0.01	
$Bugfixes \times Age$	β_9	0.06	0.17	0.02	0.00	0.21	0.33	0.02	0.03	0.07	0.06	0.00	0.01	0.00	0.01	0.02	
$NPM \times AMC$	β_{10}	0.82	0.10	0.00	0.00	0.00	0.01	0.01	0.03	0.00	0.01	0.00	0.02	0.00	0.01	0.00	
$NPM \times Developers$	β_{11}	0.03	0.03	0.01	0.41	0.36	0.03	0.01	0.07	0.00	0.00	0.01	0.00	0.00	0.00	0.01	
$NPM \times Age$	β_{12}	0.05	0.00	0.03	0.10	0.34	0.00	0.35	0.05	0.00	0.00	0.01	0.02	0.00	0.01	0.02	
$AMC \times Developers$	β_{13}	0.01	0.05	0.35	0.30	0.05	0.04	0.01	0.02	0.09	0.02	0.03	0.01	0.00	0.01	0.00	
$AMC \times Age$	β_{14}	0.00	0.22	0.28	0.06	0.02	0.04	0.01	0.01	0.08	0.17	0.04	0.02	0.01	0.02	0.00	
$Developers \times Age$	β_{15}	0.17	0.70	0.00	0.06	0.01	0.01	0.03	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	

The odds ratios of the confounders and interactions of the full model Xalan24$_f$ are shown under the first column of Table 3.50. The first interaction eliminated was Bugfixes with NPM as shown in the results of Xalan24$_2$ in the second column. The removal did not significantly affect the model as shown by the value of ($\Delta\chi^2 = 0.23$). The second interaction removed from the second model was NPM and Age ($\Delta\chi^2 = 0.37$). The process went on until we reached model Xalan24$_6$, where a total of five interactions had already been removed. We experienced significant change in the model if we try to remove any of the retained interactions. Therefore, we stopped at this point and moved on to eliminate any of the main confounders.

We found that the exposure and two more confounders were not significant. However, the exposure cannot be eliminated and should be retained in the model. The other confounders had significant interactions; therefore, neither can be eliminated. Our final model is the model achieved through the previous process (Xalan24$_6$) and it should be treated as the final model (Xalan24$_{Final}$) for this release; it is presented in Equation 3.46.

Table 3.50: The interaction elimination process for Xalan 24

Variables	Xalan24$_{full}$	Xalan24$_2$	Xalan24$_3$	Xalan24$_4$	Xalan24$_5$	Xalan24$_6$
Bugfixes	0.80	0.82	0.83	0.84	0.90	1.21
NPM	0.45	0.43	0.45	0.68	0.70	0.73
AMC	0.39	0.39	0.42	0.72	0.67	0.67
Developers	4.20***	4.26***	4.18***	4.08***	3.81***	3.14***
Age	0.20***	0.20***	0.21***	0.21***	0.25***	0.25***
Bugfixes × NPM	0.84					
Bugfixes × AMC	0.33**	0.34**	0.34**	0.33**	0.41**	0.39**
Bugfixes × Developers	0.58	0.56+	0.58	0.59	0.59	
Bugfixes × Age	2.22***	2.25***	2.19***	2.20***	2.16***	1.95**
NPM × AMC	0.48	0.49	0.53			
NPM × Developers	1.85*	1.79*	1.59*	1.55*	1.54*	1.42+
NPM × Age	0.78	0.75				
AMC × Developers	4.01*	4.02**	3.97**	3.78**	2.70**	2.69**
AMC × Age	0.47	0.46	0.48	0.53		
Developers × Age	0.33**	0.33**	0.31**	0.30**	0.29**	0.28**
χ^2	59.5***	59.26***	58.47***	57.74***	56.18***	53.41***
Likelihood ratio test = $\Delta\chi^2$		0.23	0.37	0.72	1.55	2.77
R^2	0.30	0.30	0.30	0.30	0.30	0.29
Deviance explained	89.41%	89.52%	89.00 %	89.28%	89.41 %	86.17%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$						

$$Xalan24_{Final} = 0.19 \cdot Bugfixes - 0.30 \cdot NPM - 0.39 \cdot AMC + 0.19 \cdot Developers$$
$$- 1.37 \cdot Age + -0.92 \cdot Bugfixes \times AMC + 0.66 \cdot Bugfixes \times Age + 0.35 \cdot NPM \times Developers$$
$$+ 0.99 \cdot AMC \times Developers - 1.24 \cdot Developers \times Age \quad (3.46)$$

Xalan 26

We started with the descriptive data of the selected confounders, as shown in Table 3.51. The number of files in every sample was 77 based in the distribution of the LOC, which is shown in Figure 3.40. The median of the exposure was one in both samples, which indicated that more than 50% of every sample had been exposed to bug-fixing process. Additionally, more than 50% of the files between the two samples were only exposed to one developer.

In terms of the number of public methods and complexity, both samples had a similar level because they both matched based on the results of the Wilcoxon test for the two samples. Similarly, the two samples are well matched in terms of the lines of code (p-value = 0.83). In terms of Age, all the files in the case sample were new (less than 53 weeks). Most of the files in the control sample were new and the median = 9. For that reason, Age was excluded from the initial model. The Bugfixes existed in the two samples with no significant difference. Developers in the two samples were close with a similar mean.

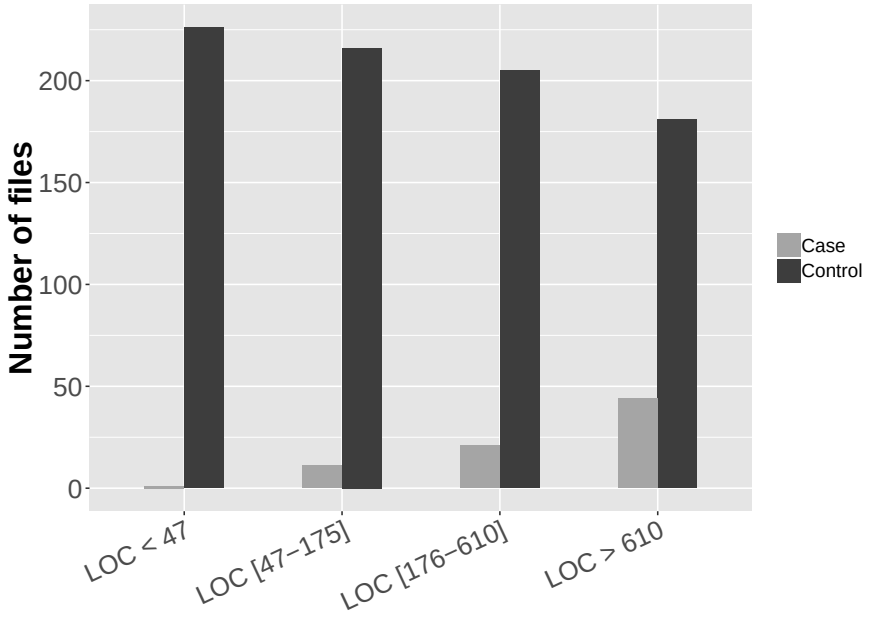
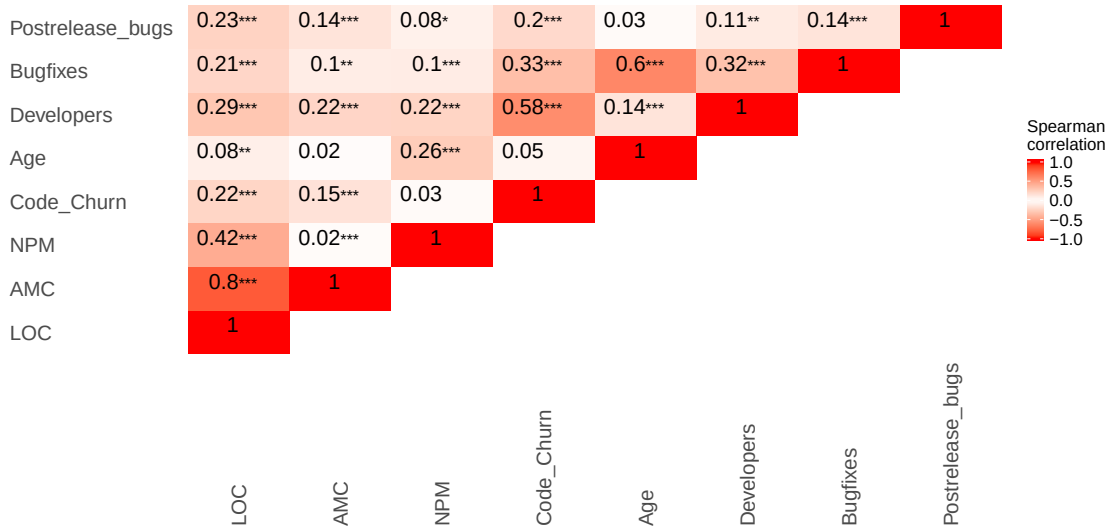


Figure 3.40: The distribution of Lines of code in the case and control groups in Xalan26

The correlation test results of Xalan26 were shown in Figure 3.41. A high correlation was detected between AMC and LOC, which was fine because LOC was the matching confounder. Medium correlations were detected between Bugfixes and Age and between Developers and Code Churn.

Table 3.51: Descriptive data of case and control groups of Xalan-26

	Case (77 files)					Control (77 files)					Wilcox Test
	Min	Max	Mean	Median	SD	Min	Max	Mean	Median	SD	P-value
LOC	10	6,114	990.4	628	1,198.55	3	6,478	919.8	628	1,100.83	0.83
NPM	0	166	18.66	9	26.81	0	110	16.21	8	19.92	0.54
AMC	3.63	815.6	51.48	38	99.42	0	1251	105.66	36.89	203.71	0.33
Bugfixes	0	1	0.72	1	0.44	0	1	0.58	1	0.49	0.06
Developers	1	5	1.48	1	0.86	0	4	1.46	1	0.81	0.91
Age	0.85	21	11.73	18.57	8.74	0.85	178	11.93	9.71	21.09	0.62
Code churn	64	634	214.5	82	226.31	0	478	92	67	64.75	< 0.001



*** p-value < 0.001 , ** p-value < 0.01 , * p-value < 0.05 , + p-value < 0.1

Figure 3.41: A pair-wise correlation test on Xalan26 using the Spearman correlation.

The first, initial model, Xalan26_{01}, included two static code confounders, the exposure, and Developers and their interactions as shown in Equation 3.47. This model was tested for multicollinearity using CNI/VDP. The highest CNI was 11.50, which indicated an absence of multicollinearity. Therefore, this model, and that is where we start the interaction elimination process.

$Xalan26_{01}$

$$\begin{aligned} Y \ &= \beta_1 \cdot Bugfixes + \beta_2 \cdot NPM + \beta_3 \cdot AMC + \beta_4 \cdot Developers \\ &+ \beta_5 \cdot Bugfixes \times NPM + \beta_6 \cdot Bugfixes \times AMC + \beta_7 \cdot Bugfixes \times Developers \\ &+ \beta_8 \cdot NPM \times AMC + \beta_9 \cdot NPM \times Developers + \beta_{10} \cdot AMC \times Developers \end{aligned} \tag{3.47}$$

Table 3.52: The CNI/VDP collinearity diagnosed for model Xalan26_0

CNI ▷▷		11.50	4.62	3.08	2.42	1.84	1.75	1.71	1.54	1.39	1.00	
Bugfixes	β_1	0.01	0.14	0.33	0.01	0.00	0.02	0.22	0.25	0.00	0.00	△ VDP △
NPM	β_2	0.88	0.00	0.00	0.00	0.03	0.01	0.03	0.01	0.03	0.00	
AMC	β_3	0.94	0.03	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	
Developers	β_4	0.00	0.13	0.57	0.04	0.04	0.11	0.00	0.03	0.01	0.00	
$Bugfixes \times NPM$	β_6	0.06	0.09	0.01	0.44	0.25	0.00	0.00	0.00	0.12	0.01	
$Bugfixes \times AMC$	β_7	0.01	0.90	0.03	0.01	0.01	0.00	0.00	0.01	0.00	0.02	
$Bugfixes \times Developers$	β_8	0.00	0.07	0.69	0.02	0.03	0.04	0.00	0.07	0.02	0.01	
$NPM \times AMC$	β_{10}	0.98	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	
$NPM \times Developers$	β_{11}	0.00	0.07	0.00	0.75	0.05	0.01	0.00	0.00	0.03	0.01	
$AMC \times Developers$	β_{13}	0.00	0.72	0.01	0.00	0.11	0.04	0.07	0.02	0.02	0.02	

The backward elimination process to eliminate insignificant interactions are shown in Table 3.53. We first removed the interaction between Bugfixes and AMC interaction, with no significant change ($\Delta\chi^2 = 0$). Second, AMC and Developers interaction were removed, with no significant impact. Third, the bugfix and NPM interaction were removed, with no significant impact. The NPM and developer interaction was the last to be removed. There were no confounders to eliminate at this point, and model Xalan26_5 is the final model (see Equation 3.48).

$Xalan26_{final}$

$$\begin{aligned} Y \ &= \ 0.04 \cdot Bugfixes - 6.7 \cdot NPM - 13.28 \cdot AMC + 0.09 \cdot Developers \\ &- 0.45 \cdot Bugfixes \times Developers - 13.96 \cdot NPM \times AMC \end{aligned} \tag{3.48}$$

Table 3.53: Xalan26 release model reduction using backward hierarchal elimination for the interactions

Variables	Xalan26$_{full}$	Xalan26$_2$	Xalan26$_3$	Xalan26$_4$	Xalan26$_5$
Bugfixes	0.99	0.99	1.03	1.04	1.04
NPM	0.00**	0.00**	0.00**	0.00**	0.00**
AMC	0.00**	0.00***	0.00***	0.00***	0.00***
Developers	1.03	1.03	1.02	1.10	1.09
Bugfixes × NPM	0.71	0.71	0.71		
Bugfixes × AMC	0.99				
Bugfixes × Developers	0.56+	0.56+	0.58+	0.63+	0.63+
NPM × AMC	0.00**	0.00**	0.00**	0.00**	0.00**
NPM × Developers	1.58+	1.58+	1.45	1.45	
AMC × Developers	1.33	1.33			
χ^2	37.86***	37.86***	35.88***	33.67***	32.89***
Likelihood ratio test = $\Delta\chi^2$		0.00	1.98	0.69	0.78
R^2	0.32	0.32	0.32	0.29	0.29
Deviance explained	62.76%	62.76%	61.98%	62.20%	61.42%
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$					

Table 3.54: Results of the goodness of fit test for models $Xalan24_{final}$ and $Xalan26_{final}$ according to Hosmer Lemoshow

Model	HL result χ^2	df	P-Value	H_0	Good Fit
Xalan24$_{final}$	5.85	8	0.66	Do not reject	Yes
Xalan26$_{final}$	3.87	8	0.86	Do not reject	Yes

H_0: The distribution of the expected values ($\hat{Y}_i$) is not significant different than the distribution of the real values (Y_i).

Goodness of Fit Test of Xalan Models and Discussion of the Results

The goodness of fit test results for the final models, Xalan24$_{final}$ and Xalan26$_{final}$, are presented in Table 3.54. The results of the χ^2 and the P-value, in the table, indicated the good fit of the two models Xalan24$_{final}$ and Xalan26$_{final}$. The null hypothesis was not rejected because there are no statistical differences between the actual observations (Y_i) and the predicted values ($\hat{Y}_i$).

The main observation for the main confounders of Xalan is as follows:

- The chance of Postrelease bugs increases with more Developers and old files by 314% and 75%, respectively.

The main observations for the interactions of confounders of Xalan are as follows:

- The interaction between Bugfixes and the average method complexity AMC decreases the chance for Postrelease bugs by 75%.
- The interaction between Bugfixes and the Age of the file increases the chance for Postrelease bugs by 95%.
- The interaction between the number of Developers and the average method complexity AMC increases the chance for Postrelease bugs by 269%.
- The interaction between the number of Developers and the Age of the file decreases the chance for Postrelease bugs by 72%.

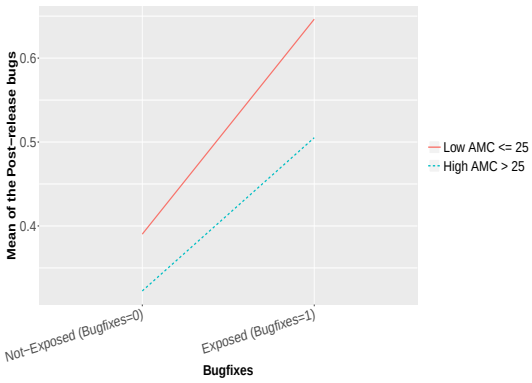
Figure 3.43 illustrates the final odds ratios and confidence intervals. The two models showed insignificant results of the exposure (i.e., Bugfixes). Age was only used in Xalan24, and it showed a significant result. For Age, we used binary values; 1 for new files (less than 53 weeks) and 0 for old files (53 weeks or older). We found that old files contained more faults. The odds ratio of Age was 0.25, which indicated old files had 75% less chance of Postrelease faults. The chance of Postrelease bugs increased by three times if more Developers were working on a single file.

In the Xalan24 release, the interaction between Bugfixes and AMC was significant and below one. Figure 3.42a showed that highly complex files (i.e., AMC $>$ 25) that had not been exposed to Bugfixes were less likely to experience Postrelease faults. The chance of Postrelease bugs was increased when less complex files were exposed to Bugfixes.

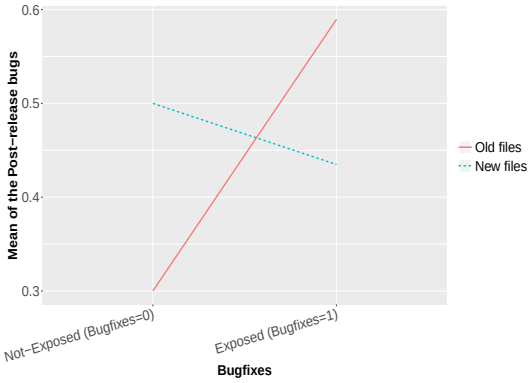
The exposure (i.e., Bugfixes) and Age significantly interacted with an odds ratio above one (OR=1.95). Figure 3.42b showed that old files have lower chance to contain Postrelease faults when these files were not exposed to Bugfixes. However, chances for new files to contain Postrelease bugs decreased when they were exposed to Bugfixes.

NPM and Developers interaction led to a decrease in Postrelease bugs as shown in Figure 3.42c. With a high number of Developers (i.e., > 2), the chance for Postrelease bugs on files decreased when they interacted with files with a high number of methods (i.e., > 9). On the other hand, files with a low number of Developers and lower number of methods experienced a very low chance of Postrelease bugs. AMC (low complex files) and Developers (low or high number) both had a high chance of Postrelease bugs (see Figure 3.42d). However, to reduce the chance of Postrelease bugs, fewer Developers are needed.

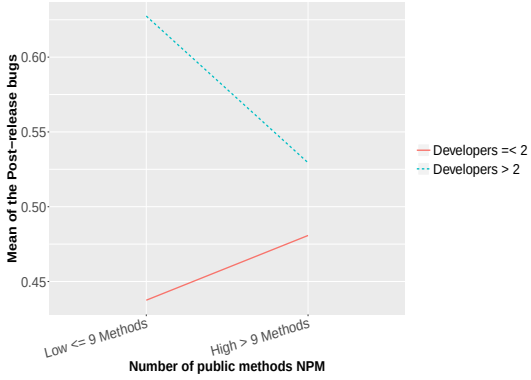
In Xalan 26, NPM and AMC are significant as confounders with an odds ratios lower than one (OR = 0.01). This indicated that files with lower complexity and lower methods considerably reduce Postrelease bugs.



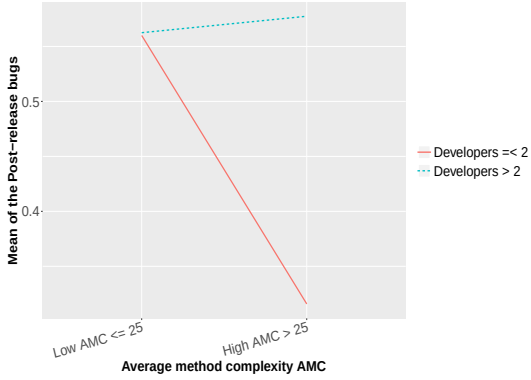
(a) Bugfixess and AMC



(b) Bugfixess and Age

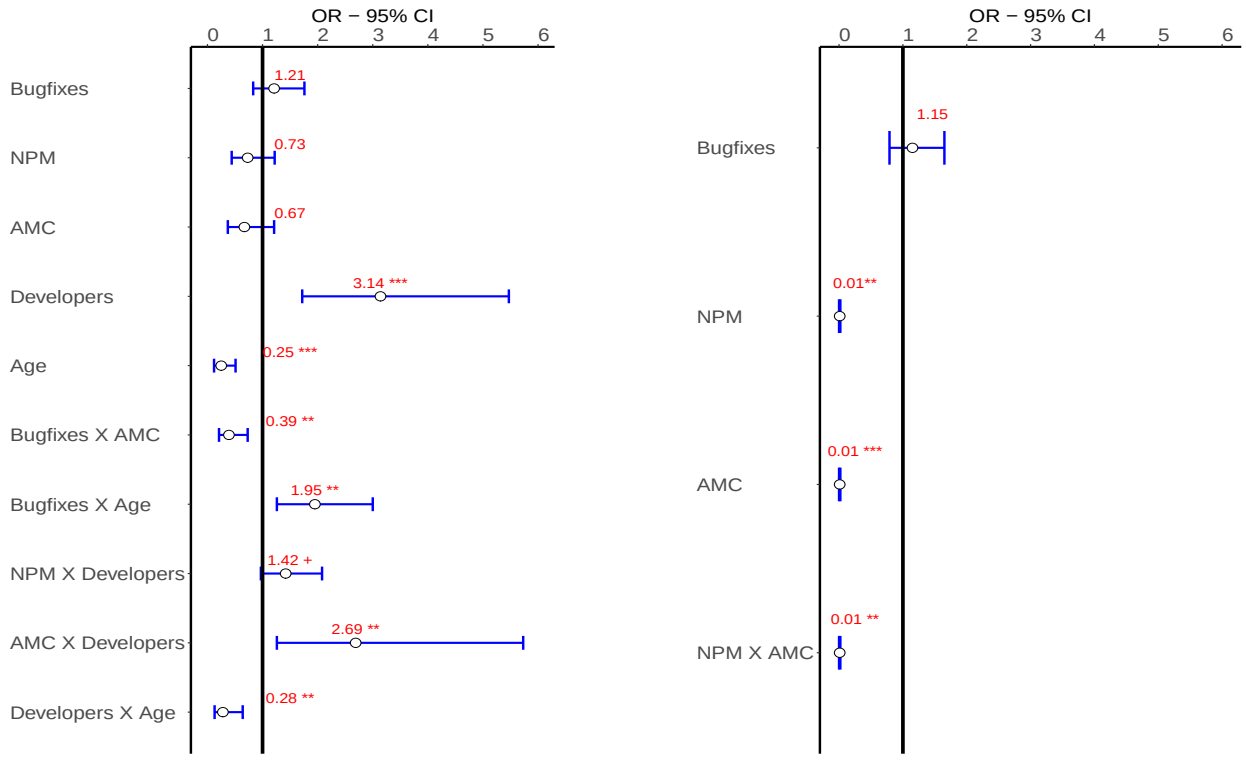


(c) NPM and Developers



(d) AMC and Developers

Figure 3.42: Interactions from the Xalan24 release



(a) Xalan24 final results

(b) Xalan26 final results

Figure 3.43: The final results for the Xalan releases

3.5 Discussion of the Results Across All Case Studies: Europa, Ganymede, Ant, Derby, and Xalan

In this section, results of all projects (i.e., the first work in Section 3.3 and the replicated study) are summarized and compared. Answers to the research questions, RQ1 and RQ2 at the beginning of Section 3.4, are answered in Tables 3.55 and 3.56.

We start with the exposure (Bugfixes), which shows consistent results across most of the projects. The first research question deals with the relation between the exposure and Postrelease bugs. The OR of the exposure is significant and above one. This means that more prerelease faults are seen in faulty files than in fault-free files. The OR had different values across all projects. Some projects showed high odds ratios, such as Derby 10.5.1.1 (OR = 6.19) and Derby 10.8.1.2 (OR = 5.59). Other projects showed lower odds ratios, such

as Derby 10.8.3.0 (OR = 3.66), Eclipse's Europa release (OR= 2.02) and Eclipse's Ganymede release (OR = 2.97). Other projects had OR higher than one but less than two, such as Derby 10.1.3.1 (OR = 1.7), 10.4.1.3 (OR = 1.7), and Derby 10.6.1.0 (OR = 1.96). Bugfixes OR was not significant in the Xalan project and it was below one in the Ant 16 project.

The second consistent result was related to the number of Developers contributing to a single file. The Developers confounder had an OR values higher than one in Europa, Ganymede, Derby 10.1.3.1, Derby 10.4.1.3, Derby 10.6.1.0, Ant16, Ant18, and Xalan24. High ORs were recorded with Ganymede (OR = 7.33), Derby 10.1.3.1 (OR = 14.86), Derby 10.4.1.3 (OR = 7.56), and Derby 10.6.1.0 (OR = 8.36). Lower ORs were recorded with Europa (OR = 2.05), Ant16 (OR = 1.61), Ant18 (OR = 3.16), and Xalan (OR = 3.14). Only in Xalan26 was Developers not found to be significant.

Age in our work was dichotomous (0 for old files and 1 for new files). Age was excluded from all Derby, Ant and Xalan26 release because all of the case samples in these releases were new files. For specific releases here the new files were likely to experience Postrelease faults than old files, which is consistent with the results found in Europa (OR = 6.57). Xalan24 had the opposite result, with OR = 0.25, which means new files have a lower chance of Postrelease faults than old files.

Code churn was found to cause Postrelease bugs in Ganymede (OR = 4.60) and Ant16 (OR = 10.20). Code churn was not used because it was highly correlated with Developers in all Derby releases. Developers was chosen to be included in the models over the Code Churn because it showed higher impact when regressed with Postrelease bugs.

The static code confounders, AMC or Average Complexity and NPM or Method Calls, were not significant across several releases (i.e., Europa, Derby 10.1.3.1, Derby 10.6.1.0, Ant 16, and Ant 18). However, some significant results were recorded in Ganymede, Derby 10.4.1.3, Derby 10.5.1.1, Derby 10.8.1.2, Derby 10.8.3.0, and Xalan 26. More complexity caused higher chance of Postrelease bugs in Ganymede (OR = 1.2), but lower chance of Postrelease bugs in Derby 10.5.1.1 (OR = 0.25), Derby 10.8.1.2 (OR = 0.11), and Xalan26 (OR = 0.01). It is important to mention that the complexity confounder used in the Eclipse datasets and Apache datasets are different. In this sense, the complexity effect on Postrelease bugs was more consistent in Apache projects. In Eclipse, there is not sufficient information

to draw a conclusion. Method Calls OR was not significant and was eliminated from the Europa Model. In Ganymede, Method Calls was significant but had a neutral odds ratio (OR = 1) which indicates no impact to the Postrelease bugs. The OR of NPM in Derby were not consistent; the OR was above one in Derby 10.4.1.3 (OR = 1.46) and below one in Derby 10.8.3.0 (OR = 0.66).

Table 3.55: Answers to the research questions RQ1 across all projects

RQ1: What are the main confounders that cause an increase or decrease to the Postrelease bugs?	
What are the impact of the exposure (Bugfixes) on the Postrelease bugs?	
Result	**Releases**
Increase	Eclipse Europa (OR = 2.02), Eclipse Ganymede (OR = 2.97), Derby 10.1.3.1 (OR = 1.70), Derby 10.4.1.3 (OR = 1.70), and Derby 10.6.1.0 (OR = 1.96), Derby 10.8.1.2 (OR = 5.59) and Derby 10.8.3.0 (OR = 3.66)
Decrease	Ant16 (OR = 0.43)
What is the impact of the complexity/AMC on the Postrelease bugs?	
Result	**Releases**
Increase	Eclipse Ganymede (OR = 1.20)
Decrease	Derby 10 5.1.1 (OR = 0.25), Derby 10 8.1.2 (OR = 0.11), Derby 10.8.3.0 (OR = 0.01), and Xalan26 (OR = 0.01)
What is the impact of the number of methods/NPM on the Post-release bugs?	
Result	**Releases**
Increase	Derby 10.4.1.3 (OR = 1.46)
Decrease	Derby 10.8.3.0 (OR = 0.66)
Neutral	Eclipse Ganymede (OR = 0.92)
What is the impact of the number of Developers working in a single file on the Postrelease bugs?	
Result	**Releases**
Increase	Eclipse Europa (OR = 2.05) and Ganymede (OR = 7.33), Derby 10.1.3.1 (OR = 14.88), Derby 10.4.1.3 (OR = 7.56), Derby 10.6.1.0 (OR = 8.36), Ant16 (OR = 1.61), An18 (OR = 3.16), and Xalan24 (OR = 3.11)
What is the impact of the Age of file (old or new) on the Post-release bugs?	
Result	**Releases**
Increase	New files of Europa Eclipse (OR = 6.57)
Decrease	New files of Xalan24 (OR = 0.25)
What is the impact of the Code Churn (code added and deleted) on the Postrelease bugs?	
Result	**Releases**
Increase	Eclipse Ganymede (OR = 4.66), and Ant16 (OR = 10.20)

We analyzed the findings of the interactions across all projects, starting with the interactions with the exposure (i.e., Bugfixes). The highest consistent interaction was found between Bugfixes (i.e., prerelease bugs) and Developers, and it was found in four projects and seven releases: Europa (OR = 0.45) , Ganymede (OR = 0.17), Derby 10.1.3.1 (OR =

0.32), Derby 10.4.1.3 (OR = 0.26), Derby 10.6.1.0 (OR = 0.32), Ant18 (OR = 0.57), and Xalan26 (OR = 0.63). The interaction was either not included because of the exclusion of Developers (in Derby 10.5.1.1, Derby 10.8.1.2 Derby 10.8.3.0) or removed during the model building process (in Xalan 26). We already know that more bug fixing and more Developers individually caused a higher chance of Postrelease faults. However, more Developers involved in the bug-fixing process caused fewer Postrelease bugs than when fewer Developers were involved.

The interaction between Bugfixes with AMC was significant in two releases of two projects. The two results were not consistent because in one release (Derby 10.8.1.2) caused an increase in Postrelease bugs and the other (Xalan24) caused a decrease. Hence, there is no sufficient evidence to provide a conclusion on this interaction. Similarly, the interaction between Bugfixes and NPM has no sufficient evidence because it appears one time, in Derby 10.1.3.1.

The interaction of Bugfixes and Age was not consistent because it appeared in three releases, showing distinct results, as shown in Table 3.55. Likewise, Bugfixes and Code Churn was presented in one model, in Ant16. It is important to recall that Age and Code Churn were excluded from many releases in this study.

The interaction between AMC and NPM appeared in two releases: Derby 10.5.1.1 (OR = 0.13) and Xalan 26 (OR = 0.01). As explained earlier, a high NPM with low AMC has the highest Postrelease bugs in Derby 10.5.1.1. In Xalan 26, highly complex files with high NPM is still the highest but at the same level as the less complex files with high NPM. The interaction between AMC and Developers appeared in one release, Xalan 24 (OR = 2.69). This indicates that more Developers working on complex files increases the risk of Postrelease bugs.

The interaction between NPM and Developers also showed in two releases with different results: high odds ratio with Xalan 24 (OR = 2.69) and low odds ratio with Derby 10.1.3.1 (OR = 0.66).

The Developers and Age interactions were significant in Europa (OR = 1.54), Ganymede (OR = 2.64), and Xalan24 (OR = 0.26). It is more consistent with Eclipse projects, but it showed an opposite result compared to Xalan24. It is important that in most of the Apache projects, Age was excluded because of the imbalance in the distribution of new and old files.

Table 3.56: Answers to research question RQ2 across all projects

RQ2: What are the main interactions that cause an increase or decrease to the Postrelease bugs?	
What is the impact of the interaction between Bugfixes and AMC/Average complexity on the Postrelease bugs	
Result	**Releases**
Increase	Derby 10.8.1.2 (OR = 4.75)
Decrease	Xalan24 (OR = 0.39)
What is the impact of the interaction between Bugfixes and NPM/Method Calls on the Postrelease bugs?	
Result	**Releases**
Increase	Derby 10.1.3.1 (OR = 1.47)
What is the impact of the interaction between Bugfixes and Developers on the Postrelease bugs?	
Result	**Releases**
Decrease	Eclipse Europa (OR = 0.45) and Ganymede (OR = 0.17), Derby 10.1.3.1 (OR = 0.33), Derby 10.4.1.3 (OR = 0.28), Derby 10.6.1.0 (OR = 0.32), and Ant18 (OR = 0.57)
What is the impact of the interaction between Bugfixes and Age on the Postrelease bugs?	
Result	**Releases**
Increase	Xalan24 (OR = 1.95)
Decrease	Eclipse Europa (OR = 0.31)
Neutral	Eclipse Ganymede (OR = 1.00)
What is the impact of the interaction between Bugfixess and Code Churn on the Postrelease bugs?	
Result	**Releases**
Increase	Ant16 (OR = 0.21)
What is the impact of the interaction between AMC and NPM on the Postrelease bugs?	
Result	**Releases**
Decrease	Derby 10.5.1.1 (OR = 0.13), Xalan 26 (OR = 0.01)
What is the impact of the interaction between AMC and Developers on the Postrelease bugs?	
Result	**Releases**
Increase	Xalan 24 (OR = 2.69)
What is the impact of the interaction between AMC/Complexity and Age on the Postrelease bugs?	
Result	**Releases**
Decrease	Eclipse Ganymede (OR = 0.81)
What is the impact of the interaction between NPM and Developers on the Postrelease bugs?	
Result	**Releases**
Increase	Xalan 24 (OR = 1.42)
Decrease	Derby 10.1.3.1 (OR = 0.68)
What is the impact of the interaction between Developers and Age on the Postrelease bugs?	
Result	**Releases**
Increase	Eclipse Europa (OR = 1.45), and Ganymede (OR = 2.64)
Decrease	Xalan 24 (OR = 0.28)

3.6 Threats to Validity for Software Faults Proneness

Construct validity can be violated if we do not test what we intend to test. The confounders were defined clearly to avoid any ambiguity. The study used a combination of static code and change confounders (i.e.,confounders) to avoid any mono-operation bias. We used the Spearman correlation test to check for any high collinearity before including the confounders in our model. Any high collinearity was eliminated from the initial models of all case studies. For example, Revision was removed due to the high collinearity with Bugfixes in Europa. To avoid bias in our samples, we matched our data based on the LOCs. In addition, we centered our confounders scale to the mean to avoid multicollinearity between confounders and interactions. Then, we tested the initial model using the multicollinearity test CNI/VDP. Any multicollinearity detected by this test was removed before starting the process of building the final model.

Data quality is an important issue of the **internal validity**. The static code and change code confounders for Eclipse projects were extracted as part of earlier research [2], [38], [159]. The level of confidence concerning the data quality is very high [2]. In addition, we did sanity checks on our confounders. For example, it was noticed that the Average Complexity for some files had zero values. It was found that these were interface files, which usually do not contain methods. Therefore, we excluded those files from our sample.

Using the wrong statistical assumption will obviously lead to threats to **conclusion validity**. We tested the matching between the samples using the Mann-Whitney test to make sure that matching was done correctly. This study used non parametric tests such as Spearman test because of the skewness of the data distribution of variables. We also used conditional logistic regression to estimate the OR of every confounder and 95% CI which is appropriate for matching studies. The elimination was based on hierarchal backward method [145], which was required due to the presence of interactions in the model. We eliminated confounders that were highly correlated with other explanatory confounders. Further, we diagnosed for multicollinearity and eliminated some interactions based on that. This should not affect our explanation of the confounders in our model. These steps should lead to a final model that is not complex and is highly significant.

External validity is concerned with generalization. This study used data from Eclipse Europa and Ganymede releases, and used data from following Apache projects: Derby, Ant, and Xalan. Some confounders showed consistent results across all projects (e.g., Bugfixes, and number of Developers). Also, the effect of the interactions between Bugfixes and Developers was consistent across most projects. We cannot claim the generalizability for other confounders and interactions that were not consistent across multiple projects or we cannot claim the generalizability for projects that were not used in this work.

3.7 Conclusion for Case-control Study

In this chapter, we used a case-control study for the first time in the field of software engineering to study software fault proneness. We presented a detailed methodology of how to build a model by eliminating unnecessary interactions and confounders. The second contribution of this chapter was including into the models the interaction between confounders and how that affected software fault proneness, in addition to the individual confounders. The interactions were not considered in any related work using explanatory studies [3, 4, 5, 6, 7, 8, 9]. The results showed, the interactions had significant effects on software fault proneness.

Further, the replicated study showed consistent results for Bugfixes (i.e., prerelease bugs), Developers and the interaction between them. Some confounders such as Age and Code churn were not included in the Apache projects due to the high correlation. Some static code confounders showed slightly higher effect in the Apache projects like Complexity AMC and Number of Public Methods (NPM) than in the case of Eclipse datasets.

The results identified confounders that can be used to explain Postrelease bugs. Specifically, Bugfixes, Developers, and Age had the highest OR. These were seen more in cases than in controls, which means they contributed more than other confounders to the software fault proneness. The results of Bugfixes and Developers are consistent across all Eclipse and

Apache projects. The results of Age were only consistent in Europa and Ganymede. The highest consistent interaction is between Bugfixes and Developers. Also, the interaction between Complexity and Bugfixes showed consistency among Apache projects. Bugfixes with Age interaction was consistent in Eclipse projects but not with Apache.

The results of this study are informative and explain impact of confounders and interactions on software fault proneness. Our future work will apply the case-control methodology on other software projects to further explore the generalizability. We will also explore using case-control studies on software development effort. Extracting more confounders is another possible direction of our future research.

Chapter 4

Software Fault Proneness Prediction

This chapter focuses on the prediction part of software fault proneness. We employed the algorithm used in the previous chapter (i.e., conditional logistic regression) and the models achieved by the case-control study. We tested the prediction performance of these models using performance metrics explained in Section 4.2. We used the same matched samples created in the previous chapter for the case-control models using conditional logistic regression (CLR) and for five other widely used classifiers in the area of software fault pronenesses: logistic regression (LR), naive Bayes (NB), decision tree (J48), random forest (RF), and decision list (PART). In addition, we applied the group lasso regression (G-Lasso) algorithm, which has not been used previously for software fault proneness prediction.

This chapter starts with introduction and motivation for the prediction work in Section 4.1. Then the approach of this study, including brief explanation of the algorithms used, statistical analysis applied, and performance metrics used are discussed in 4.2. A brief explanation of datasets used and the main features used are discussed in Section 4.3, followed by a discussion of the results in Section 4.4. Then threats of validity for the prediction research were discussed in Section 4.5, and finally the chapter is concluded in Section 4.6.

4.1 Introduction and Motivation

Software faults are problematic when they are not detected and fixed early because they may cause the software to fail to perform its required function. Therefore, predicting fault-prone software units (i.e., files) is essential because it helps fix faults before the product is deployed to end users. The sooner the software faults are detected, the better it is for the software development cost and efforts.

Many studies have focused on predicting software fault prone units, using different types of metrics, data sets, and machine learning algorithms [34, 162]. While explanatory studies address questions like 'what' and 'how', prediction studies address questions like 'where' and 'when'. Prediction studies use confounders to predict what would happen to the software unit in the future, that is, if they will be faulty or not.

Many classifiers (e.g., LR, NB, J48) have been used to predict fault proneness on many software projects (e.g., Eclipse, Apache), applying different types of metrics (e.g., static code, change metrics). Using different classifiers is essential because not all classifiers perform at the same performance level. To measure the performance of the prediction, we measure performance metrics such as recall (i.e., the rate of all true positive over the actual true) and precision (i.e., the rate of all true positive over the predicted true. performance metrics in this context are different from the features, variables, metrics, and confounders, which are synonymous and used interchangeably in this study.

Many approaches have considered improving prediction performance by the selection of the classifiers, selection of the features, improving the distribution of the response variables, or improving the distribution of the independent variables. Classifiers differ in terms of their performance. Therefore, many classifiers have been used in the area of software fault proneness. Some provided fair performance, and others were very robust. Further, different types of features (i.e., metrics) were used to improve prediction performance. Other studies focused on the type of data and worked on improving distribution (e.g., re sampling, cost sensitive, transforming data) [163, 164, 165, 166, 167, 168, 169]. The samples we used in this

chapter were stratified based on the lines of code (LOC), which required selecting fault-free files (from the controls group) with similar sizes as the files from faulty files group (i.e., cases group). This does the same job as other sampling techniques, that were developed to increase the number of minor events and made the two classes to be balanced.

In this chapter, we measure the prediction performance of the explanatory models that were built in Chapter 3. with a goal to find out whether they are useful for prediction as they were explanatory models. We built explanatory models using Eclipse and Apache projects based on a case-control methodology. These models did not use the whole set of confounders because we eliminated confounders based on the results of the correlation test, as discussed in Chapter 3. The process started with the group of confoundera and their interactions, then insignificant interactions and confounders were eliminated based on the backward hierarchal approach. The final model contained only significant interactions and confounders, which was much smaller than in the initial model. For instance, we started with six confounders and 15 interactions with Europa's model and the final model had three metrics and three interactions.

Further, we used the same matched samples on other widely used classifiers to compare their performance with the performance of our models.

The second contribution of this chapter is applying for the first time for software fault proneness prediction an algorithm that accounts for variable shrinkage and selection using lasso (least absolute shrinkage and selection operator). The method consists of eliminating unnecessary metrics from the model by assigning a penalty (i.e., λ) to regularize the model, which results with a sparse model (i.e., a model with fewer metrics). Some metrics are minimized to a very low value (i.e., shrinkage), and other metrics are eliminated (i.e., their coefficients become zeros). In case-control studies, we eliminated variables based on the correlation test, goodness of fit, and significance level in the model. With the lasso regression, this can be automated assigning the penalty, which is estimated through a k-fold cross-validation of the whole dataset. This method is good for the prediction purpose be-

cause it takes less time and effort. However, this method cannot handle the interaction as the conditional logistic regression can. The algorithm we applied is G-Lasso, which is an extension of the linear lasso regression. The G-Lasso is designed to fit the binary format of our response variable (i.e., fault prone and fault-free files).

Specifically, we measured the performance of conditional logistic regression models that were built using a case-control method and accounted for matching and involved interactions. Further, we measured the performance of G-Lasso. Then we compare their performances with six other machine learning algorithms (i.e., LR, NB, J48, PART, and RF) on 12 releases from the Eclipse and Apache projects at the file level. The following performance metrics were used for comparison: area under curve (AUC), recall, precision, false positive rate (FPR), F-score (the harmonic mean of recall and precision), and G-score (the harmonic mean of recall and 1-FPR). We also applied statistical tests to compare differences among all performance metrics of all classifiers. The research questions we address in this chapter can be summarized as follows:

- **RQ1:** Does CLR perform better than other classifiers?
- **RQ2:** How does G-Lasso perform compared to other classifiers?
- **RQ3:** What is the ranking of the classifiers in terms of the performance measures (i.e., recall, precision, FPR, G-score, F-score, and AUC)?
- **RQ4:** Does the dataset affect the prediction performance of the CLR or the prediction performance of G-Lasso?
- **RQ5:** Does the CLR using reduced models with interactions (i.e., achieved by the case-control methodology) perform better than other algorithms used in related studies?

4.2 Approach

We used 17 software change metrics as features for software fault proneness prediction. Using G-Lasso and six other machine learning algorithms, our models predicted which software units (i.e., files) are fault prone.

4.2.1 Machine learning algorithms

In this study, we used G-Lasso and CLR and five widely used supervised machine learning algorithms: LR, NB, PART, J48, and RF.

LR models describe the probability of existence of a condition (i.e., fault prone or fault-free) based on a given set of variables X_i. The set of variables is described based on a linear function and then placed into the logit model to calculate the probability ranged between 0 and 1 as shown in Equation 4.1.

$$Y = \beta_0 + \beta_1 \mathrm{X}_1 + \beta_2 \mathrm{X}_2 + ... + \beta_i \mathrm{X}_i \tag{4.1}$$

where Y is the response variable (fault prone, fault free), and X_i is the independent variable (i.e., metric).

CLR is a special case for the LR with a stratification (i.e., matching) option. CLR can have a single or multiple confounders used in the model for matching only. In the previous chapter, we explained that our matching confounder was the LOC, which means that the function compares files with the same value of LOC to estimate the coefficients for the independent variables (i.e., metrics) of the model. Note that in this model we used interactions, which was not the case for other classifiers. The CLR is presented by Equation 4.2, which is similar to the logistic model with a stratum (LOC for our our case).

$$Y = \beta_0 + \beta_1 \mathrm{X}_1 + \beta_2 \mathrm{X}_2 + ... + \beta_i \mathrm{X}_i + \alpha_{stratum(i)} \tag{4.2}$$

where $\alpha_{stratum(i)}$ is the matching confounder.

NB classification works based on Bayesian rules as defined by Equation 4.3. The classifier is famous for its simplicity and fast computation. The classifier works up a set of input metrics (numerical or categorical) as if they are independent from each other. The probability of the response variable is calculated, as shown in Equation 4.3.

$$p(X|Y_k) = p(Y_k) \prod_{i=1}^{n} p(X_i|Y_k) \tag{4.3}$$

where C is the response variable (i.e., fault prone, non-fault prone), and X is the independent variable.

Decision tree J48 works by splitting data based on the most significant splitter (i.e., metric). The splitter is chosen based on the impurity or uncertainty of the data under this subset of data. The decision of splitting is based on calculating the information gain, as shown in Equations 4.4 and 4.5. The information gain subtracts the prior entropy of the selected metric X_i. The classifier continues splitting data until a tree is formed, starting from the root (i.e., all metrics) and ending with leaves or terminal nodes (i.e, metrics that were not split).

$$H[D] = -\sum_{j=1}^{|C|} P(c_j) log_2 P(c_j) \tag{4.4}$$

$$gain(D, A_i) = H[D] - H_{X_i}[D] \tag{4.5}$$

where C is the desired class, and H[D] is the entropy.

RF model is an ensemble tree-based learning algorithm, developed by [170]. The algorithm was inspired by other ensemble classifiers (e.g., bagging, random split selection). The algorithm creates multiple trees and takes a majority voting on the predicted class instead of on a single tree decision [171].

The PART classifier was built based on two dominant rule learning algorithms, decision tree C4.5 and RIPPER [172]. The algorithm produces decision lists, which are sets of rules (i.e., the best leaf of C4.5). In the testing precess, data are compared to each rule and assigning to the class with the best match.

G-Lasso was initially developed to solve prediction problems and large variances associated with ordinary least squared (OLS) [173]. The G-Lasso algorithm subsets the substantial number of features that a model could contain, which helps determine features with strong impacts. The algorithm shrinks some coefficients and turns others into 0. This concept helps gain an optimal fit for a model by leaving important features in the model and eliminating insignificant features, which results in reducing the variance and improving the accuracy of the model.

G-Lasso is an extension of lasso proposed by [174]. The response variable y_i is binary with two levels $y_i \in 0, 1$. Integer, ordinal, and categorical data are allowed in the model. In this study, all features are in integer format, which means each feature involves one degree of freedom. Independent observations of (x_i, y_i), $i = 1, ..., n$ are assumed:

$$(y - \sum_{l=1}^{L} X_l \beta_l) + \lambda = 0, \tag{4.6}$$

where L is the total number of predictors. The optimal tuning parameter ($\lambda \geq 0$) is estimated by running a ten-cross validation of the whole dataset. Then the model is trained using the estimated λ on the training set. The performance of the model prediction is measured by predicting the fault-prone software units on the testing set.

We compare the performance of CLR and G-Lasso with five machine learning algorithms widely used for software fault proneness prediction [34]. LR and NB were heavily used in this area (i.e., 40% of papers used LR and 25% of papers used NB) [34]. Decision tree J48 and RF provided better or comparable results regarding other learners in some studies [39, 38, 40].

4.2.2 Performance metrics

To compare our models, we applied two-folds cross validation and repeat the process 100 times on random samples, as shown in Figure 4.1. The process divided the whole sample based on the data with faulty files (i.e., cases) and fault-free files (i.e., controls). Then it split every part randomly into two equal amounts. It then merged between one split from the faulty files with a split from fault-free files to create fold 1. It merged the other two

splits, faulty and fault-free files, to create the second fold (fold 2). It trained the model using fold 1 on all algorithms and tested on fold 2 and reported performance metrics (e.g., AUC, recall,etc). Then it trained all algorithms on fold 2 and test on fold 1 and reported all performance metrics. This process was repeated 100 times.

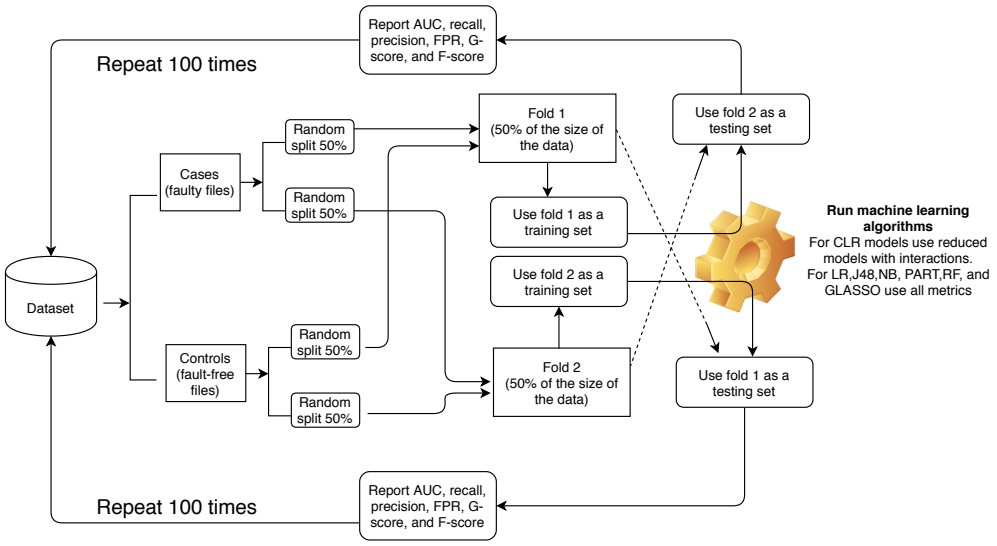


Figure 4.1: Prediction of fault proneness approach repeated 100 times, on random samples

We measured the performance of machine learning algorithms using the metrics computed from the confusion matrix of every model (see Table 4.1). Recall is the rate of all correct predictions over the actual instances of the class of interest (i.e., fault prone files or components), as in Equation 4.7. Precision refers to the correct predictions over all instances that are predicted as the class of interest, as in Equation 4.8. Generally, high recall and precision are preferable. A false positive rate (FPR) is the rate of the false positive over all the actual not-fault prone units (i.e., files or components). FPR is calculated as in Equation 4.9. A lower FPR reflects better performance. F-Score is the harmonic mean of the recall and precision. G-Score is the harmonic mean of the recall and 1-FPR, which results in a high value if the recall is high and FPR is low. Note that F-Score and G-Score are composite metrics that provide a single number that reflects different important aspects of the learners; performance. Some of the related works on software fault proneness prediction used F-Score, while others used G-Score. We chose to use both because they reflect different aspects of performance. Lastly, we used AUC as a performance measure for our models.

Table 4.1: Confusion matrix

		Actual Class	
		fault prone	non-fault prone
Predicted Class	fault prone	True Positive **TP**	False Positive **FP**
	non-fault prone	False Negative **FN**	True Negative **TN**

$$\text{Recall (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.7}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.8}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{4.9}$$

$$\text{F-Score} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \tag{4.10}$$

$$\text{G-Score} = \frac{2 \times \text{Recall} \times (1 - \text{FPR})}{\text{Recall} + (1 - \text{FPR})} \tag{4.11}$$

4.2.3 Statistical comparisons of results

In this study, we applied three sets of statistical tests to analyze differences among the performance of all classifiers.

The Friedman test is the non-parametric one-way ANOVA with repeated measures used to test the differences in several groups. We used the Friedman test to see whether there were significant differences among multiple algorithms' performance [175]. Rejecting the null hypothesis means that a statistically significant difference exists at least between one pair of algorithms. In cases when the Friedman test rejected the null hypothesis, the Nemenyi post-hoc test was used to determine where the differences were located (i.e., to test the statistical significance between each pair of algorithms) [175]. We also used the critical difference diagrams [175] to visualize the ranks of the algorithms based on the recall, G-Score, and F-Score performance metrics. In addition, these diagrams show if the differences are statistically significant.

4.3 Datasets and Features Definition

In this study, we used 27 releases from four open source projects distributed as follows: seven releases of Eclipse, seven releases of Apache Ant, nine releases of Apache Derby, and four releases of Apache Xalan.

Table 4.2: Training and testing samples of the Eclipse and Apache releases

Eclipse					
	Release date	**No. of files**	**Fault prone files**	Sample size	File prone file
Europa	June 2007	31,484	23%	2000	50%
Ganymede	June 2008	31,648	17%	2000	50%
Apache					
Derby.10.1.3.1	July-2006	1,919	9%	306	153
Derby.10.4.1.3	April-2008	1,528	5%	292	146
Derby.10.5.1.1	May-2009	1,494	5%	280	140
Derby.10.6.1.0	May-2010	2,191	4%	252	126
Derby.10.8.1.2	May-2011	2,311	5%	348	174
Derby.10.8.3.0	January-2013	2,426	3%	222	111
Ant16	December-2003	643	22%	196	98
Ant18	February-2010	820	11%	164	82
Xalan 24	April-2011	770	18%	256	128
Xalan 26	April-2013	905	9%	252	126

In Table 4.2, the releases included in this study are listed with their total number of files, and percentages of faulty files. Project sizes vary from several hundred files, such as with Apache Ant, to several thousands, such as with Eclipse. Typically, the number of files increases in later releases compared to older releases, with a few exceptions in Apache Derby and Eclipse.

4.3.1 Features

In this study, we used a set of commonly used change metrics (i.e., features) because it was shown that they provide better performance than static code metrics [21]. Specifically, we used 17 change metrics, which are listed and defined in Table 4.3. The metrics for Eclipse were extracted for the previous work [38, 2]. Change metrics for Apache projects were extracted by Mohammad J. Ahmad and used in [159]. The second reason we used the

change metrics was to be consistent across all data sets, because the static code metrics of Eclipse [38, 2] are different from the object-oriented static code metrics extracted for Apache projects [159]. Therefore, it was important to use the same set of metrics across all projects to maintain the construct validity.

We stratified the sample based on the LOC, which gave our sample a similar pattern of LOC in the faulty files and fault-free files. Note that LOC was added only in the case-control models as a stratified metric and not a predictor. In all other algorithms, LOC was not added because it is part of the static code features, which were not used in this chapter.

Table 4.3: Metrics at file level used as features

Static code metrics: Used only for case-control models	
Metric	**Definition**
LOC	Total number of lines
Method Call Statements	All method calls, in statements and logical expressions
Average Complexity	Sum of all method complexity values divided by the number of methods
Number of Public Methods (NPM)	All methods in a file that declared as public
Average complexity method (AMC)	The average method size for each file, which is the number of Java binary codes in the method
Change metrics [2]	
Metric	**Definition**
Revisions	Number of revisions made to a file
Refactorings	Number of times a file has been refactored
Bugfixes	Number of times a file was involved in bug fixing (pre-release bugs)
Developers	Number of distinct authors who revised the file
LOC Added	Sum of all revisions of the number of LOC added to the file
Max LOC Added	Maximum number of LOC added for all reversions
Ave LOC Added	Average LOC added per reversion
LOC Deleted	Sum of all revisions of the number of LOC deleted from the file
Max LOC Deleted	Maximum number of LOC deleted for all revisions
Ave LOC Deleted	Average LOC deleted per revision
Codechurn	Sum of (added LOC - deleted LOC) over all revisions
Max Codechurn	Maximum codechurn for all revisions
Ave Codechurn	Average codechurn per revisions
Max Changeset	Maximum number of files committed together to the repository
Ave Changeset	Average number of files committed together to the repository
Age	Age of a file in weeks (counting backwards from a specific release)

4.4 Results and Discussion

In this section, results of all performance metrics are presented for all datasets. Seven classifiers were used on 12 releases from four projects, Eclipse, Apache Derby, Apache Ant, and Apache Xalan. For every dataset, a boxplots figure is provided, which contains all performance of all performance metrics (i.e., AUC, recall, precision, FPR, F-score, and G-score). Each boxplot in the figure describes 200 observations resulting from 100 runs of the 2-fold cross validation. Comparing all classifiers to determine the best and the worst performing ones is the interest of this chapter. Specifically, extra focus is given to compare CLR and GLASSO performance to the performance of other learners. This helps address the research questions presented at the beginning of this chapter (**RQ1**, **RQ2**, **RQ3**, **RQ4**, and **RQ5**).

First, the results of all datasets are presented in Figure 4.2. The total number of releases presented by this figure is 12 releases: Eclipse, Ganymede, six releases from Derby project, two releases from Ant project, and two releases from Xalan project. The total number of iterations represented by each boxplot of the figure is 2,400. The recalls of all classifiers are comparable, and their medians range between 0.62 to 0.68. The precision of RF has the highest median (= 0.73), followed by the precision of CLR, J48, LR, and PART (median = 0.65). The overall best performing classifiers in terms of the FPR are RF (median = 0.18) and CLR (median = 0.22). The F scores are affected by the bad performing recalls or precisions. NB has shown the worst F-score because it performed with low precision. Other classifiers performed similar to each other in terms of the F-scores. G-scores of CLR, G-Lasso, and RF were the best with medians higher than 0.70. Although G-Lasso did not perform well in terms of the FPR, with the highest recall (median 0.68), G-Lasso was able to achieve good G-Score, comparable with CLR and RF.

Friedman test results indicate that there are significant differences (p value < 0.05) among classifiers in terms of AUC, precision, FPR, G-Score, and F-Score. In terms of the recall, all classifiers performed similarly, and there are no statistical differences among them (p value = 0.08).

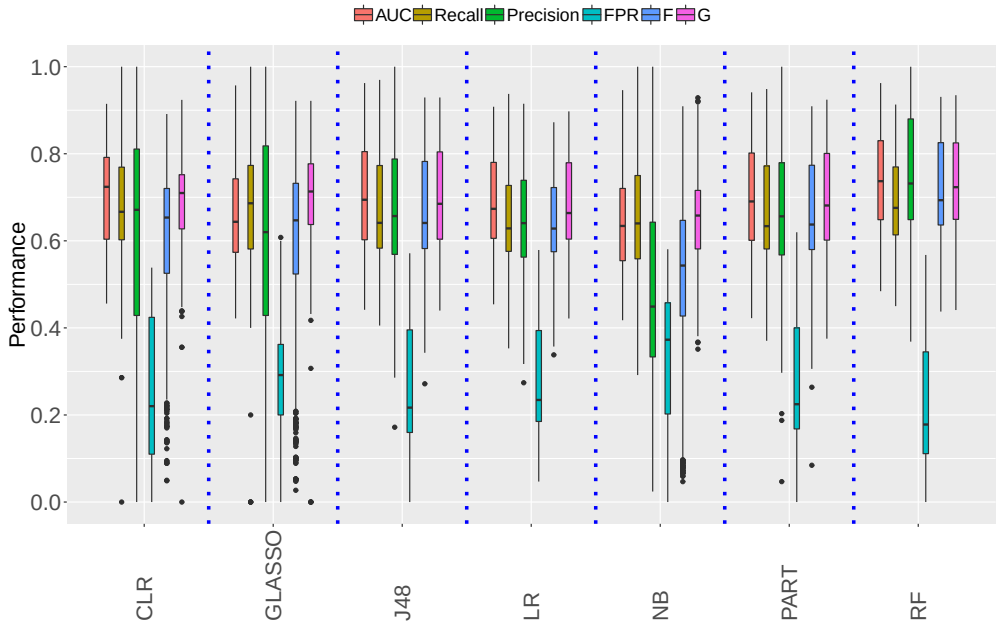


Figure 4.2: Performance metrics of all datasets

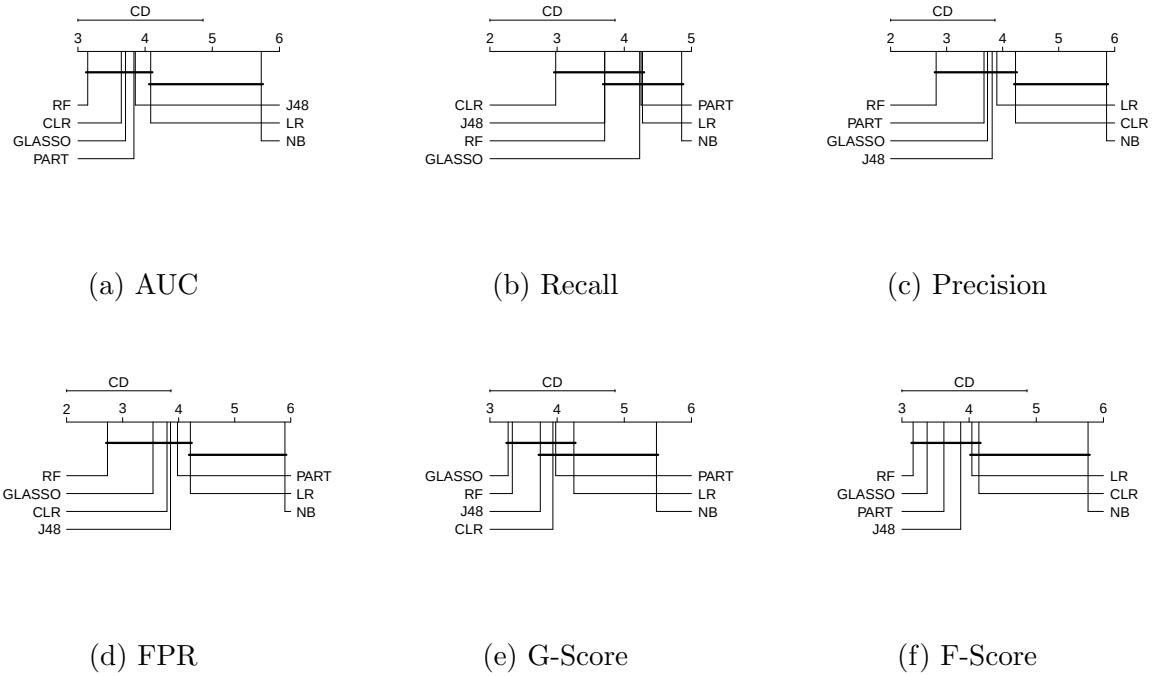
There is no need to apply the Nemenyi post hoc test for the recalls of all classifiers because the Friedman test indicated that the differences among classifiers are not significant in terms of the recall. Nemenyi post hoc test results indicate that differences of AUC, precision, and FPR are significant (p value < 0.05) only between NB and all other classifiers. In terms of the G-Score and F-Score, differences are significant (p value < 0.05) between NB and G-Lasso, J48, PART, and RF.

Figure 4.3 presents critical differences diagrams of all performance metrics on all datasets. The results presented in these figures with the Friedman and Nemenyi post hoc help to address research questions **RQ1**, **RQ2**, and **RQ3**. Theses figures present two important piece of information: the mean ranking of classifiers in terms of performance metric and critical differences among classifiers. The horizontal line connecting two or more classifiers indicates that no significant differences occurred among them.

The RF classifier came at the top of all in terms of the AUC, followed by CLR, G-Lasso, PART, J48, and LR, as shown in Figure 4.3a. With the exception of the NB, all other classifiers are not statistically different. In terms of the recall, CLR is at the top of all other classifiers, followed by J48, RF, G-Lasso, PART, LR, and NB, as shown in Figure 4.3b. CLR, J48, and RF are statistically significantly different from other classifiers. RF leads all the

classifiers in the precision with the best ranking average, followed by PART, G-Lasso, J48, LR, CLR, and NB, as shown in Figure 4.3c. There is no statistically significant difference among all classifiers except the CLR and NB. The mean ranking of the classifiers of F-score is similar to the mean ranking of the classifiers precisions, as shown in Figure 4.3f. The performance of CLR has improved in terms of the F-score and no statistically significant difference was detected between CLR and top classifiers. G-Lasso is at the top in terms of the G-Score, followed by RF, J48, CLR, PART, LR, and NB, as shown in Figure 4.3e. NB has the lowest performance in terms fo the G-score with a statistically significant difference between the NB and all other classifiers. In terms of the FPR, RF has the best (i.e., lowest) mean ranking, followed by G-Lasso, CLR, and J48 with no statistically significant difference among them, as shown in Figure 4.3d.

In summary, apart from the NB classifier, there are no statistical differences among other classifiers in terms of the AUC, precision, FPR, G-Score, and F-Score. In terms of the recall, there are no significant differences among all classifiers.



(a) AUC (b) Recall (c) Precision

(d) FPR (e) G-Score (f) F-Score

Figure 4.3: Critical difference diagrams

Next, results for each individual dataset are presented. This helps identify whether classifiers were affected by the dataset. This can specifically help address **RQ4**.

Results of Europa's model performance are presented in Figure 4.4. CLR has the highest recall with a median of 0.85. The next top recall is with LR, J48, and RF with a median of 0.80. G-Lasso recall of Europa comes next with a median of 0.77. The highest classifier in terms of precision was RF (median = 0.87), and the worst was NB (median = 0.2). Recall is a vital measure because it measures the correct classification over all faulty files. Precision measures the rate of faulty files over all files that were predicted as faulty. Lower precision means that we have more fault-free files that were mistakingly predicted to be faulty files (i.e., false positives). Lower precision is not as critical as low recall, but it means extra manual investigation, which means extra effort in manual classification. Apart from the NB, all the AUC, F-Scores, and G-Scores measures are roughly comparable in all classifiers with no significant differences.

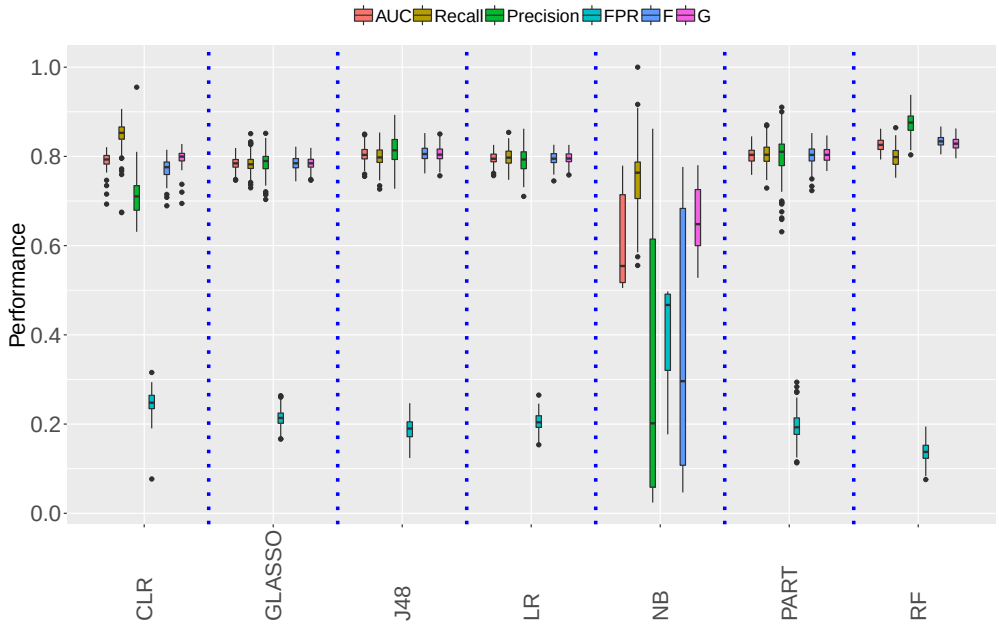


Figure 4.4: Performance metrics for Europa

Results of Ganymede's model performance are presented in Figure 4.5. The recalls of all classifiers are comparable with medians from 0.77 to 0.83. The highest precision median is shown with the RF classifier, and the worst precision appeared with NB. The second top precision medians are with J48, LR, and PART. CLR and G-Lasso precision medians come next with comparable medians. The best-performing classifiers on Ganymede dataset are RF, J48, LR, and PART. CLR and G-Lasso have comparable recalls with the top performing classifiers, but lower precisions, which led to lower F-Score and G-Scores.

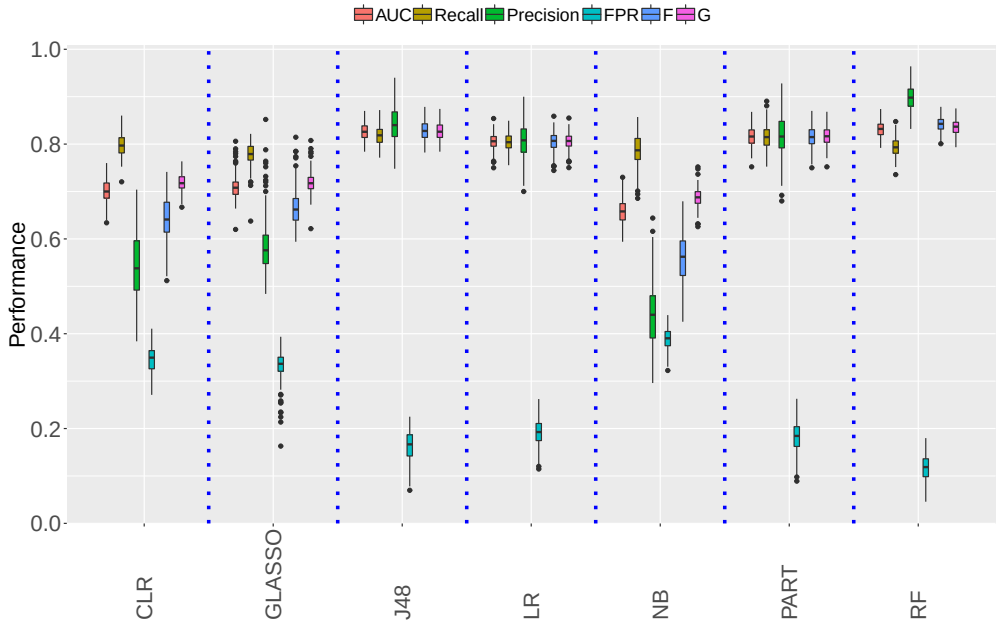


Figure 4.5: Performance metrics for Ganymede

The performance of Derbys' models are presented in Figure 4.6, which describes the six releases used in the case-control study from the Derby project (i.e., Derby 10.1.3.1, Derby 10.4.1.3, Derby 10.5.1.1, Derby 10.6.1.0, Derby 10.8.1.2, and Derby 10.8.3.0), which made the number of iterations for every classifier 1,200. The highest AUC (median = 0.79), precision (median = 0.81), and F-Score (median = 0.72) were reported with CLR. RF AUC and precision were reported as the second highest performing with medians = 0.75 and 0.74, respectively. The best recall median was shown with the G-Lasso (median = 0.79). With the high recall and low FPR, the best G-Scores were reported with CLR, G-Lasso, and RF.

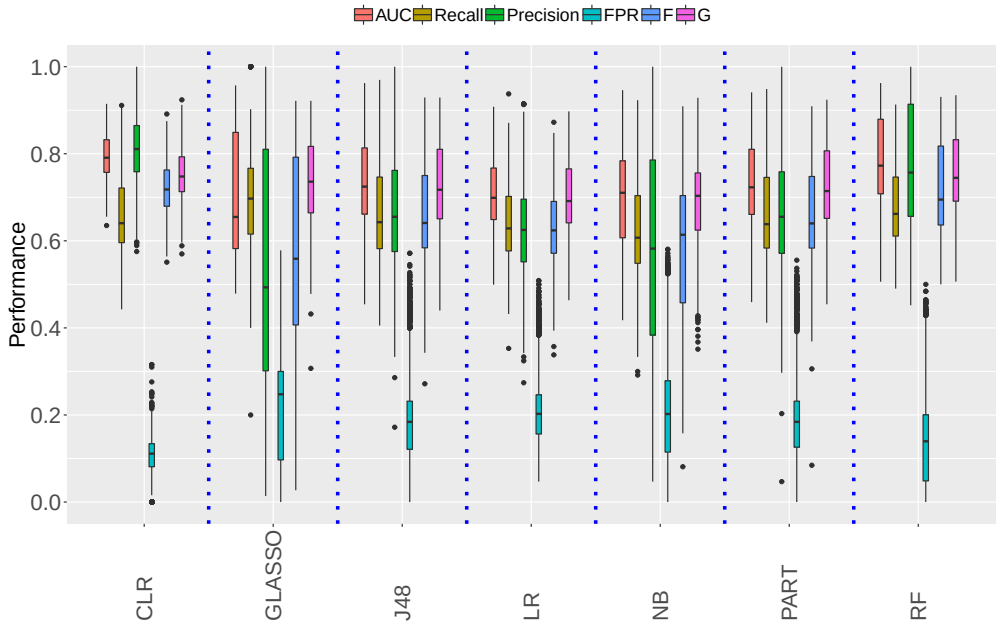


Figure 4.6: Performance metrics for Derby project

For the Ant project, we used only two releases, Ant16 and Ant18, in the case-control study. The results of all classifiers are presented in Figure 4.7. The total number of iterations for every classifier is 400, 200 run for each release. With respect to the recall CLR outperformed (median = 0.75) all other classifiers followed by G-Lasso, NB, and RF (median = 0.65). The AUC of CLR, G-Lasso, and RF were comparable. The highest precision was achieved by RF with a median of 0.69. The precision of the CLR and NB are the lowest, which also led to low F-Scores for the two classifiers. The best G-Score medians are shown with CLR, G-Lasso, and RF with medians from 0.66 to 0.69. The FPR of the RF outperformed other classifiers with the lowest median of 0.31.

The Xalan results are presented in Figure 4.8. We used two releases for this project (Xalan24 and Xalan26), which means there are a total of 400 iterations for every classifier. The recalls of CLR and RF are the top recalls of all (median = 0.63). J48, LR, and PART came next with a recall median = 0.59. The best precision was reported for G-Lasso (median = 0.68), followed by RF (median = 0.65), LR (median = 0.62), J48 (median = 0.60), and PART (median = 0.60). The lowest precision was reported for NB (median = 0.33) and CLR (median = 0.36). This low precision led to a low F-Score for NB (median = 0.43) and CLR

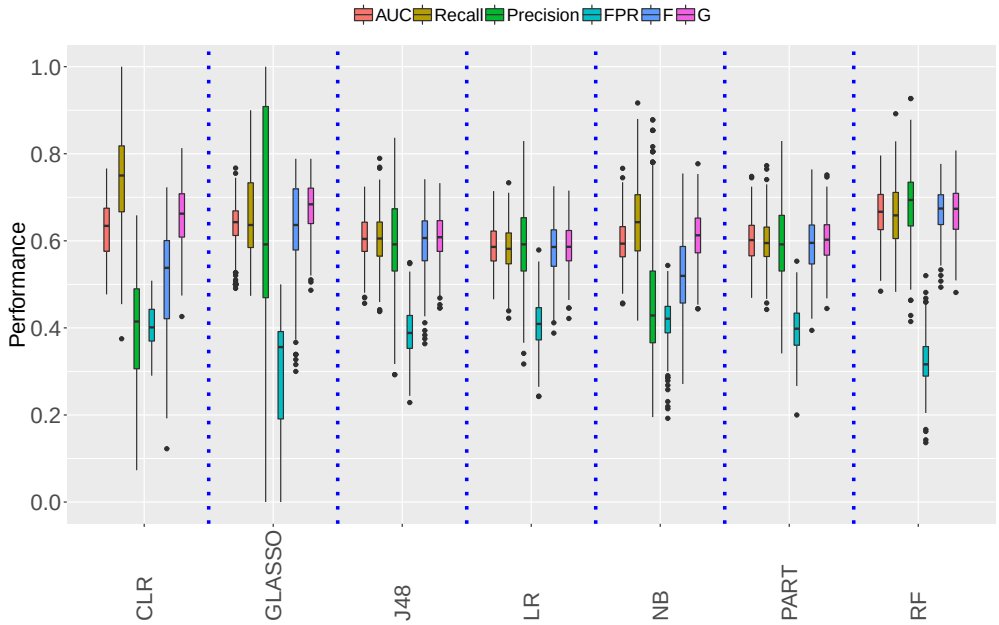


Figure 4.7: Performance metrics for Ant project

(median = 0.46). FPR did not show good results, as with other datasets. FPRs are good when they are close to 0. In Xalan, FPRs were from 0.38 to 0.48. The best FPR median was with RF, and the worst was with NB. Low FPRs with low-performing recalls led to low G-Scores in Xalan.

Classifiers performed well with small variances with Europa and Ganymede. Tables 4.4 and 4.5 present descriptive statistics of CLR and G-Lasso performance, which show that the IQR values of Europa and Ganymede are very low compared to the performance on other projects. Low IQR means that the distance between the 75th percentile and the 25th percentile is low, which means less variance in the results. The CLR performance on Derby project was comparable to the CLR on Ganymede with high variabilities (i.e., IQR > 7). In Derby project, we applied six releases and two releases from Ant and two from Xalan. The CLR performance on Derby was better than the CLR on Ant and Xalan with lower variabilities. Classifiers used in Ant and Xalan reported low performances and high variances.

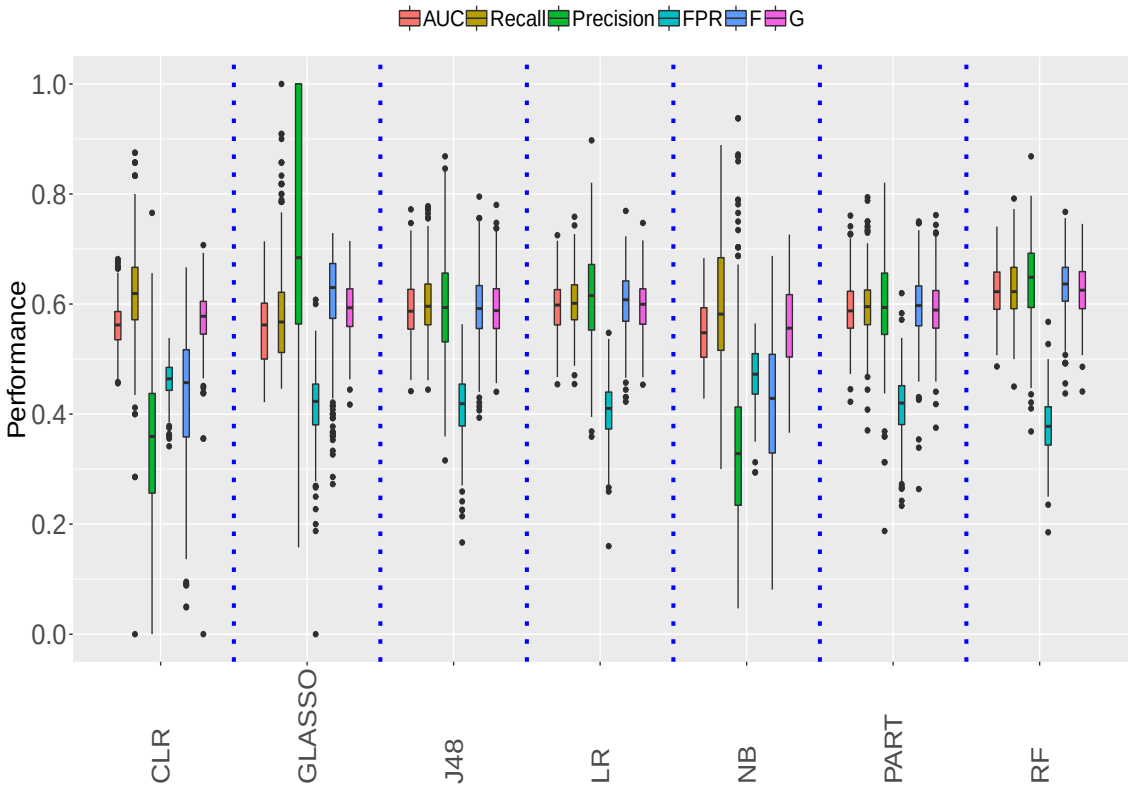


Figure 4.8: Performance metrics for Xalan project

The answer to **RQ4** suggested that CLR and G-Lasso performance are affected by the dataset used. The median recall of CLR on Europa is 0.85, and it is 0.80 on Ganymede. The median recall of the G-Lasso on Europa and Ganymede is 0.78. The median recall of the CLR dropped on Derby and Xalan to 0.64. G-Lasso recalls dropped on Ant and Xalan projects. In general, performances of all classifiers improved when used with a large sample (e.g., Europa, and Ganymede) or when multiple releases were applied (e.g., Derby). Additionally, using large data sets and multiple releases resulted in small variances, which means more reliable results.

Next, we compared our models using CLR with other classifiers from related studies, which performance is in Tables 4.6 and 4.7.

Table 4.4: Performance of the CLR per project

Project/Release	Measure	N	min	max	mean	median	variance	IQR
Europa	AUC	200	0.69	0.82	0.79	0.79	0.00	0.02
	Recall	200	0.67	0.90	0.84	0.85	0.00	0.03
	Precision	200	0.63	0.95	0.71	0.71	0.00	0.06
	G-Score	200	0.69	0.82	0.79	0.82	0.00	0.01
	F-Score	200	0.68	0.81	0.77	0.77	0.00	0.02
Ganymede	AUC	200	0.63	0.76	0.70	0.70	0.00	0.03
	Recall	200	0.72	0.86	0.79	0.79	0.00	0.03
	Precision	200	0.38	0.70	0.54	0.53	0.00	0.10
	G-Score	200	0.66	0.76	0.71	0.71	0.00	0.02
	F-Score	200	0.51	0.74	0.64	0.64	0.00	0.06
Derby (6 releases)	AUC	1,028	0.63	0.91	0.79	0.79	0.00	0.08
	Recall	1,028	0.44	0.91	0.81	0.81	0.00	0.13
	Precision	1,028	0.57	1.00	0.81	0.81	0.00	0.11
	G-Score	1,028	0.56	0.92	0.75	0.74	0.00	0.08
	F-Score	1,028	0.55	0.89	0.72	0.71	0.00	0.09
Ant (2 releases)	AUC	400	0.47	0.76	0.63	0.63	0.00	0.10
	Recall	400	0.37	1.00	0.74	0.75	0.01	0.15
	Precision	400	0.07	0.65	0.40	0.41	0.01	0.18
	G-Score	400	0.42	0.81	0.65	0.66	0.00	0.10
	F-Score	400	0.12	0.72	0.51	0.53	0.01	0.07
Xalan (2 releases)	AUC	400	0.45	0.68	0.56	0.56	0.00	0.05
	Recall	400	0.00	0.87	0.62	0.61	0.00	0.09
	Precision	400	0.00	0.76	0.34	0.35	0.01	0.18
	G-Score	400	0.00	0.70	0.57	0.57	0.00	0.06
	F-Score	400	0.04	0.66	0.42	0.45	0.01	0.16

First, we compare our recalls and precision with recalls and precision applied using the LR in [23] for Eclipse 2.0, 2.1, and 3.0. Our recall medians of all releases outperformed recalls in [23]. Precision medians of Europa, Ganymede, and Derby are comparable with precisions of [23]. Further, our recall medians of all releases except Xalan are higher than J48 classifers used in [21] on Eclipse 2.0, 2.1, and 3.0. Also, our recall medians of all releases outperformed the J48 used in [38]. AUC medians of our CLR are slightly better and comparable to AUCs in [38].

Next, we compared our results with those from many studies using several classifiers for prediction as reported in Malhotra [176]; that review study reported minimum and maximum AUC achieved by every classifier. Our best AUC was reported with Europa and Derby releases. In Europa, the AUC ranged between 0.69 to 0.82, and in Derby, the AUC ranged

Table 4.5: Performance of the G-Lasso per project

Project/Release	Measure	N	min	max	mean	median	variance	IQR
Europa	AUC	200	0.75	0.87	0.80	0.80	0.00	0.03
	Recall	200	0.75	0.87	0.80	0.79	0.00	0.03
	Precision	200	0.72	0.91	0.82	0.83	0.00	0.07
	G-Score	200	0.75	0.87	0.79	0.82	0.00	0.01
	F-Score	200	0.74	0.87	0.80	0.81	0.00	0.01
Ganymede	AUC	200	0.68	0.84	0.78	0.79	0.00	0.04
	Recall	200	0.71	0.83	0.77	0.77	0.00	0.03
	Precision	200	0.56	0.93	0.80	0.82	0.00	0.11
	G-Score	200	0.68	0.84	0.78	0.79	0.00	0.04
	F-Score	200	0.64	0.85	0.78	0.79	0.00	0.05
Derby (6 releases)	AUC	1,028	0.49	0.95	0.74	0.73	0.01	0.27
	Recall	1,028	0.00	1.00	0.72	0.73	0.00	0.14
	Precision	1,028	0.00	1.00	0.61	0.62	0.11	0.70
	G-Score	1,028	0.00	0.93	0.77	0.78	0.03	0.16
	F-Score	1,028	0.04	0.93	0.62	0.65	0.08	0.42
Ant (2 releases)	AUC	400	0.47	0.77	0.63	0.64	0.00	0.06
	Recall	400	0.48	0.90	0.65	0.62	0.01	0.16
	Precision	400	0.00	1.00	0.46	0.59	0.06	0.46
	G-Score	400	0.44	0.78	0.67	0.67	0.01	0.09
	F-Score	400	0.30	0.76	0.62	0.64	0.03	0.14
Xalan (2 releases)	AUC	400	0.45	0.66	0.54	0.54	0.00	0.07
	Recall	400	0.42	0.73	0.55	0.54	0.00	0.07
	Precision	400	0.18	1.00	0.74	0.70	0.04	0.41
	G-Score	400	0.36	0.66	0.57	0.57	0.00	0.06
	F-Score	400	0.29	0.69	0.61	0.64	0.01	0.10

between 0.63 to 0.91. Our range resided in the same range of all studies using classifiers reported in [176]. The mean and median of of AUC in Europa and in Derby using CLR were higher than the mean and median reported for MLP, NB, SVM, and C4.5 classifiers reported in [176]. AUC mean and median of RF in [176] were higher than our AUC mean and median using CLR. This is consistent with our results because our CLR outperformed RF in recall but not in AUC.

It is important to note that the final models developed in this study are different than others in three ways. First, interactions were involved in our models, but not in other models. Second, the algorithm we used in this study (i.e., CLR) matched between files, whereas other algorithms did not. Third, the response variables in our models were balanced (i.e., 50% fault prone and 50% no-fault prone) because of the matched sample approach (i.e., one to

Table 4.6: Performance (precision, recall , accuracy, and AUC) of this study and related studies

		Method	Accuracy	Recall	Precision	AUC
Zimmermann et al. [23]	Eclipse 2.0	LR	0.76	0.24	0.65	x
	Eclipse 2.1		0.64	0.21	0.78	x
	Eclipse 3.0		0.71	0.37	0.66	x
Moser et al. [21]	Eclipse 2.0	J48	0.82	0.69	x	x
	Eclipse 2.1		0.83	0.60	x	x
	Eclipse 3.0		0.80	0.65	x	x
Krishnan et al. [38]	Eclipse 2.0	J48	0.79	0.52	63	74
	Eclipse 2.1		0.81	0.46	0.63	0.73
	Eclipse 3.0		0.80	0.38	0.63	0.71
	Europa 3.3		0.84	0.25	x	0.65
	Ganymede 3.4		0.88	0.40	x	0.75

one). All these factors may have contributed to the performance of our models. Therefore, we conducted a comparison with other studies [169, 177] in Table 4.7. The two studies used imbalance treatment, which caused the classes of response variables they used for their models to be equally distributed (i.e., 50% fault prone, 50% no-fault prone). This comparison was made to avoid the possibility that the performance of our model was due to the balanced distribution of the response variable.

Table 4.7: Performance (AUC) of related studies applying top-performing classifiers and imbalance treatment

			Method	AUC			
				min	max	mean	median
Malhotra [176]	Number of studies	35	RF	0.66	1.00	0.83	0.82
		40	MLP	0.54	0.95	0.78	0.77
		47	NB	0.64	0.95	0.78	0.78
		25	Bayesian networks	0.62	0.90	0.78	0.79
		17	SVM	0.50	0.94	0.70	0.71
		20	C4.5	0.50	0.99	0.77	0.77
				min	max	mean	median
Bennin et al. [177]	20 datasets by 7 sampling techniques		RF with 50% fault prone	0.16	0.92	x	x
			nnet with 50% fp	0.24	0.88	x	x
			knn with 50% fp	0.24	0.88	x	x
			SVM with 50% fp	0.36	0.94	x	x
			C4.5 with 50% fp	0.27	0.93	x	x

The results of our models lean toward the maximum values of the several models in the two studies [169, 177]. This means the balanced distribution of the response variable may not be the major reason for the high performance of our models. Also, matching between files may not be the reason either because we compared the same models with other classifiers that did not involve matching, and we had comparable performance. The only explanations of the good performance are the methodology used for building the model, the goodness of fit for the models used for prediction, and involving interactions. Interactions are important to consider in prediction as well as in explanation. Exploring this may be a good topic for the future in this area and other areas. It is also important to consider the methodology used for keeping significant metrics and their interactions in the model and to eliminate unnecessary elements. Also, it is essential to test for the goodness of fit for the final model because this affects the prediction performance.

Table 4.8: Summary of the research questions

RQ	Description	Result	Evidence
RQ1:	Does CLR perform better than other classifiers?	No	CLR provided comparable performance to all other classifiers in terms of all performance measures. There were no statistical differences between CLR and other classifiers in terms of all performance measures except with the NB. CLR outperformed NB in terms of the AUC, precision, G-Score, F-Score, and FPR.
RQ2:	Does G-Lasso perform compared to other classifiers?	No	G-Lasso provided comparable performances to all other classifiers in terms of all performance measures. No statistical differences were detected between G-Lasso and all other classifiers. Similar to the CLR, G-Lasso outperformed NB in terms of the AUC, precision, G-Score, F-Score, and FPR.
RQ3:	What is the ranking of classifiers in terms of the performance measures (i.e., recall, precision, G-Score, F-Score, and AUC)?		RF is first in terms of the AUC, followed by the CLR and G-Lasso. CLR is ranked first in terms of the recall, followed by the RF, J48, and G-Lasso. RF is the first in terms of the precision, followed by PART and G-Lasso. In terms of the FPR, RF, G-Lasso, and CLR are the top ranked classifiers. In terms of the G-Score, G-Lasso, followed by RF and J48. In terms of the F-Score, RF followed by G-Lasso, and PART.
RQ4:	Does a particular dataset affect the prediction performance of the CLR?	Yes	The best performance was when dataset used with large sample size such as in Europa and Ganymede. Small sample and less number of releases reduce the overall performance.
RQ5:	Does CLR using reduced models with interactions (i.e., achieved by the case-control methodology) perform better than other algorithms used in related studies?	Yes	CLR performed better than related studies applied MLP, NB, SVM, C4.5, and J48. Also, it performed close to the top-performing levels of models applied 50% fp imbalanced treatment with algorithms like RF, nnet, knn, SVM, and C4.5.

4.5 Threats to Validity for the Software Fault Proneness Prediction

We took the necessary steps to ensure that the construct validity was not violated such as splitting data for training and testing, and using 50% of the original data for testing the models. With the exception of the CLR, we used the same set of features for all other classifiers. The reason for that is because explanatory models was built earlier and included two features from the static code metrics as described in the previous chapter. Our goal is to measure the performance of these models without introducing any change in the features applied.

To address the internal validity, we ensured that the data were of high quality. For the extracted change metrics (i.e., features), we used sanity checks. For random samples of change metrics, for each project, we manually compared the values of change metrics with their actual values in the commit and source code files.

With respect to the conclusion validity, we used multiple performance metrics to compare the software fault proneness prediction results. This is important because, as our results showed, an algorithm can have good performance with respect to one metric (e.g., G-Score), but bad performance on another metric (e.g., F-Score). Using multiple performance metrics provides a complete picture of an algorithms' performance and allows projects to select the algorithm with the best performance on metrics of their interest.

With respect to the conclusion validity, we used the non-parametric statistical tests appropriate for the distribution of our data. For some releases that had heavily imbalanced datasets, if a certain algorithm failed to provide meaningful classification (all instances were classified in one class) for a given release, that release was excluded from the analysis. This reduced the number of releases from 27 considered initially to 14 releases that were used for the analysis. In addition, we used multiple performance metrics to compare the software fault proneness prediction results. This is important because, as our results showed, an

algorithm can have good performance with respect to one metric (e.g., G-Score), but bad performance on another metric (e.g., F-Score). Using multiple performance metrics provides a complete picture of algorithms' performance and allows projects to select the algorithm with the best performance on metrics of their interest.

The external validity is related to the generalizability of the results. For this study we used releases from Eclipse and three Apache projects. Some of the research questions considered in the paper, such as comparing the performance of multiple learners, were considered earlier using the NASA MDP dataset [40]. In addition, wherever relevant, we compared our results with those in related works.

4.6 Conclusion for the Software Fault Proneness Prediction

In this chapter, we used the explanatory models built in the previous chapter for prediction and compared the results with six other classifiers, including G-Lasso which was applied for first time in this area. CLR and G-Lasso showed comparable performance to other classifiers without any significant differences, and they outperformed the NB classifier. Additionally, we found that data sets affect the performance of CLR and G-Lasso, which performed well with large samples and on multiple releases. However, their performance was affected using smaller samples. A summary of the answers for the research questions RQ1, RQ2, RQ3, RQ4, and RQ5 were answered in Table 4.8.

Future work will include using G-Lasso on other data sets and using different sets of metrics to further explore the generalizability of our findings.

Chapter 5

Explanatory and Prediction Studies of Software Development Effort

This chapter covers explanatory and prediction studies of software development efforts using three datasets: International Software Benchmarking Standards Group (ISBSG), Desharnais, and Maxwell. First, this chapter introduces the topic and explains the motivation for the work in Section 5.1. The methodology is briefly discussed in Section 5.2, and the steps of the methodology are explained in detail as we discuss our first case study, ISBSG, in Section 5.3. The second and third case studies are discussed in Sections 5.4, and 5.5, respectively. The prediction performance are presented in Section 5.6. Then, this chapter explains threats to validity in Section 5.7 and concludes the work in Section 5.8.

5.1 Introduction and Motivation

Software effort estimation is a very critical issue in the software development life cycle. Good estimation results in a very good project plan with respect to cost, time, and resources. Efforts in software projects are measured by the amount of average work achieved by a developer in one hour (i.e., man-hour). By estimating the correct amount of man-hours, the project manager can determine the amount of work needed for the project and plan the budget and time accordingly. Estimation is considered good when the estimated values are very close to actual efforts. The most common estimation methods can be classified under four main categories: expert guessing, estimation based on analogy, and prediction using machine learning methods and historical data, and algorithmic-based methods using mathematical equations [80]. Achieving a good estimation seems to be a problem for the software industry. It has been reported that 60% to 80% of software projects encounter effort, schedule, and cost overrun [178]. Additionally, the average effort overrun is 36%, and the average schedule overrun is 22% [179]. Further, 45% of projects are between 25 and 50% over- or underestimated [180]. Only 22% of projects have been reported to be within 5% of the correct estimation [180]. Overestimating the development effort of a project is as bad as underestimating it because if other companies are also bidding for the same project, overestimation can cause a company to lose the bidding.

Understanding the reasons for the increase in software development efforts is crucial and can be helpful for project managers in estimating efforts or planning their projects. For example, changes in requirements may cause a significant increase in effort [180]. Further, some issues related to project management and team skills can potentially increase effort [180]. Other important factors are related to overoptimism and lack of a formal process of estimation [180]. Overoptimism occurs when practitioners consider the best-case scenario to get the best price and ultimately win the project. Therefore, it is important to consider the worst and best scenarios in the estimation. This can be achieved by estimating an interval (i.e., range) rather than estimating a number, which involves dividing the numerical values of variables to levels. Every level contains an upper and lower limit that can be easily interpreted.

Research in this area, has focused on prediction and has not paid attention to explaining what metrics cause an increase or decrease in effort and by how much the metrics contribute to efforts. To be able to provide such explanations, quantifying the probability of effects of different levels of these metrics is essential. Further, explaining and predicting the probability of the effects of interactions of different metrics' levels and determining how they increase or decrease efforts. Interactions provide insight into how the relation between two independent metrics can contribute to efforts and can explain beyond what metrics cannot.

We can achieve our approach by building a model that can fit the data, estimate OR, and explain metrics and interactions. Different types of algorithms can help to achieve that. For example, different types of linear models are helpful to determine how close numerical predicted values are to actual efforts calculated in man-hours. The problem with linear models is that they are restricted to the distribution of the data assumption. Model estimation is affected by how much data are skewed. One of the alternative approaches is to consider the discretization of data and fit the data into an ordinal regression instead of using linear regression. The discretization should be carefully done using a statistical method that considers significance between levels by creating multiple cut-points that are significantly different from each other. Therefore, instead of dealing with a specific number, we deal with levels, which makes estimation and explanation easier. The advantage of using discretized data is that this allows us to deal with a high level of information (e.g., size of a project: large, medium, and small) instead of a lower level (e.g., number of lines of code) [181]. We used the ChiMerge technique to discretize numerical variables into multiple levels with the condition that every level would be statistically significantly independent from its neighbors. Models of this nature are very helpful for explaining and quantifying the probability of the existence of every level of every independent variable in the model. For this approach, we need to follow several steps concerning data preprocessing, variable selection, multicollinearity, and building a final model that can accurately explain data.

We create an ordinal regression model that explains which metrics and interactions significantly affect the software development effort. This includes quantifying probabilities of multiple levels of different metrics and their interactions. Odds ratios (OR) for these metrics and their interactions help to quantify the probability of increase or decrease these metrics

and their interactions. For example, we have two possible levels of a metric X: X_1 and X_2. When the probabilities of the two levels are equal, OR should produce a number around one. Level X_1 has higher probability than X_2 when the OR produces a number higher than one. Regarding the percentage determined by the value of the OR, every 0.01 over one is considered to be 1%. For example, OR=1.2 means a 20% increase, and an OR = 2.2 means 120%. Level X_1 has lower probability than X_2 when the OR is produces a number lower than one. OR less than one cannot be less than zero, and every 0.01 below one is also considered to be a 1% decrease. For example, an OR=0.5 means that the probability for Level X_1 is less than 50% of the probability of Level X_2.

In this chapter, we address the following research questions:

RQ1: Which metrics significantly affect the software development effort?

RQ2: Are there any interactions among the metrics that significantly affect the software development effort?

RQ3: Does the prediction of the ordinal regression algorithm perform better than predictions of other algorithms used in related studies?

There are several challenges associated with software development effort studies. First, the size of data is usually very small, especially data that are publicly available, such as COCOMO and the NASA data sets. Using this data may result in low statistical power. The second problem deals with the number of missing values in every data set and metrics. Any statistical model will ignore the entire row if a single value in that row is missing. Therefore, this problem needs to be handled very carefully either with data imputation or deletion. Third, software effort data sets usually contain a mixture of numerical, binary ordinal, and categorical data. This mixture should be treated either using the same data type or using a proper method to handle the data conversion. Fourth, few studies apply the modeling technique to explain efforts. Some early studies tried to explain efforts using data based on

questionnaires [182, 179, 183, 184, 185, 186, 81]. Fifth, the effect of interactions among metrics on software efforts is not considered. Only three previous works considered interactions for their models, but their objective was to improve the accuracy of their predictive models [101, 102, 103].

The work presented in this chapter overcomes the challenges as following: The work includes a combination of data preprocessing; statistical tests for association, discretization, and data imputation; and modeling based on an ordinal regression algorithm. Furthermore, we validate our models using well-know performance measures (i.e., Mean Magnitude of Relative Error MMRE, Median of the Magnitude of Relative Error MdMRE, and Percentage of Predictions PRED(25)). Additionally, other classification performance measures are used (i.e., recall, precision, and F-score). We first apply the modeling approach on the ISBSG data set, and then we replicate the same approach for the Desharnais and Maxwell data sets. The proposed approach combines several methods to reach the final optimal model that will explain metrics and interactions and predict the level of effort. The following list summarizes our holistic approach:

- Treat the missing values using the k-nearest neighbors (k-NN) algorithm to impute missing data;
- Conduct discretization of numerical data using the statistical tool ChiMerge [187];
- Select or eliminate independent metrics based on a pair-wise correlation test;
- Select or eliminate categorical metrics using chi-square test conducted on the contingency table of pairs of metrics;
- Design an explanatory fitted categorical model, based on the backward hierarchical modeling strategy [148], that can help explain which metrics and interactions contribute to the increase or decrease of the software development effort;
- Conduct the multicollinearity diagnoses using perturbation, which is recommended to measure any sign of collinearity for ordinal regression and a model with interactions [188, 189]; and

- Test for the goodness of fit of the final model and measure performance and multi-class accuracy.

The key findings of all the three data sets used in this study are summarized in the following list:

- Software development effort increased, as the size (measured in function point FP), elapsed time, and team size of projects increase.
- Faults in software projects and complexity of software led to increase software development effort.
- Using an advanced programming language reduced software development effort.
- High requirements of staff availability, efficiency, and installation increased software development efforts.
- The interaction between the size of the project and the size of the team working on the same project increased the effort.
- The interaction between the existence of faults in the project and time to deliver a project decreased the effort.
- The interaction between time and team size decreases the effort.
- The interaction between staff availability and efficiency requirements decreased the effort.

5.2 Methodology

In this study, we use the total number of efforts calculated in man-hours as a response variable. This metric and other independent metrics are preprocessed to fit into the ordinal regression model. Figure 5.1 summarizes the methodology proposed in this work, which is divided into two main subprocesses: data preprocessing and building the model.

In the data preprocessing phase, we start with the initial selection of metrics based on the prior studies' findings. Second, we prepare our data set and impute missing values using the k-NN method. Third, we discretize numerical values using the ChiMerge method. Fourth, we conduct a correlation test between ordinal metrics using the Spearman test and eliminate metrics with high correlation. Last, we measure the association between categorical metrics and ordinal metrics and eliminate metrics with high association.

The second stage is building the model. In this phase, we achieve the final model after eliminating insignificant metrics and interactions. For this step, we need to follow the hierarchical backward elimination process [148]. The process states that we should start with the highest order (i.e., interactions) and then move to the lowest order (i.e., main metrics). First, we need to confirm that the initial model contains no multicollinearity issues. Therefore, we build the initial model, which includes all selected metrics and interactions. Second, we test for multicollinearity using the perturbation test. Then, we eliminate insignificant metrics, starting with the least significant. After we reduce all trivial interactions, we verify whether there is any primary metric that we need to eliminate. Then, we have our final model, which we test for accuracy and goodness of fit.

5.3 First Case Study: ISBSG

We use data from the ISBSG. The ISBSG repository has grown in both the number of projects and the number of variables over the last twenty years [190]. The April 2016 release contains 7,518 projects and 264 metrics. Metrics are organized into several categories: project, grouping, sizing, schedule, effort, quality, and more. Around 350 works used the ISBSG data set, and most of them were published between 2005 and 2013 [190].

5.3.1 Data Preprocessing

We start with the exclusion criteria concerning the quality of data. In ISBSG, the quality of the data is represented by four classes: A, B, C, and D. Classes A and B have the highest integrity and are recommended to be used [81, 191]. Excluding C and D projects reduced the number of projects in our data set from 7,518 to 7,058.

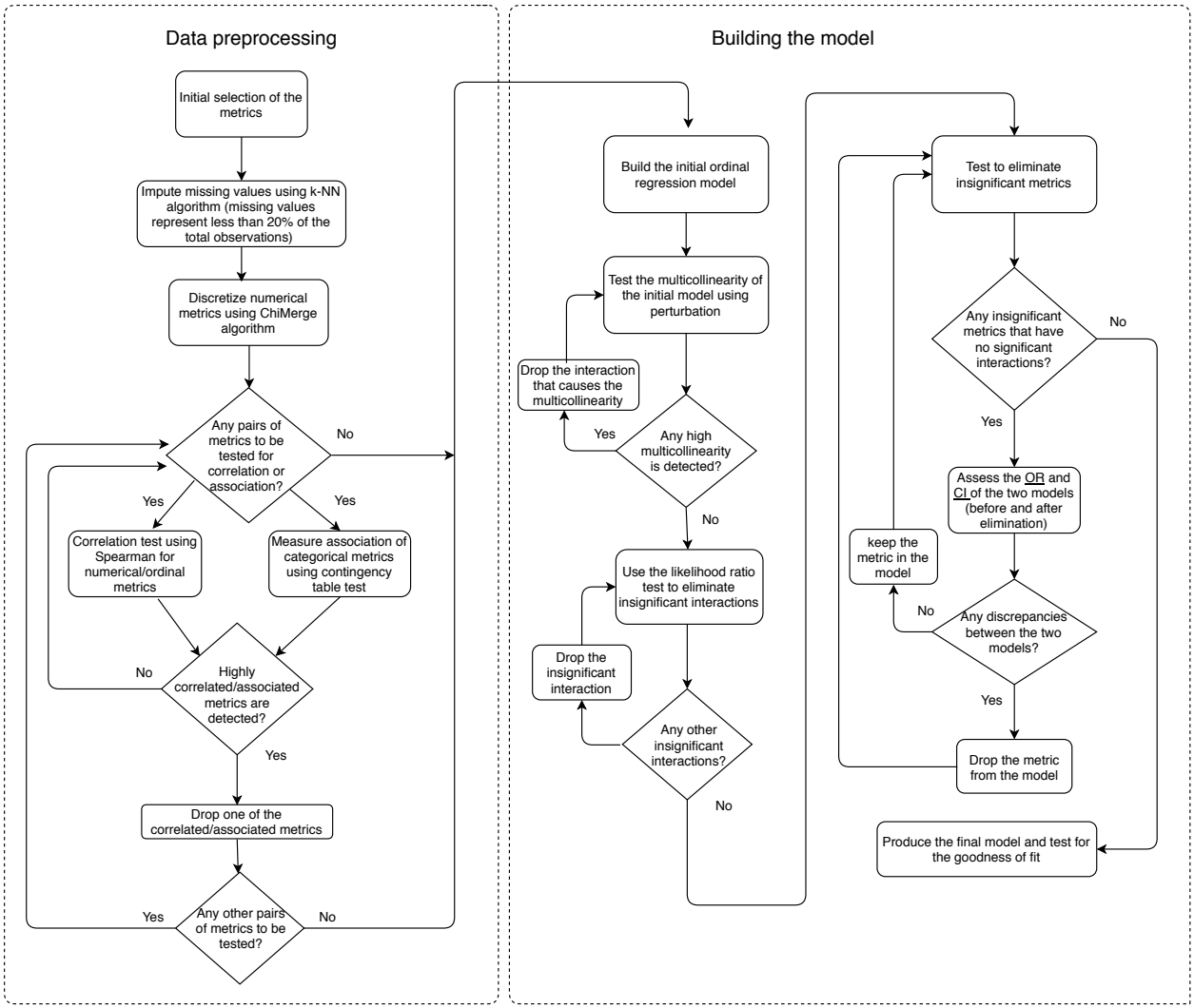


Figure 5.1: Summary of the methodology of this research

We selected this study's metrics on 1) earlier works, 2) results of the correlation test, and 3) not having more than 20% of missing values. The response metric is summary work effort (SWE), which refers to the total effort in hours recorded against the project. The SWE is the sum of the efforts that are used in planning, specifying, designing, building, testing, and implementing the project. The SWE was a response variable in 89% of the studies described in [190]. The project size in the ISBSG data set is represented using functional size, which was chosen in 62% of studies [190]. Functional size is defined as the product of unadjusted function points (FPs) and the complexity of the adjustment factor. Faults are the number of software faults delivered with a software project. The project faults metric has been used by a minimal number of studies (1%) [190]. Although the project faults metric has not been

explored by many studies in the area, we believe that this metric has affects efforts. The elapsed time metric is calculated from the planning date to the implementation date. Time Elapsed is among the top ten ISBSG metrics most commonly used in software development efforts [190]. We also consider the team size metric, which has been used by 28% of ISBSG studies [190].

The final set of selected metrics is already notated and defined in Table **??**.

Table 5.1: Metrics definitions

Metric	Notation	Definition
Summary Work Effort	SWF	Total number of efforts recorded in hours
Functional Size	Size	Total number of function points
Project faults[1]	Faults	Total number of faults delivered with the project
Elapsed Time	Time	Duration of the project in months
Max Team Size	Team	Maximum number of team assigned in the project
Speed of Delivery	Speed	Functional Size Units per elapsed month
Manpower Delivery Rate	Manpower	Functional Size Units per person per elapsed month

Missing Values

Missing values are empty inputs found in the data set that exist for several reasons: data entry errors, unknown values, and loss of data [192]. There are several solutions for handling missing values. The easiest one involves list-wise deletion (i.e., deleting rows contain missing values) and column deletion (i.e., deleting metrics with a large number of missing values). Several studies on effort prediction have used the former deletion method [193, 194, 195, 196]. Instead of deletion, we decided to use imputation, which is a procedure that substitutes the missing values in a data set with some plausible values [197]. The simplest imputation technique is using the set's mean or median for the missing values. Imputation using the mean or the median is not a good option when significant amount of data are missing [192]. In this work, we use an imputation method based on k-NN which has been proven to improve the model fit in effort prediction models [104] and is often used with the ISBSG data set [190]. Additionally, [197] applied k-NN for up to 60% missing values from the whole BUPA data set, and in every metric, only 10% of missing values were imputed.

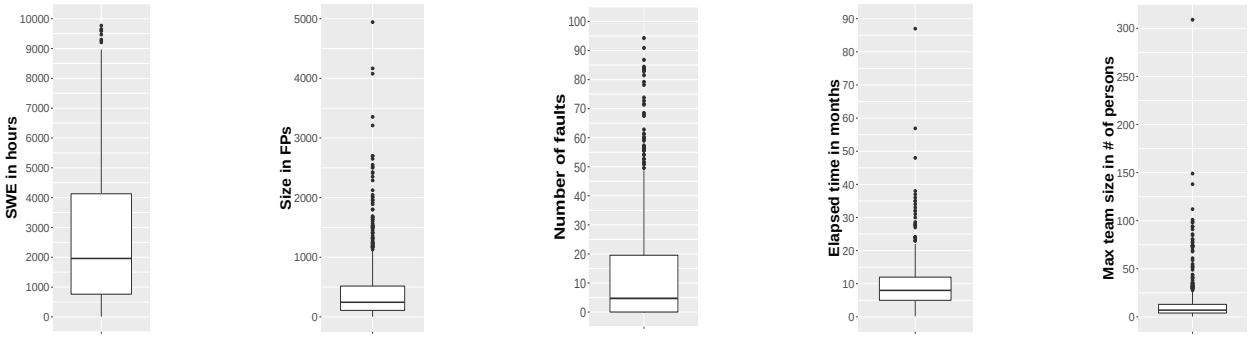
In the ISBSG data set, 584 projects had a complete set of values for our selected metrics. Another 1,887 projects had a complete set of values for all metrics except for project faults and maximum team size. We do not want to impute more than 10% of missing values on every metric as in [197]. Because we have two metrics with missing values, we added 20% projects that were randomly sampled from the 1,887 projects, which gives us a total of 700 projects in our sample. We use the k-NN method to impute the missing values for the two metrics as suggested by [104, 197].

Discretization

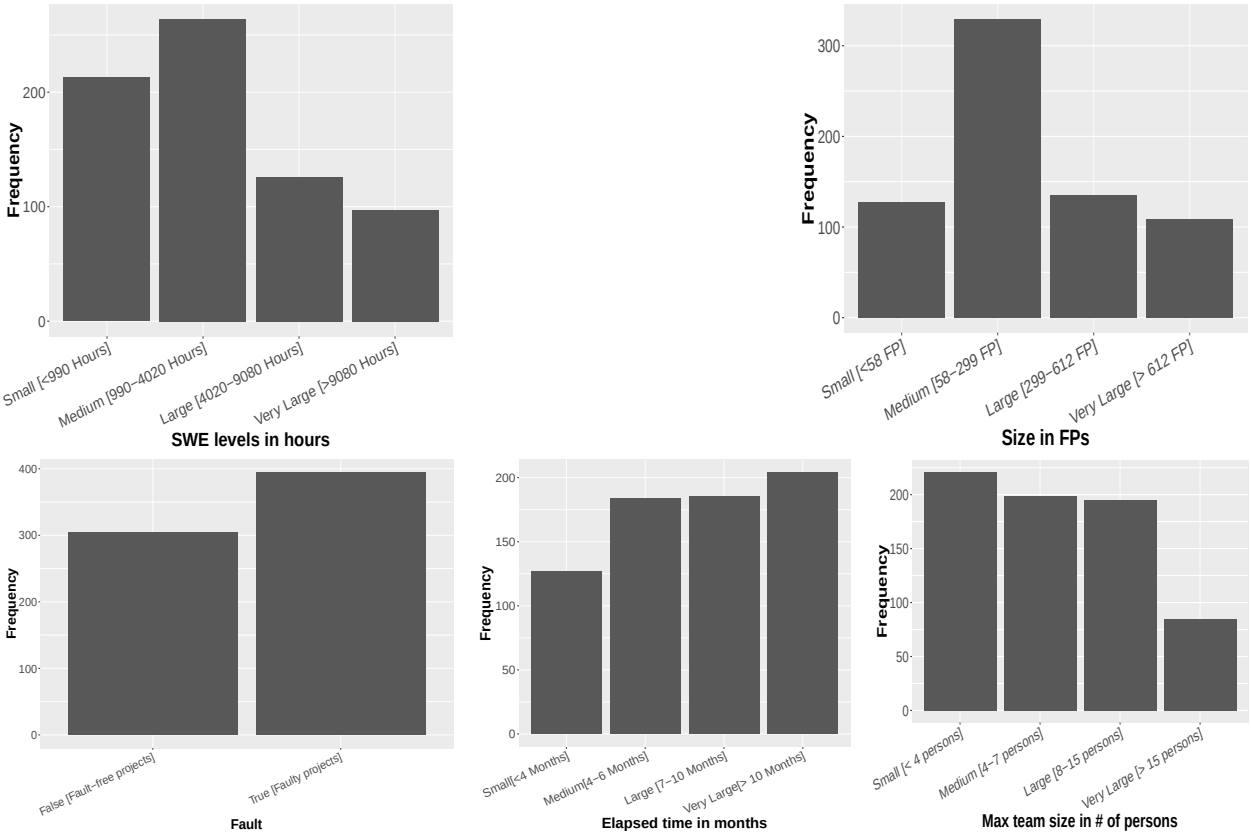
Discretization is the process of converting continuous numerical metrics to ordinal ones. Because we are dealing with an ordinal regression model, we have to discretize the response variable. Discretizing independent metrics is not required to fit into the ordinal regression model. However, we are interested in measuring the contribution of different levels of every single metric and interaction to the response variable.

The discretization method can be as simple as dividing data using equal width or equal frequency. Equal width is based on subtracting the minimum from the maximum value and then dividing the results into the desired number of bins. Equal frequency involves dividing the frequency of data into the desired number of bins o that each bin contains the same frequency. Although these two methods have been widely used, both are very sensitive to outliers [198].

Our method uses ChiMerg, which is a statistical tool for supervised discretization [187]. This tool starts by placing every value at its interval. Then, an χ^2 test is conducted between adjacent intervals. If the two adjacent intervals are not statistically independent, they are merged. This process continues until all adjacent intervals become independent. The distribution of all selected metrics before and after discretization is shown in Figure 5.2. As shown in the figure, all metrics have been discretized into four ordered levels (i.e., small, medium, large, and very large) except the fault metric, which was discretized into two levels (i.e., faulty project, and fault-free project).



(a) Distribution of selected metrics in numerical format



(b) Distribution of selected metrics in categorical format

Figure 5.2: Basic statistics graphs of selected metrics of ISBSG data set

5.3.2 Correlation Test

In an explanatory model, multicollinearity is not desirable and must be treated. If multicollinearity exists, variances of metrics and interactions will be very high. Moreover, it becomes hard to explain if the response variable Y is affected by one any of the highly correlated metrics X_1 or X_2. We treat multicollinearity before and after building the initial model.

Before building the initial model, we test correlation pair-wise between all selected metrics. We test the numerical metrics before discretizing them into categorical using the non-parametric Spearman correlation coefficient. For the categorical metrics (e.g., Language Type) we use contingency correlation coefficient [199]. The second phase tests the multicollinearity of the initial model for all terms of the model (i.e., metrics and interactions). We explain this process after building the initial model in Section 5.3.3.

Spearman Correlation Test for Numerical Confounders

As shown in Table 5.2, there are three cases of high correlations. The first case exists between the functional size and speed of delivery ($r = 0.77$). Second, high correlation is detected between the speed of delivery and manpower delivery rate ($r = 0.84$). Third, high correlation coefficient is also detected between manpower delivery rate and max team size ($r = -0.72$). The response metric SWE has low to medium correlation with all independent metrics.

Table 5.2: Spearman correlation coefficients for numerical metrics

Size						
0.16 ***	Fault					
0.74 ***	0.08**	Speed				
0.48 ***	-0.06	0.70***	Manpower			
0.09***	0.14***	-0.33***	-0.45***	Time		
-0.08***	0.07+	-0.20***	-0.72***	0.42***	Team	
0.41***	0.19***	0.12***	-0.23	0.55***	0.53***	Effort
*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$						

We excluded speed of delivery from the initial model because it is correlated with two other metrics (i.e., size and manpower delivery). Manpower delivery rate is also highly negatively correlated with max team size. This is expected because team size is the denominator of the manpower equation. We have two options: (1) either eliminate the manpower rate or (2) team size. We decide to exclude manpower metric and retain max team size metric because it has higher correlation with the response metric. Moreover, max team size metric was used by many other studies [190, 191].

Level of Association Test for Categorical Metrics

Now, we test the level of association between the two categorical metrics and all the other numerical metrics after their discretization. This method is based on the description provided in [199] and is similar to the method applied in the first case study. We use the contingency table to test the level of association between categorical and ordinal metrics.

Table 5.3: Association levels of nominal and ordinal metrics

First metric	Second metric	χ^2	p value	$C = \sqrt{\frac{\chi^2}{N+\chi^2}}$	$C_{max} = \sqrt[4]{\frac{m-1}{m} \times \frac{n-1}{n}}$	$C^* = \frac{C}{C_{max}}$
Development type	Language type	14	< 0.001	0.15	0.75	0.20
	Size	1000	< 0.001	0.79	0.85	0.93
	Fault	26	< 0.001	0.21	0.75	0.28
	Time	156	< 0.001	0.45	0.85	0.53
	Team	177	< 0.001	0.48	0.84	0.57
Language type	Size	50	< 0.001	0.28	0.79	0.35
	Fault	0.6	0.4	0.03	0.71	0.04
	Time	71	< 0.001	0.33	0.79	0.42
	Team	33	< 0.001	0.23	0.78	0.29

Table 5.3 presents χ^2 values, coefficient C , maximum possible coefficient C_{max}, and p values. χ^2 is calculated based on $(m-1)(n-1)$ degree of freedom. The results indicate that all metrics are highly correlated with the development type. Therefore, we exclude the development type from our initial model. Language type is also highly correlated with all the other metrics except with fault metric. We exclude language type because of its high collinearity with other metrics.

Based on the the final test of association, we have our final selection of independent metrics, which consists of functional size, fault, elapsed time, max team size. Our response variable for this study is SWE. The next step is to build the model by starting with the initial model, which includes all the independent metrics and their interactions.

5.3.3 Building the Model

Ordinal logistic regression is an extension of the standard logistic regression [148]. Logistic regression deals with a response variable with two categories (i.e., true and false). However, ordinal logistic regression deals with more than two levels ordered categories (e.g., low, medium, and high).

We start with the initial model Model_0 shown in Equation 5.1.

$$\begin{aligned} logit[P(\text{SW Effort} \leq k)] = \alpha_k + \sum_{j=1}^{n} \beta_{1j} \cdot \text{Size}_j + \beta_2 \cdot \text{Fault} + \sum_{j=1}^{l} \beta_{3j} \cdot \text{Time}_j + \sum_{j=1}^{m} \beta_{4j} \cdot \text{Team}_j \\ + \sum_{j=1}^{n} \beta_{5j} \cdot (\text{Size} \times \text{Fault})_j + \sum_{j=1}^{n \cdot l} \beta_{6j} \cdot (\text{Size} \times \text{Time})_j + \sum_{j=1}^{n \cdot m} \beta_{7j} \cdot (\text{Size} \times \text{Team})_j \\ + \sum_{j=1}^{l} \beta_{8j} \cdot (\text{Fault} \times \text{Time})_j + \sum_{j=1}^{m} \beta_{9j} \cdot (\text{Fault} \times \text{Team})_j + \sum_{j=1}^{l \cdot m} \beta_{10j} \cdot (\text{Time} \times \text{Team})_j \end{aligned} \quad (5.1)$$

where k is the total number of categories of the response metric;

n is the total number of categories of the size;

l is the total number of categories of the time; and

m is the total number of categories of the team.

Because this model includes all metrics and their interactions, we need to test the multicollinearity that may exists because interactions are correlated with the main effect. We use perturbation, which should test the change in variance of every metric and change in interactions after iterating the model using random values. The model will pass the multi-

collinearity test if the results indicate that the variances of the coefficients of all metrics and interaction are zeros. The initial model in this case is considered to be free of multicollinearity, and we call it the full model Model_f at this stage. The elimination process starts from this model and continues to reach the optimal final model Model_{final}.

We present the ORs and CIs with the final model. Negative coefficients produce ORs between zero and one, and positive coefficients produce ORs higher than one. In ordinal regression, ORs quantify the probability of a higher interval over the probability of lower intervals. Thus, an OR with a value higher than one means that the probability of a higher interval is likely to cause higher effort than the probability of a lower interval. ORs between zero and one indicate that the probability of lower intervals is more likely to cause an effort increase than the probability of higher intervals.

The first term in the model (α_k) represents the intercepts of the model. The number of intercepts depends on the total number of categories of the response metric (SWE). Our response metric is ordered into four levels (i.e., k=1,2,3,4). Therefore, we have three intercepts: $\alpha_{1|2}$, $\alpha_{2|3}$, and $\alpha_{3|4}$ (see Equation 5.2). The first intercept compares the outcomes of top categories (2,3,4) against the first category. The second intercept is when the outcomes compare the top two categories (3,4) against the first two categories.

$$\alpha_k = \begin{array}{lllll} \alpha_{1|2} & P(Y \geq 2) & \text{vs.} & P(Y < 2) \\ \alpha_{2|3} & P(Y \geq 3) & \text{vs.} & P(Y < 3) \\ \alpha_{3|4} & P(Y \geq 4) & \text{vs.} & P(Y < 4) \end{array} \tag{5.2}$$

Independent metrics have dummy variables based on the number of categories. The number of dummy variables required to represent a categorical metric is equal to the number of its categories minus one. Size, time, and team metrics have four levels each, so each needs three dummy variables in the model. The fault metric has one dummy variable because it has two levels. For every interaction, we need the multiplication of the number of dummy variables for the two interacting metrics. For example, the number of dummy variables for the interaction between size and time is nine. This is because $n = 4$ and $l = 4$, so $((n-1) \times (l-1) = 9)$.

Multicollinearity Test for the Initial Model

In this stage, we apply the multicollinearity test (perturbations) on the initial model (Model_0) [188]. The perturbations method is a good alternative for variance inflation factor VIF especially when ordinal regression model is used, and because the model involves interactions [188]. Use of the multicollinearity test at this stage helps to diagnose the inflation caused by any of the metrics or interactions that may cause instability of the model. The instability can be seen by the change in the signs of the coefficients in the model [200]. Additionally, the coefficients can face significant change if a small change occurs to the data [200].

The perturbation method miss classifies categories and reports the coefficients of the model. This process is run multiple times, and all coefficients are recorded every time. Then, the test provides a summary of the coefficients, which includes minimum, maximum, mean, and standard deviation. The results of the mean, minimum, and maximum are identical, for all metrics and interactions, as shown in Table 5.4. This indicates that the coefficients have the same value at every run. Further, the value of the standard deviation is zero with all metrics and interactions, which also indicates that no variation can be found. We conclude that there is no multicollinearity issue associated with the initial model and that we can use this model as a full model (Model_{full}) from which to start the elimination process.

Eliminate Interactions

To eliminate interactions, we need to start with the least significant one based on the t-statistic results. The least absolute t-statistic value is considered the least significant in the model. Because we have multiple levels of interactions that result in multiple terms in the model, we need to ensure that all terms or as many terms as possible are insignificant.

After removing the interaction, we make sure that this removal did not cause a significant change. This is done through a goodness of fit test (i.e., likelihood ratio test) conducted between the two models (i.e., before and after the elimination) [148].

Table 5.4: Multicollinearity test results of ISBSG data set

	Mean	St Deviation	Minimum	Maximum
$\alpha_{1\|2}$	5.38	0.00	5.38	5.38
$\alpha_{2\|3}$	9.17	0.00	9.17	9.17
$\alpha_{3\|4}$	11.49	0.00	11.49	11.49
$Size_{1\|2}$	1.11	0.00	1.11	1.11
$Size_{2\|3}$	2.31	0.00	2.31	2.31
$Size_{3\|4}$	1.74	0.00	1.74	1.74
$Fault$	1.92	0.00	1.92	1.92
$Time_{1\|2}$	3.73	0.00	3.73	3.73
$Time_{2\|3}$	5.34	0.00	5.34	5.34
$Time_{3\|4}$	8.40	0.00	8.40	8.40
$Team_{1\|2}$	2.27	0.00	2.27	2.27
$Team_{2\|3}$	3.30	0.00	3.30	3.30
$Team_{3\|4}$	4.64	0.00	4.64	4.64
$Size_{1\|2} \times Fault$	0.54	0.00	0.54	0.54
$Size_{2\|3} \times Fault$	1.47	0.00	1.47	1.47
$Size_{3\|4} \times Fault$	1.27	0.00	1.27	1.27
$Size_{1\|2} \times Time_{1\|2}$	-0.25	0.00	-0.25	-0.25
$Size_{2\|3} \times Time_{1\|2}$	-1.72	0.00	-1.72	-1.72
$Size_{3\|4} \times Time_{1\|2}$	-0.08	0.00	-0.08	-0.08
$Size_{1\|2} \times Time_{2\|3}$	-0.69	0.00	-0.69	-0.69
$Size_{2\|3} \times Time_{2\|3}$	-2.78	0.00	-2.78	-2.78
$Size_{3\|4} \times Time_{2\|3}$	-2.28	0.00	-2.28	-2.28
$Size_{1\|2} \times Time_{3\|4}$	-2.01	0.00	-2.01	-2.01
$Size_{2\|3} \times Time_{3\|4}$	-3.51	0.00	-3.51	-3.51
$Size_{3\|4} \times Time_{3\|4}$	-2.74	0.00	-2.74	-2.74
$Size_{1\|2} \times Team_{1\|2}$	1.01	0.00	1.01	1.01
$Size_{2\|3} \times Team_{1\|2}$	1.87	0.00	1.87	1.87
$Size_{3\|4} \times Team_{1\|2}$	3.11	0.00	3.11	3.11
$Size_{1\|2} \times Team_{2\|3}$	1.77	0.00	1.77	1.77
$Size_{2\|3} \times Team_{2\|3}$	3.14	0.00	3.14	3.14
$Size_{3\|4} \times Team_{2\|3}$	3.77	0.00	3.77	3.77
$Size_{1\|2} \times Team_{3\|4}$	-0.62	0.00	-0.62	-0.62
$Size_{2\|3} \times Team_{3\|4}$	1.99	0.00	1.99	1.99
$Size_{3\|4} \times Team_{3\|4}$	2.62	0.00	2.62	2.62
$Fault \times Time_{1\|2}$	-2.27	0.00	-2.27	-2.27
$Fault \times Time_{2\|3}$	-2.43	0.00	-2.43	-2.43
$Fault \times Time_{3\|4}$	-1.77	0.00	-1.77	-1.77
$Fault \times Team_{1\|2}$	-0.26	0.00	-0.26	-0.26
$Fault \times Team_{2\|3}$	-0.74	0.00	-0.74	-0.74
$Fault \times Team_{3\|4}$	0.84	0.00	0.84	0.84
$Time_{1\|2} \times Team_{1\|2}$	-1.43	0.00	-1.43	-1.43
$Time_{2\|3} \times Team_{1\|2}$	-1.51	0.00	-1.51	-1.51
$Time_{3\|4} \times Team_{1\|2}$	-2.68	0.00	-2.68	-2.68
$Time_{1\|2} \times Team_{2\|3}$	-1.09	0.00	-1.09	-1.09
$Time_{2\|3} \times Team_{2\|3}$	-1.62	0.00	-1.62	-1.62
$Time_{3\|4} \times Team_{2\|3}$	-2.72	0.00	-2.72	-2.72
$Time_{1\|2} \times Team_{3\|4}$	-0.63	0.00	-0.63	-0.63
$Time_{2\|3} \times Team_{3\|4}$	0.63	0.00	0.63	0.63
$Time_{3\|4} \times Team_{3\|4}$	0.79	0.00	0.79	0.79

The null hypothesis test of the elimination sets all coefficients associated with the interaction to zero (i.e., insignificant), as shown in Table 5.5. The first interaction selected for elimination is between Fault and Team. The following is the statement of the null hypothesis test and the alternative hypothesis of the first eliminated interaction:

H_0: The elimination does not have any effect on the model ($\beta_{91} = \beta_{92} = \beta_{93} = 0$)

H_1: The elimination has a significant effect on the model ($\beta_{91} \neq \beta_{92} \neq \beta_{93} \neq 0$)

The null hypothesis (H_0) states that the reduced model $Model_2$ (after removing all terms of Fault $\times$ Team) is not significantly different from the full model $Model_{full}$. The $\Delta\chi^2$ result with the p value (> 0.05) indicates that the removal of the interaction has no effect on the model, and therefore, the null hypothesis is not rejected. In case the null hypothesis is rejected, the alternative hypothesis is consequently not rejected, which means that the interaction has a significant impact on the model and it should be retained.

The second insignificant interaction in $Model_2$ is Size and Fault. The likelihood ratio test between $Model_2$ and $Model_3$ shows that there is no significant difference between the two. As a result, the interaction is removed from the model. Similarly, the interaction between size and time is the third interaction to be eliminated due to the lack of significant differences between $Model_3$ and $Model_4$.

Next, we explore the Fault and Time interaction, which contains three terms. We set all coefficients to zero ($\beta_{81} = \beta_{82} = \beta_{83} = 0$). The result of this reduction causes a significant impact on the model that leads to rejecting of (i.e., $p < 0.05$) the null hypothesis. Further, we try to eliminate the interaction between size and team, and but this cannot be accomplished because of the significant impact ($p < 0.05$) that occurs when we remove all the terms associated with this interaction. Likewise, the removal of the interaction of time and team is rejected because of the significant change that occurs when the interaction is removed.

Table 5.5: Interaction Elimination Process and Final Model Results

	(Model_{full})	Model_2	Model_3	$\text{Model}_4 = \text{Model}_{final}$
Metrics/Interactions	Coeff.	Coeff.	Coeff.	Coeff.
$\alpha_{1\|2}$	5.38 ***	5.25 ***	5.57 ***	4.48 ***
$\alpha_{2\|3}$	9.17 ***	9.03 ***	9.35 ***	8.24 ***
$\alpha_{3\|4}$	11.49 ***	11.34 ***	11.64 ***	10.49 ***
$Size_{1\|2}$	1.11	1.41	1.31	1.61 +
$Size_{2\|3}$	2.31	2.59+	3.65 *	1.44 *
$Size_{3\|4}$	1.74	1.95	2.93 +	1.59 *
$Fault$	1.92+	1.73+	2.17 **	2.04**
$Time_{1\|2}$	3.73 *	3.68 *	3.76 *	2.75 **
$Time_{2\|3}$	5.34 ***	5.19 ***	5.15 **	3.67 ***
$Time_{3\|4}$	8.40 ***	8.35 ***	8.09 ***	5.80 ***
$Team_{1\|2}$	2.27 +	2.08 +	2.18 +	1.76 +
$Team_{2\|3}$	3.30 *	2.82 *	2.91 *	3.02 **
$Team_{3\|4}$	4.64 *	5.16 *	5.27 *	6.38 ***
$Size_{1\|2} \times Fault$	0.54	0.35		
$Size_{2\|3} \times Fault$	1.47 +	1.27		
$Size_{3\|4} \times Fault$	1.27	1.08		
$Size_{1\|2} \times Time_{1\|2}$	-0.25	-0.36	-0.48	
$Size_{2\|3} \times Time_{1\|2}$	-1.72	-1.81	-2.17	
$Size_{3\|4} \times Time_{1\|2}$	-0.08	-0.09	-0.28	
$Size_{1\|2} \times Time_{2\|3}$	-0.69	-0.74	-0.98	
$Size_{2\|3} \times Time_{2\|3}$	-2.78 +	-2.87 +	-3.13 +	
$Size_{3\|4} \times Time_{2\|3}$	-2.28	-2.28	-2.66	
$Size_{1\|2} \times Time_{3\|4}$	-2.01	-1.99	-1.91	
$Size_{2\|3} \times Time_{3\|4}$	-3.51*	-3.56 *	-3.48 *	
$Size_{3\|4} \times Time_{3\|4}$	-2.74	-2.68	-2.73	
$Size_{1\|2} \times Team_{1\|2}$	1.01	0.91	0.78	0.85
$Size_{2\|3} \times Team_{1\|2}$	1.87	1.73	1.46	1.29
$Size_{3\|4} \times Team_{1\|2}$	3.11 **	2.94 *	2.81 *	2.45*
$Size_{1\|2} \times Team_{2\|3}$	1.77 +	1.61	1.47	1.45
$Size_{2\|3} \times Team_{2\|3}$	3.14 **	2.96 **	2.59 *	2.37 *
$Size_{3\|4} \times Team_{2\|3}$	3.77 **	3.41 **	3.23 **	2.97 **
$Size_{1\|2} \times Team_{3\|4}$	-0.62	-0.46	-0.59	-1.23
$Size_{2\|3} \times Team_{3\|4}$	1.99	1.88	1.44	0.25
$Size_{3\|4} \times Team_{3\|4}$	2.62	3.02+	2.86	1.85
$Fault \times Time_{1\|2}$	-2.27 *	-2.21 *	-2.14 *	-1.81*
$Fault \times Time_{2\|3}$	-2.43 **	-2.27**	-2.01*	-1.95*
$Fault \times Time_{3\|4}$	-1.77 *	-1.79*	-1.51+	-1.39+
$Fault \times Team_{1\|2}$	-0.26			
$Fault \times Team_{2\|3}$	-0.74			
$Fault \times Team_{3\|4}$	0.84			
$Time_{1\|2} \times Team_{1\|2}$	-1.43	-1.29	-1.28	-0.79
$Time_{2\|3} \times Team_{1\|2}$	-1.51	-1.38	-1.28	-1.00
$Time_{3\|4} \times Team_{1\|2}$	-2.68 **	-2.57 *	-2.60*	-2.13 *
$Time_{1\|2} \times Team_{2\|3}$	-1.09	-0.84	-0.81	-0.76
$Time_{2\|3} \times Team_{2\|3}$	-1.62	-1.46	-1.27	-1.53+
$Time_{3\|4} \times Team_{2\|3}$	-2.72 **	-2.57 **	-2.50 **	-2.57 **
$Time_{1\|2} \times Team_{3\|4}$	-0.63	-0.82	-0.68	-0.59
$Time_{2\|3} \times Team_{3\|4}$	0.63	0.37	0.52	0.00
$Time_{3\|4} \times Team_{3\|4}$	0.79	0.55	0.49	0.05
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$				
Test		$\beta_{91} = \beta_{92} = \beta_{93} = 0$	$\beta_{51} = \beta_{52} = \beta_{53} = 0$	$\beta_{91} = ...\beta_{99} = 0$
$\Delta\chi^2$		5.98	5.87	15.1
df	49	46(-3)	43 (-3)	34 (-9)
p value		0.11	0.12	0.08
Result		Do not reject H_0 (Eliminate interaction)	Do not reject H_0 (Eliminate interaction)	Do not reject H_0 (Eliminate interaction)

Eliminate Metrics

Eliminating main metrics is the next and final process in building the final model. First, we need to ensure that a metric is not significant. Second, we need to ensure that the insignificant metric has no significant interaction. If these two conditions are met, then we can try to eliminate the metric. However, we need to ensure that this elimination does not cause a significant change to the original model. The way to do this is to check the ORs and CIs of both models (i.e., the pre- and postelimination models) have not been meaningfully changed according to method by [148].

In this case study, the metrics in the model, achieved in the previous phase (Model_4) are all significant. Moreover, all metrics have at least one significant interaction. Therefore, it is not possible to eliminate any metric from Model_4, and this model is considered to be the final model (Model_{final}), as shown by Equation 5.4.

Goodness of Fit

The last step is to assess whether the model explains data well in a process called the goodness of fit. The goodness of fit test compares the expected outcomes $\hat{Y}$ to the observation Y. The perfect fit is when they are identical, that is, $\hat{Y} - Y = 0$. For ordinal regression models with categorical predictors, we can use the usual likelihood ratio deviance and Pearson χ^2 statistics, which measure the fit of the given model versus the 'saturated' [201]. Unlike the Hosmer test, which is used for logistic regression models, the Pearson test accounts for multi level response categories. We calculate χ^2 based on Equation 5.3.

$$\chi^2 = \sum_{i=1}^{k} \frac{\hat{Y}_i - Y_i}{\hat{Y}_i} \tag{5.3}$$

where k is the number of categories of the outcome, Y_i is the observed values, $\hat{Y}_i$ is the expected values, and degree of freedom is equal to $(rows - 1) \times (columns - 1)$.

The null hypothesis (H0) states that the observations and expected values have no significant differences at $\alpha = 0.05$ (p value > 0.05).

The result of the χ^2 test of our final model indicates that the predicted data fit the observed data. Hence, we cannot reject the null hypothesis (i.e., $P = 0.4 > \alpha = 0.05$) that states that there is no significant difference between the observed and predicted data. Therefore, our final model shown by Equation 5.4 is fit.

$$
\begin{aligned}
logit[P(\text{SW Effort} \leq 3)] = \alpha_k + 1.61 \cdot \text{Size}_{1|2} + 1.44 \cdot \text{Size}_{2|3} + 1.59 \cdot \text{Size}_{3|4} + 2.04 \cdot \text{Fault} \\
+2.75 \cdot \text{Time}_{1|2} + 3.67 \cdot \text{Time}_{2|3} + 5.80 \cdot \text{Time}_{3|4} + 1.76 \cdot \text{Team}_{1|2} + 3.02 \cdot \text{Team}_{2|3} + 6.38 \cdot \text{Team}_{3|4} \\
0.85 \cdot \text{Size}_{1|2} \times \text{Team}_{1|2} + 1.29 \cdot \text{Size}_{2|3} \times \text{Team}_{1|2} + 2.45 \cdot \text{Size}_{3|4} \times \text{Team}_{1|2} \\
1.45 \cdot \text{Size}_{1|2} \times \text{Team}_{2|3} + 2.37 \cdot \text{Size}_{2|3} \times \text{Team}_{2|3} + 2.97 \cdot \text{Size}_{3|4} \times \text{Team}_{2|3} \\
-1.23 \cdot \text{Size}_{1|2} \times \text{Team}_{3|4} + 0.25 \cdot \text{Size}_{2|3} \times \text{Team}_{3|4} + 1.85 \cdot \text{Size}_{3|4} \times \text{Team}_{3|4} \\
-1.81 \cdot \text{Fault} \times \text{Team}_{1|2} - 1.95 \cdot \text{Fault} \times \text{Team}_{2|3} - 1.39 \cdot \text{Fault} \times \text{Team}_{3|4} \\
-0.79 \cdot \text{Time}_{1|2} \times \text{Team}_{1|2} - 1.00 \cdot \text{Time}_{2|3} \times \text{Team}_{1|2} - 2.13 \cdot \text{Time}_{3|4} \times \text{Team}_{1|2} \\
-0.76 \cdot \text{Time}_{1|2} \times \text{Team}_{2|3} - 1.53 \cdot \text{Time}_{2|3} \times \text{Team}_{2|3} - 2.57 \cdot \text{Time}_{3|4} \times \text{Team}_{3|4} \\
-0.59 \cdot \text{Time}_{1|2} \times \text{Team}_{3|4} + 0.05 \cdot \text{Time}_{3|4} \times \text{Team}_{3|4}
\end{aligned}
\tag{5.4}
$$

$$
\alpha_k = \begin{cases} \alpha_{1|2} = 4.48 \\ \alpha_{2|3} = 8.24 \\ \alpha_{3|4} = 10.49 \end{cases} \tag{5.5}
$$

5.3.4 Explanation of the results

In this section, we explain the metrics' ORs and how they contribute to the different levels of efforts. This part is very useful because we eliminated all noise from the data such as multicollinearity and insignificant metrics and interactions. This section explains both the main metrics and interactions that cause an increase or decrease in efforts.

We first discuss the effect of the size of the project in regard to effort. It is obvious that a large project needs more effort than a small project. However, we need to identify when size becomes critical and what level of size really makes a difference. As shown in Figure 5.3a, the effort increases as the functional size increases. The first level of comparison (Level 1|2), compares project size between > 60 and ≤ 60. At this level, results show high ORs (OR=2.45), which means when a project size exceeds sixty functions, the likelihood of increase in effort is almost 2.5 times more than when the project size is less than sixty functions. The second level (level 2|3) compares between project size > 300 and size ≤ 300. At this level, OR is four times higher when the projects exceed three hundred functions compared to projects that have less than three hundred functions. Efforts increase five times when the project size goes beyond 600 functions points compared to projects with size less than six hundred FPs.

Our result concerning the effect of the size of projects in effort is consistent with several other works. For example, in a survey of [185] of the Chinese industry, it was found that the effort in large projects is significantly higher than the effort in small projects. Mendes et al. [202] found a positive coefficient between efforts and logarithmic unadjusted FPs using ISBSG data set. Size confounder using adjusted thousand lines of code was one of the top metrics for different modeling approaches in [83]. In addition, the effort increases as the size (measured in KLOC and FPs) increases in the COCOMO, Desharnais, and Maxwell data sets [76]. The authors in [81] found similar results with ISBSG data. Our work provides more details about the behavior of the metric in the model. It also highlights the different intervals and the change of the coefficient at every level and discusses what increase in effort we may experience at every level. Our real focus is on the effect of the metric on effort, rather than on the accuracy of the prediction.

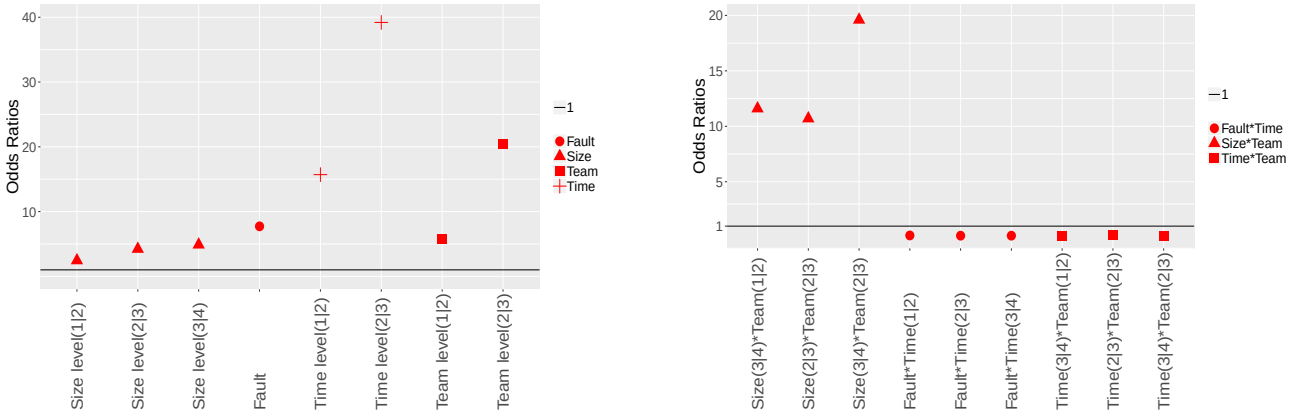
Fault is another new metric in the study of software development effort. Few have considered Fault metric in prediction models. Faults in the ISBSG data set were used earlier as a response variable and not as a predictor, as in [203]. In our study, the Fault metric has two levels (i.e., faulty project or fault-free project). Therefore, the Fault metric in the model is shown in one term. The high OR of the Fault (OR=7.72) indicates that faulty projects are more than seven times more likely to increase in efforts.

The time of the project is significant in all three data sets. In the ISBSG data set, efforts also increase as the time spent on the project (elapsed time) increases, as shown in Figure 5.3b. Level 1|2 compares time of projects ($>$ 4 months) and projects delivered in less than four months. At this point, the OR shows that projects are likely to experience fifteen times larger efforts than projects delivered in less than four months. Then, the higher level (2|3) suggests that projects delivered in more than six months are likely to experience forty times larger efforts than projects delivered in eight months or less.

Other studies found that The time of projects can be a good predictor and can increases efforts. Time is one of the top predictors used with CART, analogy, and CART+OLS models [83]. In addition, duration from the COCOMO81 data set was used as a numerical predictor for the ordinal regression model in [100]. The results of the model showed a positive coefficient ($\beta = 1.53$) that was consistent with our findings. Our metrics, however, are categorical, and our study is explanatory, not predictive.

The effort increases as the size of the team increases. maximum team size is represented with three ORs, as shown in Figure 5.3a. The trend in this case is similar to the time in the sense that the effort of every level is higher than the preceding one. OR of the first level (1|2) indicates that projects with teams of more than four people experience six times more effort than projects that have teams of less than four developers. When level (2|3) is reached, effort increases to twenty times higher than the effort when the team size is less than seven people. At the third level (3|4) of team size, effort enormously increases by almost six hundred times.

Other results are also consistent with our finding. Team Size was one of the top predictors used in modeling approaches by [83]. It was used with OLS, ANOVA, CART, and analogy, and a combination of CART with OLS [83]. Average team size was found to increase productivity in a linear model [191].



(a) ORs for the main metrics for the ISBSG data set

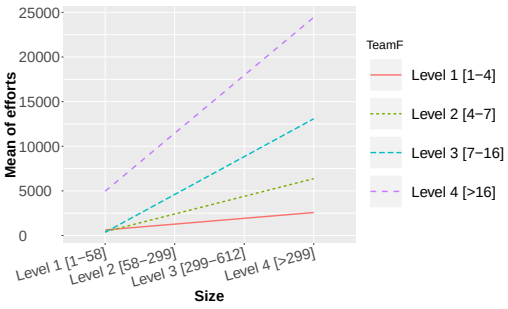
(b) ORs for the interactions for the ISBSG data set

Figure 5.3: Final ORs for the main metrics and interactions of ISBSG

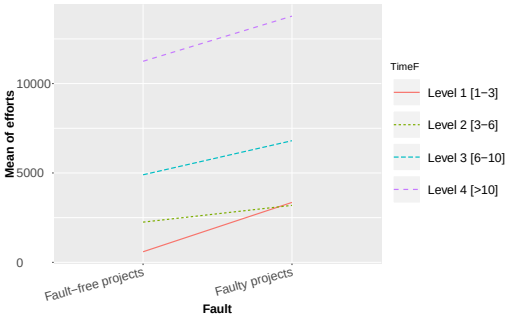
Interaction Results

In this section, we present and discuss the main findings related to the interactions in our model. Interactions explain the relationship of the two metrics when they are combined, and how that combination affect the response variable. We choose two levels of interaction to avoid any complexity in explaining the data. However, three or more levels of interactions can be considered as well. The interaction may behave differently from the behavior of the main confounder. Even though the effect of the main metric increases the effort, interactions decrease the effort except in the size and team interaction. Three dimensions figures of the significant interactions found in Figure 5.3 are plotted in 5.4.

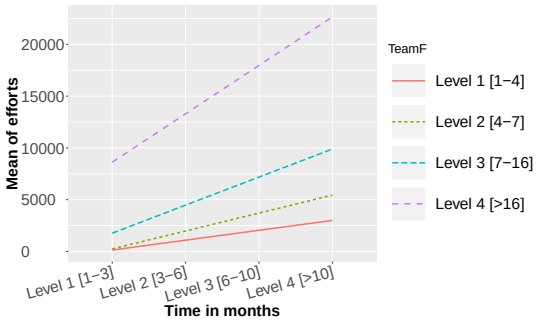
The interactions between size and team are not stable, and only three terms are significant, as shown in Figure 5.3b. It is clear from the figure that size and Team Size lead to an increase in effort. The highest possible effort is shown when the highest level of Size (> 600) is combined with the third level of Team (seven to sixteen people) with OR = 20. When the Size or Team Size is lowered by one level, OR decreases by half (OR=10). We recommended keeping the Team Size at small or medium levels (below 7 people) for large-size projects. The interaction plot between Size and Team Size is shown in Figure 5.4a. The figure shows that small-size teams can keep software efforts at very low levels even with large-size projects.



(a) ISBSG: Size and Team interaction



(b) ISBSG: Time and Fault interaction



(c) ISBSG: Time and Team interaction

Figure 5.4: ISBSG significant interactions

The Fault, when interacting with the Time, decrease efforts, as illustrated in Figure 5.3b. Although Fault as an individual metric, increases Efforts, Fault decreases Efforts when interacting with Time. Faulty projects delivered after a long time involve lower effort than faulty projects delivered after a shorter time. The lowest possible efforts are shown when level one and two of Time interact with Fault (OR = 0.16). Figure 5.4b shows how the two variables interact. This figure shows that faulty projects delivered in less than seven months involve less effort compared to fault-free projects delivered in more than seven months. Similarly, faulty projects delivered in ten months or less involve less effort than fault-free projects delivered in more than ten months. This indicates that the time of projects affects effort more than delivering faulty projects. In this study, we explored Fault and did not investigate the number of faults and type of faults, which can be area for future work.

All significant interactions between time and team have ORs below one, as shown in Figure 5.3b. This indicates that efforts decrease when the two metrics are interacting. In this interaction, low team size is always recommended even for projects delivered after a long period, as shown in Figure 5.4c. The figure indicates that projects delivered in less than three months by a small number of team members (< 7) involve less effort than projects with a large team size (> 16).

5.4 Second Case Study: Desharnais

We repeat the proposed methodology (Section 5.2) on the Desharnais data set. The Desharnais project was developed by a Canadian software house in 1989 [204]. The data set is publicly available, and it contains eighty-one projects. We find only four projects in the data set have missing values, we deleted them, leaving our data set with seventy-seven projects.

The data set has a total of eleven metrics that can be classified based on the experience of developers, duration, size of projects, and the level of language used in the development process. The response variable is effort, which is measured in man-hours. All independent metrics are represented in a continuous format except for language level (see Table 5.6).

Table 5.6: Metrics' definitions in the Desharnais data set [1]

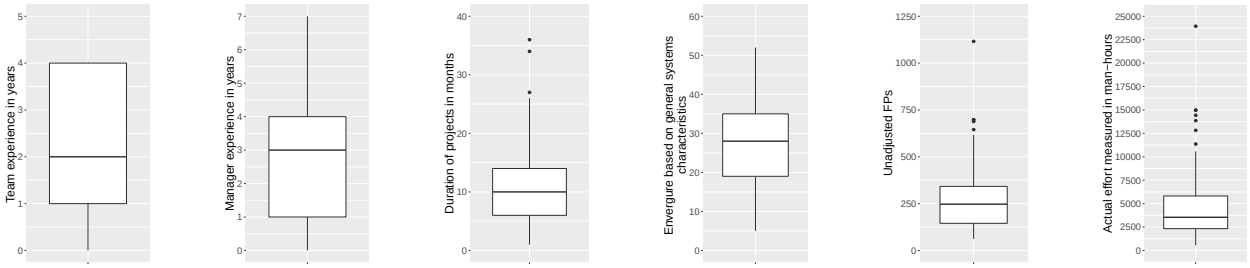
Metric	**Notation**	**Type**	**Definition**
Actual effort	Effort	Numerical	Total number of efforts recorded in hours
Team experience	TeamEx	Numerical	Team experience measured in years
Manager experience	MangExp	Numerical	Manager experience measured in years
Length	Length	Numerical	Duration of the project in months
Envergure	Enverg	Numerical	FP complexity adjustment factor, which is based on the general systems characteristics.
FPs	UFP	Numerical	This is calculated as transactions plus entities
Adjusted FPs	AFP	Numerical	Size of the project measured in adjusted FPs (AFP = UFP$\times$(0.65+ 0.01 $\times$ Envergure))
Language	Language	Categorical	Type of language used in the project: Basic Cobol, Advanced Cobol, and 3/4GL language

We test the correlation between all numerical metrics using the Spearman test. High correlation exists when the coefficient is > 0.7 (the high coefficients). The results of the test are shown in Table 5.7, with gray cells highlighting the high coefficients. Both team experience and manager experience metrics have low correlation between themselves and among other metrics. Length of projects has medium correlation with FPs and actual efforts. Transactions and entities are not strongly correlated with each other, but both of them are strongly correlated with adjusted and unadjusted FPs. The complexity of the project (i.e., envergure) has low to medium correlations with other metrics. Intuitively, the adjusted and unadjusted FPs are strongly correlated with each other and they are highly correlated with both transactions and entities. FPs metric is the summation of the transactions and entities described in Table 5.6. It is clear that we need to exclude some of the independent metrics that are strongly correlated. We need to have FPs in our model because this metric is used in the ISBSG case study. We use unadjusted FPs, which is the same choice we made earlier. As a result, we exclude the adjusted FPs, transactions, and entities from the initial model.

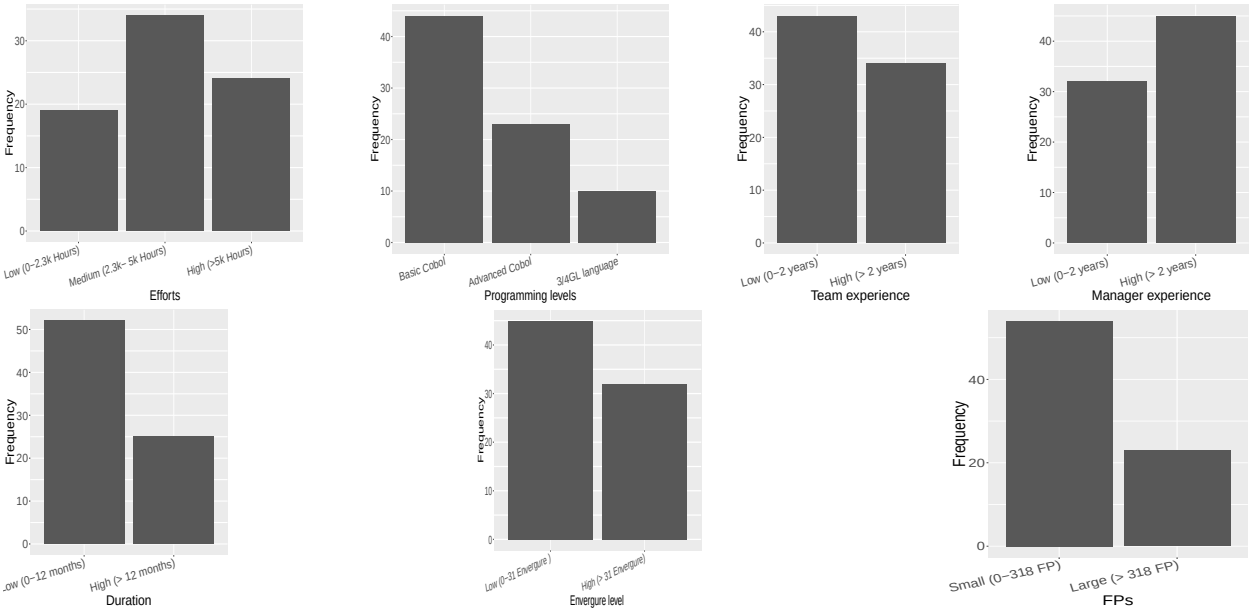
Table 5.7: Spearman correlation coefficients of the numerical metrics of the Desharnais data set

TeamExp								
0.39***	**ManagExp**							
0.36**	0.22*	**Length**						
0.09	0.10	0.38***	**Transactions**					
0.32***	0.17	0.53***	0.26*	**Entities**				
0.30***	-0.09	0.23*	0.45***	0.34***	**Enverg**			
0.26*	0.19	0.59***	0.74***	0.78***	0.47***	**AFP**		
0.30**	0.17	0.58***	0.75***	0.77***	0.58***	0.99***	**FP**	
0.25*	0.08	0.57***	0.47***	0.65***	0.51***	0.69***	0.71***	**Effort**
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$								

Next, we discretize the numerical metrics and convert them into categorical metrics using the ChiMerge method described earlier. Basic statistics of all numerical metrics are illustrated in Figure 5.5. All the numerical independent metrics are discretized into two levels (i.e., low and high) except the response variable (i.e., effort), which was discretized into three levels. The language metric already has three categories and the response variable has three levels of efforts, as shown in Figure 5.5.



(a) Distribution of selected metrics in numerical format



(b) Distribution of selected metrics in categorical format

Figure 5.5: Basic statistics graphs of selected metrics of the Desharnais data set

Now we present the results of the association test using a contingency table shown in Table 5.8. All results have the same degree of freedom because we test the same metric (i.e., language) against all independent metrics, and all of them have the same number of levels. All p values are > 0.05, which means that the null hypothesis that states that there is no association between the two metrics is not rejected. Therefore, the language metric can be in the initial model. We also calculate the coefficient C, maximum possible coefficient C_{max}, and the coefficients in 0-1 scale C^*.

Table 5.8: Association levels of nominal and ordinal metrics

First metric	Second metric	χ^2	df	p value	$C = \sqrt{\frac{\chi^2}{N+\chi^2}}$	$C_{max} = \sqrt[4]{\frac{m-1}{m} \times \frac{n-1}{n}}$	$C^* = \frac{C}{C_{max}}$
Language	TeamExp	1.04	2	0.59	0.36	0.75	0.48
	MangExp	3.01	2	0.22	0.19	0.75	0.25
	Length	2.82	2	0.24	0.19	0.75	0.25
	Enverg	3.56	2	0.16	0.21	0.75	0.28
	FP	1.25	2	0.53	0.12	0.75	0.16

The final selection of the metrics for the Desharnais data set contains team experience, manager experience, length, envergure, FPs, language, and the response variable is effort. The initial model is presented in Equation 5.6.

$$
\begin{aligned}
logit[P(\text{Effort} \leq k)] = \alpha_k + \beta_1 \cdot \text{TeamExp} + \beta_2 \cdot \text{MangExp} + \textstyle\sum_{i=1}^{2} \beta_{3i} \cdot \text{Language}_i \\
+\beta_4 \cdot \text{Length} + \beta_5 \cdot \text{Enverg} + \beta_6 \cdot \text{FP} + \beta_7 \cdot \text{TeamExp} \times \text{MangExp} \\
+ \textstyle\sum_{i=1}^{2} \beta_{8i} \cdot \text{TeamExp} \times \text{Language}_i + \beta_9 \cdot \text{TeamExp} \times \text{Length} + \beta_{10} \cdot \text{TeamExp} \times \text{Enverg} \\
+\beta_{11} \cdot \text{TeamExp} \times \text{FP} + + \textstyle\sum_{i=1}^{2} \beta_{12i} \cdot \text{MangExp} \times \text{Language}_i + \beta_{13} \cdot \text{MangExp} \times \text{Length} \\
+\beta_{14} \cdot \text{MangExp} \times \text{Enverg} + \beta_{15} \cdot \text{MangExp} \times \text{FP} + \textstyle\sum_{i=1}^{2} \beta_{16i} \cdot \text{Language}_i \times \text{Length} \\
+ \textstyle\sum_{i=1}^{2} \beta_{17i} \cdot \text{Language}_i \times \text{Enverg} \textstyle\sum_{i=1}^{2} \beta_{18i} \cdot \text{Language}_i \times \text{FP} + \beta_{19} \cdot \text{Length} \times \text{Enverg} \\
+\beta_{20} \cdot \text{Length} \times \text{FP} + \beta_{21} \cdot \text{Enverg} \times \text{FP}
\end{aligned}
\tag{5.6}
$$

where k = 3 is the total number of categories of the response metric.

The results of the perturbation test did not indicate any multicollinearity issues, as shown in Table 5.9. Therefore, we used this model as the full model of this case study. Coefficients of all terms of the full model are shown in Table 5.10. This table also depicts the results of the likelihood ratio test ($\Delta\chi^2$), p value, and change in the degree freedom between every model and its previous. The change in the degree of freedom indicates how many interactions were removed. We have only two values (i.e., -1, and -2) because interactions have only one or two terms maximum. All metrics in the model have two levels except language, which

has three levels. Therefore, they are represented with one or two terms in the model, as shown in Equation 5.6. The results of the p value (i.e., > 0.05) in Table 5.10 indicate that all the removed interactions do not cause significant effects. The last model in Table 5.10 is Model_{15}, which has retained one interaction: $MangrExp \times FP$.

Table 5.9: Multicollinearity test results of the Desharnais data set

	Mean	Standard Deviation	Minimum	Maximum
$\alpha_{1\|2}$	-0.506	0.00	-0.506	-0.506
$\alpha_{2\|3}$	3.293	0.00	3.293	3.293
$TeamExp$	0.299	0.00	0.299	0.299
$MangrExp$	0.285	0.00	0.285	0.285
$Length$	1.477	0.00	1.477	1.477
$Language_{1\|2}$	0.369	0.00	0.369	0.369
$Language_{2\|3}$	-22.964	0.00	-22.964	-22.964
$Evenger$	3.096	0.00	3.096	3.096
FP	1.393	0.00	1.393	1.393
$TeamExp \times MangrExp$	1.559	0.00	1.559	1.559
$TeamExp \times Length$	-1.506	0.00	-1.506	-1.506
$TeamExp \times Language_{1\|2}$	-4.624	0.00	-4.624	-4.624
$TeamExp \times Language_{2\|3}$	-3.044	0.00	-3.044	-3.044
$TeamExp \times Evenger$	27.08	0.00	27.08	27.08
$TeamExp \times FP$	-23.457	0.00	-23.457	-23.457
$MangrExp \times Length$	-0.695	0.00	-0.695	-0.695
$MangrExp \times Language_{1\|2}$	-0.549	0.00	-0.549	-0.549
$MangrExp \times Language_{2\|3}$	-0.731	0.00	-0.731	-0.731
$MangrExp \times Evenger$	0.49	0.00	0.49	0.49
$MangrExp \times FP$	21.314	0.00	21.314	21.314
$Length \times Language_{1\|2}$	-2.526	0.00	-2.526	-2.526
$Length \times Language_{2\|3}$	22.946	0.00	22.946	22.946
$Length \times Evenger$	3.751	0.00	3.751	3.751
$Length \times FP$	-0.049	0.00	-0.049	-0.049
$Language_{1\|2} \times Evenger$	-1.271	0.00	-1.271	-1.271
$Language_{2\|3} \times Evenger$	-3.096	0.00	-3.096	-3.096
$Language_{1\|2} \times FP$	8.464	0.00	8.464	8.464
$Language_{2\|3} \times FP$	-21.582	0.00	-21.582	-21.582
$Evenger \times FP$	22.332	0.00	22.332	22.332

To reach the final mode, we check how many of the main metrics in Model_{15} are not significant and do not have interactions. Team experience is the only metric that can be removed from Model_{15}. If the removal has no effect on ORs and CIs, this removal is accepted, and the final model is produced. ORs should not significantly change if the metric is removed from the model. Table 5.11 presents the results of the two assessments of ORs and CIs. We

Table 5.10: Interactions Elimination Process of Desharains data set

	$\mathbf{Model}_f$	$\mathbf{Model}_2$	$\mathbf{Model}_3$	$\mathbf{Model}_4$	$\mathbf{Model}_5$	$\mathbf{Model}_6$	$\mathbf{Model}_7$	$\mathbf{Model}_8$	$\mathbf{Model}_9$	$\mathbf{Model}_{10}$	$\mathbf{Model}_{11}$	$\mathbf{Model}_{12}$	$\mathbf{Model}_{13}$	$\mathbf{Model}_{14}$	$\mathbf{Model}_{15}$
$\alpha_{1\|2}$	-0.51	-0.54	-0.72	-0.71	-1.01	-1.00	-1.02	-1.10	-1.03	-1.16	-1.07	-0.88	-0.85	-0.99	-0.83
$\alpha_{2\|3}$	3.29	3.22	2.99	3.01	2.54	2.55	2.53	2.42	2.50	2.32	2.40	2.54	2.61	2.48	2.66 ***
$TeamExp$	0.30	0.22	-1.20	-1.26	-1.18	-1.20	-1.33	-1.12	-1.14	-1.39	-0.73	-0.77	-0.79	-1.01	-0.88
$MangrExp$	0.28	0.18	0.74	0.92	-0.04	-0.05	-0.07	-0.16	0.12	0.12	0.51	0.53	0.46	0.54	0.49
$Length$	1.48	1.70	1.61	1.59	1.19	1.17	0.96	0.22	0.21	0.25	0.13	0.84	1.37	1.28	1.17 +
$Language_{1\|2}$	0.37	0.27	-0.18	-0.15	-0.88	-0.86	-0.85	-0.94	-0.98	-1.01	-1.09	-0.82	-0.83	-0.97	-0.71
$Language_{2\|3}$	-22.96	-22.00	-46.15	-29.45	-7.27	-7.27	-7.06	-6.62	-7.08	-7.65	-8.67	-7.03	-6.58	-5.24	-4.09 ***
$Evenger$	3.10	2.96	2.72	2.78	1.88	1.87	1.84	1.83	2.15	1.74	1.77	1.87	2.20	2.06	2.44 ***
FP	1.39	1.44	1.23	1.23	0.88	0.87	0.87	0.84	0.87	0.67	0.70	0.99	1.13	-0.12	-0.24
$TeamExp \times MangrExp$	1.56	1.02	1.19	1.34	1.57	1.59	1.67	1.37	1.27	1.15	X	X	X	X	X
$TeamExp \times Length$	-1.51	-2.21	-1.00	-0.70	-0.52	-0.46	-1.58	-1.34	-0.91		X	X	X	X	X
$TeamExp \times Language_{1\|2}$	-4.62	-3.40	X	X	X	X	X	X	X	X	X	X	X	X	X
$TeamExp \times Language_{2\|3}$	-3.04	-2.61	X	X	X	X	X	X	X	X	X	X	X	X	X
$TeamExp \times Evenger$	27.08	26.19	-2.32	-1.83	-1.58	-1.56	X	X	X	X	X	X	X	X	X
$TeamExp \times FP$	-23.46	-2.42	-2.11	-3.65	-3.19	-3.27	-3.28	-2.72	-3.04	-3.41	-3.93	-2.41	-1.91	X	X
$MangrExp \times Length$	-0.70	0.58	-0.02	-1.63	-1.40	-1.39	-1.38	X	X	X	X	X	X	X	X
$MangrExp \times Language_{1\|2}$	-0.55	-0.36	-1.17	-1.73	X	X	X	X	X	X	X	X	X	X	X
$MangrExp \times Language_{2\|3}$	-0.73	-1.49	23.01	22.79	X	X	X	X	X	X	X	X	X	X	X
$MangrExp \times Evenger$	0.49	0.62	0.15	-0.39	0.95	0.95	1.03	0.90	X	X	X	X	X	X	X
$MangrExp \times FP$	21.31	2.83	3.01	4.29	4.06	4.29	4.26	3.77	4.03	4.50	5.17	4.09	3.31	2.92	2.96 *
$Length \times Language_{1\|2}$	-2.53	-4.29	-2.99	X	X	X	X	X	X	X	X	X	X	X	X
$Length \times Language_{2\|3}$	22.95	19.14	19.20	X	X	X	X	X	X	X	X	X	X	X	X
$Length \times Evenger$	3.75	3.53	2.81	2.02	2.36	2.33	2.39	2.55	2.22	2.12	1.96	1.42	X	X	X
$Length \times FP$	-0.05	0.81	0.63	2.22	2.11	2.21	2.07	1.56	1.55	1.78	1.95		X	X	X
$Language_{1\|2} \times Evenger$	-1.27	-0.87	0.30	1.45	0.53	0.60	0.59	0.63	0.98	1.11	0.99	0.53	0.68	0.80	X
$Language_{2\|3} \times Evenger$	-3.10	-2.96	22.99	6.19	4.47	4.51	4.33	3.85	4.28	4.80	5.87	4.07	3.45	2.11	X
$Language_{1\|2} \times FP$	8.46	6.21	3.38	0.88	0.31	X	X	X	X	X	X	X	X	X	X
$Language_{2\|3} \times FP$	-21.58	-1.84	-2.13	-1.77	0.25	X	X	X	X	X	X	X	X	X	X
$Evenger \times FP$	22.33	X	X	X	X	X	X	X	X	X	X	X	X	X	X
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$															
$\Delta\chi^2$		3.22	1.937	2.30	4.14	0.01	0.08	0.54	0.35	0.55	0.66	1.24	0.91	0.88	1.31
$P - value$		0.07	0.37	0.31	0.12	0.99	0.77	0.46	0.55	0.45	0.41	0.26	0.34	0.34	0.51
df (No. of removed interactions)		-1	-2	-2	-2	-2	-1	-1	-1	-1	-1	-1	-1	-1	-2

divide the data set into two parts, and we run the model before eliminating the TeamExp metric (i.e., Model_{15}) using the two sets of data and report ORs (i.e., OR1-Before and OR2-

Before, as shown in the table) and CIs (i.e., CI1-Before and CI2-Before, as shown in the table). We compare odds ratios and confidence intervals of the same set of data before and after the elimination of the metric. For example, we compare OR1-Before and OR1-After and OR2-Before and OR2-After.

As shown under the OR assessment part in Table 5.11, we find that after eliminating the metric, ORs either do not change or change with less than 5%. The gray cells show that the ORs after eliminating the metric are either the same or have not significantly changed. We can see that most of the cells of ORs after the elimination are colored in gray, which indicates that no significant change has occurred between the two models. Therefore, removing the metrics does not cause a significant difference.

Table 5.11: ORs and CIs' assessment of the Desharnais model

	OR Assessment		**CI Assessment**	
	Before eliminating TeamExp	**After eliminating TeamExp**	**Before eliminating TeamExp**	**After eliminating TeamExp**
	OR1-Before	OR1-After	CI1-Before	CI1-After
$TeamExp$	0.74	X	1.45	X
$MangrExp$	1.09	1.08	2.47	2.43
$Length$	1.26	1.19	2.58	2.41
$Language_{1\|2}$	0.47	0.48	1.02	1.03
$Language_{2\|3}$	0.04	0.04	0.16	0.15
$Evenger$	6.46	6.20	12.81	12.08
FP	1.27	1.14	4.82	4.21
$MangrExp \times FP$	14.64	14.41	84.56	83.13

In terms of the CIs, we do not want to see significant change as well. At the same time, if new CIs are less than old CIs, this is considered an advantage. The results show more gray cells covering the CI-After, which means the new model causes the new CIs to be exactly the same or smaller in value. Therefore, removing the metric results in a better model. The final model Model_{final} of the Desharnais data set is presented in Equation 5.7 after removing the TeamExp metric.

$$logit[P(\text{Effort} \leq k)] = \alpha_k + 0.21 \cdot \text{MangExp} - 0.65 \cdot \text{Language}_{1|2} - 3.83 \cdot \text{Language}_{2|3} + 1.08 \cdot \text{Length} + 2.38 \cdot \text{Enverg} - 0.63 \cdot \text{FP} + 3.14 \cdot \text{MangExp} \times \text{FP} \tag{5.7}$$

$$\alpha_k = \begin{array}{l} \alpha_{1|2} = -0.64 \\ \alpha_{2|3} = 2.80 \end{array} \tag{5.8}$$

The last part of this section is intended to test the goodness of fit of the final model. The χ^2 test (p value < 0.01) indicates that the predicted classes are a good fit with the actual classes. We discuss the prediction part in Section 5.6 and compare it with other the results of this work and related works.

We explain the metrics' and interactions' ORs and how they contribute to the different levels of efforts. Figure 5.6 presents the ORs of the significant metrics of the Desharnais final model.

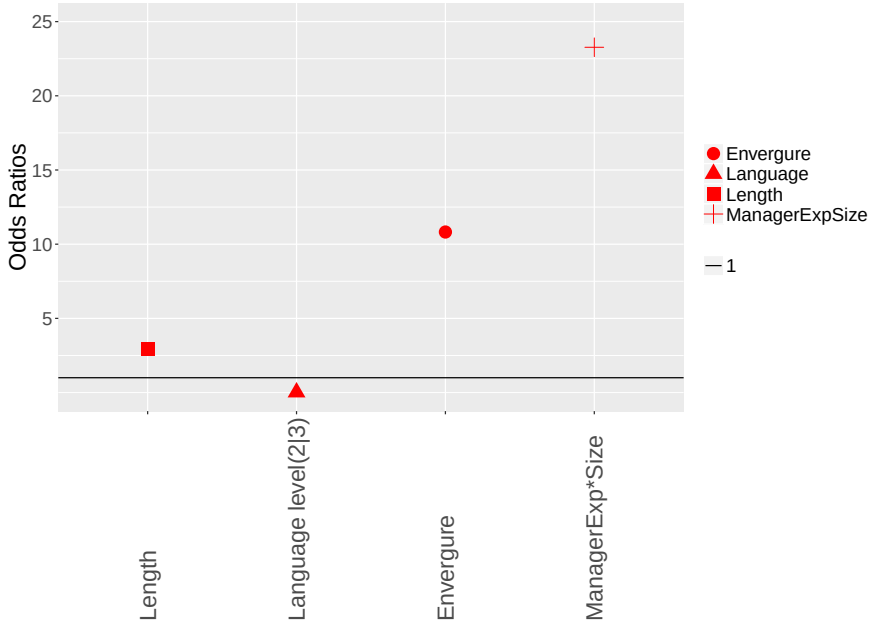


Figure 5.6: Final ORs for the main metrics and interactions of Desharnais

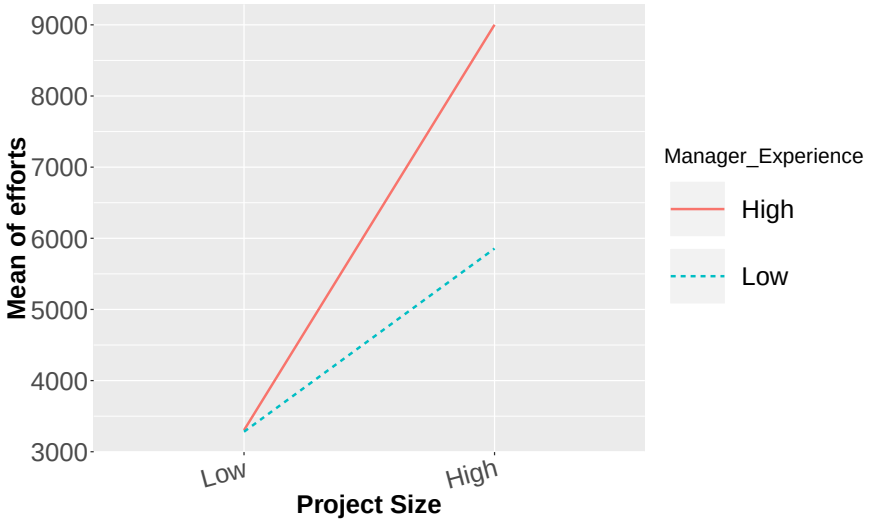


Figure 5.7: Desharnais: Manager experience and project size interaction

The project length is in two levels: Low (0-12 months) and high (more than 12 months). The final odds ratio of the project length indicated that efforts increased three times (OR = 2.94) when the project was delivered in more than twelve months. The result concerning the length of the project in Desharnais was consistent with ISBSG and the results found in [83, 100].

The programming levels in the models are in three levels: Basic Cobol, Advanced Cobol, and 3/4 generation programing languages. The results of the Desharnais model indicated that using 3rd of 4th generations of programing languages reduced the effort with more than 90% (OR = 0.02) when the projects used basic or advanced Cobol. Even using advanced Cobol reduced the effort comparing to using basic Cobol.

The complexity of the size (i.e., envergure) is also divided into two levels: Low (0-31 envergure) and high (more than 31 envergure). The final OR in the Desharnais model showed that when the level of envergure exceeds thirty-one, the probability of effort increases ten more times (OR = 10.80) than when the value is less than thirty-one.

The Desharnais data set has only one significant interaction between manager experience and size (OR = 23), as shown in 5.6. When projects use experienced managers (> 2 years) on large projects (> 318 FPs), the number of efforts becomes significantly high. Figure 5.7 shows that low-size projects with low manager experience involve the lowest possible

efforts. However, with large-size projects, low-manager experience significantly lowers efforts compared to high manager experience. Having manager experience is potentially needed and unavoidable. However, if it is not necessary to recruit highly experienced managers, then software development efforts can be significantly reduced.

5.5 Third Case Study: Maxwell

The Maxwell data set contains sixty-two projects from the largest banks in Finland. It is described using twenty-four metrics, two of which are numerical (i.e., size and duration), and the rest are categorical.

Our selection of the metrics is based on measuring the correlation coefficients and level of association between the categorical metrics. The two numerical metrics are highly correlated based on the Spearman test. We are forced to eliminate one of them, but because we are uncertain about which one should be removed, we decided to make this decision after descretizing the two metrics and measuring their association with other categorical metrics. Based on this result, we decided to eliminate the size metric. This decision was based in the size's strong association with duration and the other two metrics (i.e., App and T03). Furthermore, after eliminating the size metric, we detected no more pair-wise correlation. Our final selection of metrics is defined in Table 5.12.

Table 5.12: Metrics' definitions in the Maxwell data set

Metric	**Type**	**Definition**
Effort	Numerical	Total number of efforts recorded in hours
Duration	Numerical	Duration of projects in months
T03	Categorical	Staff availability
T08	Categorical	Requirements volatility
T10	Categorical	Efficiency requirements
T11	Categorical	Installation requirements

There are five levels for the categorical metrics from T02 to T11 (i.e., development adequacy, and installation requirements) are five: very low, low, nominal, high, and very high. We had events on the two extreme levels (i.e., very low and very high levels). Our model contains metrics and interactions. Using all these levels makes the model very complex to explain and probably impossible to run. Therefore, we eliminated the number of categories

in every level from five to two based on the categories achieved in [100]. We also follow the recommendations of the same study by excluding some metrics that do not have significant effects, such as T02 and T06 (i.e., development adequacy, and tools use). This gives us a smaller and less complex model. The continuous metrics (i.e., effort and duration) were discretized using ChiMerge algorithm into three levels for every metric. The results of the association test using the contingency table of the selected metrics are shown in Table 5.13. All the results show that there is no association between them, which indicates that the number of correlations is very low.

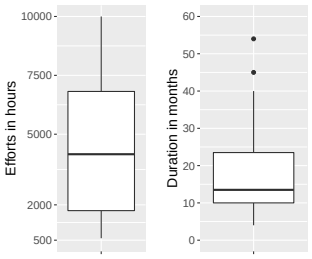
Table 5.13: Association levels of nominal and ordinal metrics

First metric	**Second metric**	χ^2	**df**	**p value**	$C = \sqrt{\frac{\chi^2}{N+\chi^2}}$	$C_{max} = \sqrt[4]{\frac{m-1}{m} \times \frac{n-1}{n}}$	$C^* = \frac{C}{C_{max}}$
T03	T08	10.19	9	0.33	0.38	0.86	0.38
	T10	5.79	9	0.76	0.29	0.86	0.34
	T11	6.86	9	0.65	0.32	0.86	0.37
	Duration	4.79	6	0.57	0.26	0.84	0.31
T08	T10	9.7	9	0.37	0.36	0.86	0.42
	T11	11.22	9	0.26	0.39	0.86	0.45
	Duration	9.32	6	0.15	0.36	0.84	0.43
T10	T11	9.03	9	0.43	0.35	0.86	0.41
	Duration	3.92	6	0.68	0.24	0.84	0.29
T11	Duration	4.86	6	0.56	0.26	0.84	0.31

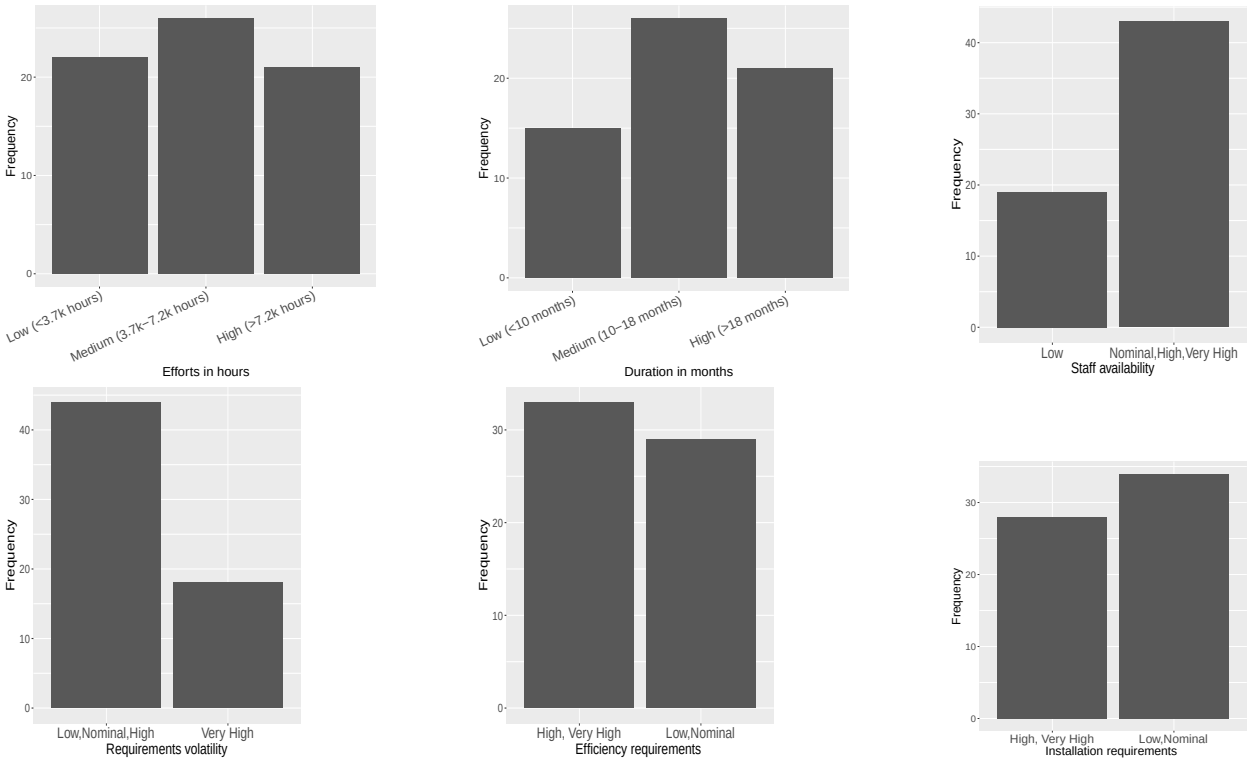
The statistics of our selected metrics are shown in Figure 5.8. Figure 5.8a shows boxplots of the numerical metrics effort and duration. Figure 5.8b presents distribution of the projects of every level discretized metrics.

We start with the selected metrics, and their interactions for the initial model as shown in Equation 5.9.

$$\begin{aligned} logit[P(\text{Effort} \leq k)] = \alpha_k + \beta_1 \cdot \text{T03} + \beta_2 \cdot \text{T08} + \beta_3 \cdot \text{T10} + \beta_4 \cdot \text{T11} + \textstyle\sum_{j=1}^{2} \beta_{5j} \cdot \text{Duration}_j \\ + \beta_6 \cdot \text{T03} \times \text{T08} + \beta_7 \cdot \text{T03} \times \text{T10} + \beta_8 \cdot \text{T03} \times \text{T11} + \textstyle\sum_{j=1}^{2} \beta_{9j} \cdot \text{T03} \times \text{Duration}_j \\ + \beta_{10} \cdot \text{T08} \times \text{T10} + \beta_{11} \cdot \text{T08} \times \text{T11} + \textstyle\sum_{j=1}^{2} \beta_{12j} \cdot \text{T08} \times \text{Duration}_j \\ + \beta_{13} \cdot \text{T10} \times \text{T11} + \textstyle\sum_{j=1}^{2} \beta_{14j} \cdot \text{T10} \times \text{Duration}_j + \textstyle\sum_{j=1}^{2} \beta_{15j} \cdot \text{T11} \times \text{Duration}_j \end{aligned} \tag{5.9}$$



(a) Selected metrics in numerical format in the Maxwell data set



(b) Distribution of selected metrics in categorical format

Figure 5.8: Basic statistics graphs of selected metrics in the Maxwell data set

$$\alpha_k = \begin{array}{lll} \alpha_{1|2} & P(Y \geq 2) \text{ vs. } & P(Y < 2) \\ \alpha_{2|3} & P(Y \geq 3) \text{ vs. } & P(Y < 3) \end{array} \tag{5.10}$$

The perturbation test results (as shown in Table 5.14) did not detect any multicollinearity in the model.

Table 5.14: Multicollinearity test results of Maxwell data set

	Mean	St Deviation	Minimum	Maximum
$\alpha_{1\|2}$	27.87	0.00	27.87	27.87
$\alpha_{2\|3}$	30.69	0.00	30.69	30.69
$T03$	26.11	0.00	26.11	26.11
$T08$	-27.13	0.00	-27.13	-27.13
$T10$	6.7	0.00	6.7	6.7
$T11$	3.24	0.00	3.24	3.24
$Duration_{1\|2}$	25.96	0.00	25.96	25.96
$Duration_{2\|3}$	25.44	0.00	25.44	25.44
$T03 \times T08$	4.32	0.00	4.32	4.32
$T03 \times T10$	-2.47	0.00	-2.47	-2.47
$T03 \times T11$	-2.88	0.00	-2.88	-2.88
$T03 \times Duration_{1\|2}$	-25.15	0.00	-25.15	-25.15
$T03 \times Duration_{2\|3}$	-21.89	0.00	-21.89	-21.89
$T08 \times T10$	-1.12	0.00	-1.12	-1.12
$T08 \times T11$	-0.05	0.00	-0.05	-0.05
$T08 \times Duration_{1\|2}$	27.27	0.00	27.27	27.27
$T08 \times Duration_{2\|3}$	24.84	0.00	24.84	24.84
$T10 \times T11$	-2.79	0.00	-2.79	-2.79
$T10 \times Duration_{1\|2}$	-2.24	0.00	-2.24	-2.24
$T10 \times Duration_{2\|3}$	-1.14	0.00	-1.14	-1.14
$T11 \times Duration_{1\|2}$	2.22	0.00	2.22	2.22
$T11 \times Duration_{2\|3}$	2.84	0.00	2.84	2.84

The elimination process results are shown in Table 5.15. We did not detect any multicollinearity in the initial model, so we used the same terms from the initial model in the full model (Model_f). The process eliminates all interactions, as shown in the table, except the interaction between $T03$ and $T10$. All p values resulting from the likelihood ratio test are > 0.05, which indicates that removing the interaction is safe and does not have any significant impact on the model that we remove the interaction from. The last model in this stage is Model_{10}, which means nine interactions have already been removed from the first model (Model_f).

Next, we eliminated insignificant metrics. The only insignificant metric was $T08$, as shown in Table 5.15. All interactions of the $T08$ metric were removed in the previous process. In this step, we assess ORs and CIs for models with and without the $T08$ metric. The results of OR and CI assessments are presented in Table 5.16.

Table 5.15: Interactions elimination process of the Maxwell data set

	$\mathbf{Model}_f$	$\mathbf{Model}_2$	$\mathbf{Model}_3$	$\mathbf{Model}_4$	$\mathbf{Model}_5$	$\mathbf{Model}_6$	$\mathbf{Model}_7$	$\mathbf{Model}_8$	$\mathbf{Model}_9$	$\mathbf{Model}_{10}$
$\alpha_{1\|2}$	27.87	26.58	5.11	5.04	4.86	4.88	5.45	5.29	4.99	5.31
$\alpha_{2\|3}$	30.69	29.12	7.45	7.37	7.19	7.20	7.74	7.56	7.22	7.46
$T03$	26.11	24.38	2.80 .	2.74 .	2.71 .	2.66 *	2.76 *	2.30 .	1.75 .	2.20 *
$T08$	-27.13	0.55	0.57	0.47	0.40	-0.17	-0.15	-0.32	-0.47	0.79
$T10$	6.70	5.49	3.48	3.25	3.01	2.90	3.81 **	3.37 **	3.38 **	3.41 **
$T11$	3.25	3.82	2.53	2.46	2.13	2.27	2.11	2.69 *	1.77 **	1.80 **
$Duration1\|2$	25.96	24.66	2.33	2.30	2.28	2.47	2.98 *	3.01 ***	3.03 ***	2.90 **
$Duration2\|3$	25.44	24.10	3.63 *	3.61 *	3.51 *	3.72 *	4.32 **	4.78 ***	4.91 ***	4.95 ***
$T03 \times T08$	4.33	1.73	1.49	1.61	1.61	1.78	1.67	1.76	1.94	X
$T03 \times T10$	-2.47	-2.94	-2.90 .	-2.86 .	-2.88 .	-2.87 .	-3.00 .	-2.33 .	-2.39 .	-2.38 .
$T03 \times T11$	-2.88	-2.53	-1.22	-1.18	-1.09	-1.11	-1.00	-1.36	X	X
$T03 \times Duration1\|2$	-25.15	-22.83	X	X	X	X	X	X	X	X
$T03 \times Duration2\|3$	-21.89	-20.37	X	X	X	X	X	X	X	X
$T08 \times T10$	-1.12	-2.61	-1.18	-0.86	-0.82	X	X	X	X	X
$T08 \times T11$	-0.05	2.64	0.67	X	X	X	X	X	X	X
$T08 \times Duration1\|2$	27.27	X	X	X	X	X	X	X	X	X
$T08 \times Duration2\|3$	24.85	X	X	X	X	X	X	X	X	X
$T10 \times T11$	-2.80	-1.85	-0.64	-0.43	X	X	X	X	X	X
$T10 \times Duration1\|2$	-2.25	-1.07	0.73	0.89	0.95	0.85	X	X	X	X
$T10 \times Duration2\|3$	-1.14	0.08	1.17	1.28	1.42	1.29	X	X	X	X
$T11 \times Duration1\|2$	2.22	0.54	0.03	0.08	0.08	-0.18	-0.12	X	X	X
$T11 \times Duration2\|3$	2.85	1.71	1.74	1.74	1.93	1.62	1.79	X	X	X
**** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$, + $p < 0.1$										
$\Delta\chi^2$		4.70	6.35	0.09	0.09	0.26	0.42	1.21	1.05	1.63
$P - value$		0.10	0.06	0.75	0.76	0.61	0.81	0.54	0.30	0.20
df (No. of removed interactions)		-2	-2	-1	-1	-1	0.42	-2	-1	-1

Regarding ORs, we have two metrics and one interaction (highlighted in gray) that have either an exact value or a less than 5% difference from ORs of the pre-elimination model. The other metrics have approximately between 10% and 25% differences in ORs in the reduced model.

In terms of CIs, we detect one interaction with exactly the same value of CI before and after. We also detect another metric with a smaller CI (i.e., better) than the CI of the model before removing the $T08$ metric. The $T03$ CI increases by less than 5% after removing the $T08$ metric. Other metrics, CIs increase between 10% and 25% when we use the reduced metric.

In general, the differences between the two models are not significant according to the OR and CI assessments. We also confirm this with the likelihood ratio test using the $\Delta\chi^2$ of the two models. We find that there is no statistical difference (p value > 0.05) between the two models. Therefore, our final model is shown in Equation 5.11 without the $T08$ metric.

Table 5.16: OR and CI assessment of the Maxwell model

	OR Assessment		**CI Assessment**	
	OR before eliminating TeamExp	**OR after eliminating TeamExp**	**CI before eliminating TeamExp**	**CI after eliminating TeamExp**
$T03$	9.03	9.28	23.07	23.78
$T08$	2.19	X	3.55	X
$T10$	30.32	33.36	100.49	111.91
$T11$	6.06	5.85	9	8.6
$Duration_{1\|2}$	18.12	20.2	40.31	44.89
$Duration_{2\|3}$	141.42	176.8	426	535.53
$T03 \times T10$	0.09	0.09	0.33	0.33

$$logit[P(\text{Effort} \leq k)] = \alpha_k + 2.22 \cdot \text{T03} + 3.51 \cdot \text{T10} + 1.76 \cdot \text{T11} + 3 \cdot \text{Duration}_{1|2} + 5.17 \cdot \text{Duration}_{2|3} - 2.37 \cdot \text{T03} \times \text{T10} \tag{5.11}$$

$$\alpha_k = \begin{array}{l} \alpha_{1|2} = 5.32 \\ \alpha_{2|3} = 7.43 \end{array} \tag{5.12}$$

Finally, we conduct the goodness of fit test, which is the χ^2 test between the predicted and actual classes. The model is fit at p value > 0.01, which means the predicted classes are not significantly different from the actual classes.

Staff availability, efficiency requirement, installation requirements (i.e., T03, T10, and T11), and project duration are all significant in the final model and had odds ratios higher than one, as shown in Figure 5.9. Further, the figure shows one significant interaction existed between staff availability and installation requirements.

Staff availability had odds ratio higher than one (OR = 9.20), which suggested that projects with high staff availability are nine times more likely to experience increase in the effort than projects with low level of staff availability. Similarly, projects with high efficiency requirements are thirty three (OR = 9.20) times more likely to experience increase in the effort than projects with low level of efficiency requirements. High level of installation requirements had higher chance (OR = 5.81) to increase the effort than the project with low level of installation requirements. With respect to the projects duration, projects experience twenty times more effort when delivered in more than ten months compared to projects delivered in less than ten months. Projects delivered in eighteen months are likely to experience 175 times more efforts than projects delivered in less than eighteen months. The result concerning the duration of the project in Maxwell was consistent with ISBSG and Desharnais data sets, and consistent with the results found in [83, 100].

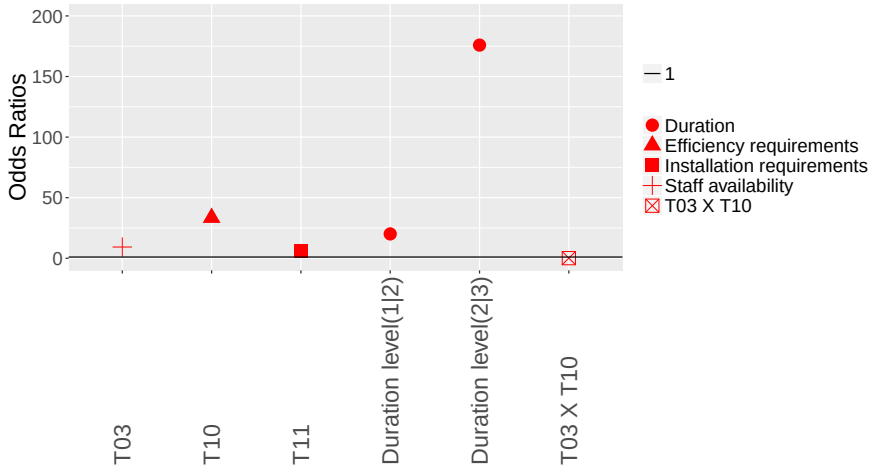


Figure 5.9: ORs for the main metrics and interactions for the Maxwell data set

The Maxwell data set also has one significant interaction: staff availability and efficiency (OR=0.06) requirement, as shown in 5.9. High level of staff availability leads to high effort, and the same is true for efficiency requirements. When they interact, they lead to significantly low effort. Figure 5.10 explains the interaction, which shows that low staff availability with low efficiency requirements has the lowest possible mean of effort. As shown in the figure, the mean of the efforts potentially increases when staff availability is in high.

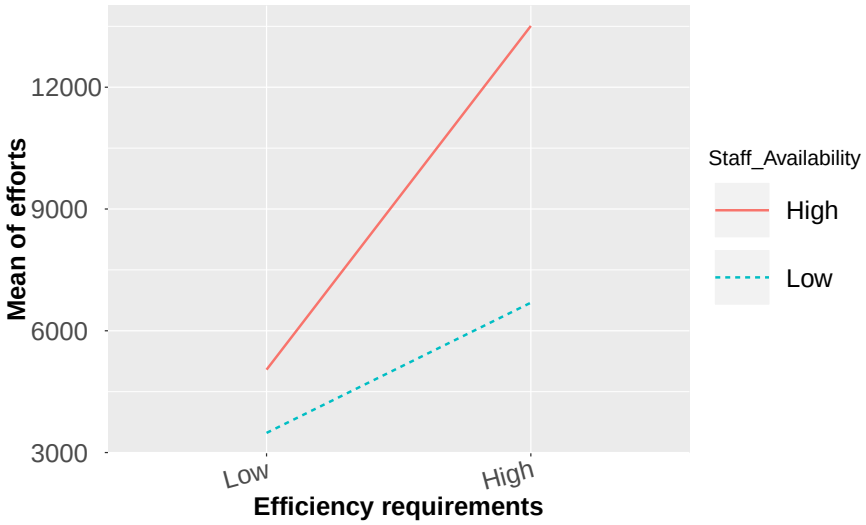


Figure 5.10: Maxwell: Staff availability and efficiency requirements interaction

5.6 Prediction

In this section we conduct a prediction study using the multi-class classification. For performance measure, we use recalls, precision, and F-scores of all the classes of every model. Further, this section discusses performance measures using the mean and median of the magnitude of relative errors MMRE and MdMRE, and PRED(25). Figure 5.11 shows the steps we have taken in this section to calculate all performance metrics (i.e., recall, precision, etc). First, we separate our data set based on the level of the response variables (i.e., four levels for ISBSG and three levels for Desharnais and Maxwell). Then, all the separated

samples are divided into two equal splits. Next, the first 50% fold is used as the training set and the second 50% fold as the testing set. We train the model using the training set and then predict the level of effort using the testing set. All performance metrics are reported, and then the process is iterated one hundred times.

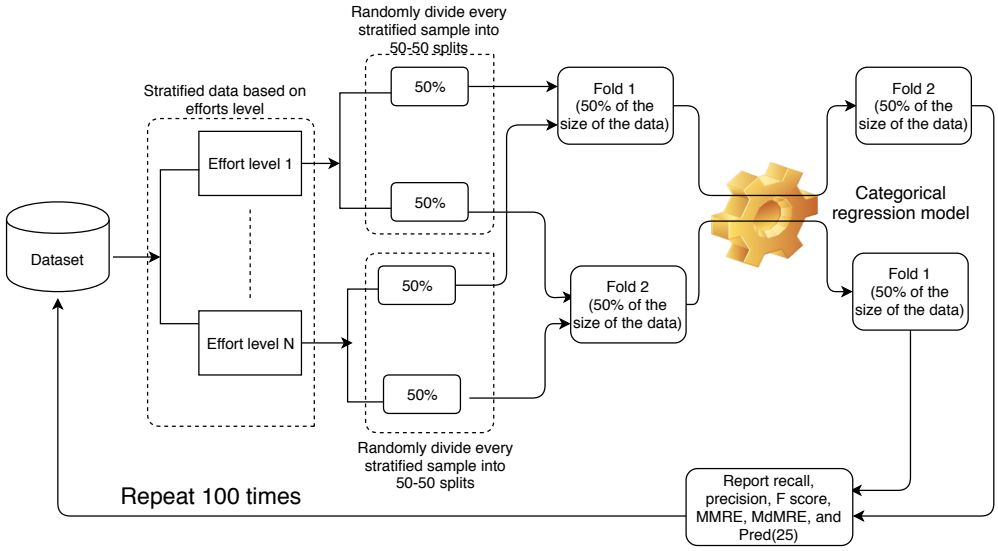


Figure 5.11: Processes for prediction

5.6.1 Multi-Class Classification

We have four levels of efforts in the ISBSG data set, and three levels in the Desharnais and Maxwell data sets. The recall of every class is defined as the percentage of events that are correctly classified as the events that are actually belong to that class, as shown in Equation 5.13. Precision refers to the total number of correct predict class A divided by the total number of events predicted as A, as shown in Equation 5.14. The F-score is the harmonic mean of the recall and precision is calculated as shown in Equation 5.15. The correct classification of all classes is given by Equation 5.16.

$$Recall_A = \frac{\text{Number of correct predictions of class A}}{\text{All actual events of class A}} \tag{5.13}$$

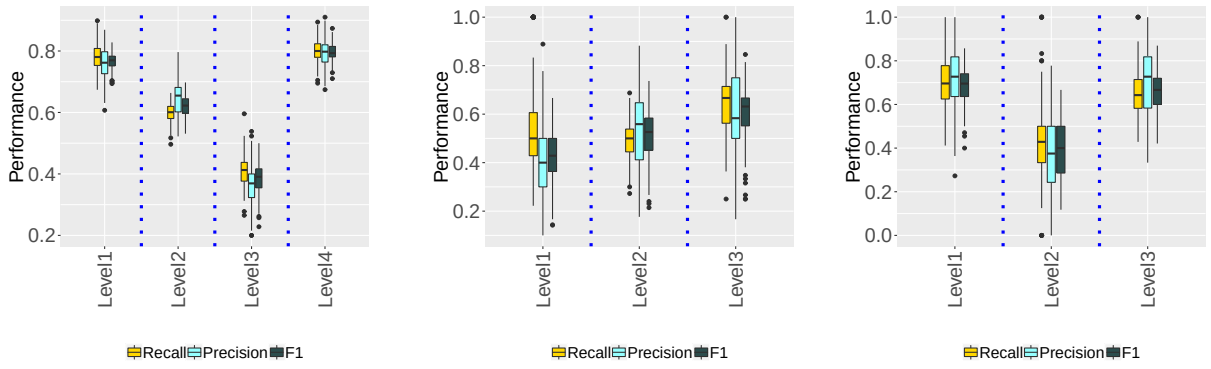
$$Precision_A = \frac{\text{Number of correct predictions of class A}}{\text{All events that were predicted class A}} \tag{5.14}$$

$$F - score = \frac{2 \times Recall_A \times Precision_A}{Recall_A + Precision_A} \tag{5.15}$$

$$Accuracy = \frac{\text{Number of correct predictions of all classes}}{\text{All tested projects}} \tag{5.16}$$

Figure 5.12 presents recalls, precisions, and F-scores of the three data sets used in this study. Every boxplot in this figure explains the results of 200 values that resulted from 100 iterations from the two fold cross validation applied on every data set.

The performance measures the number of events predicted at the right class. In ISBSG, level one and four have the highest recall, precision, and F-score. Level three of ISBSG has the worst performance. Most misclassified events of level two and three go one class higher or one class lower. Similar results are found with the Maxwell data set. Level two of the Maxwell dataset has misclassified events in the upper or lower class (i.e., level one or level three).



(a) ISBSG (b) Desharnais (c) Maxwell

Figure 5.12: Recall precision and F-score of Effort levels of ISBSG, Desharnais, and Maxwell

5.6.2 Performance Metrics

Performance metrics can also be measured by: 1) the magnitude of relative error MRE, which is given in Equation 5.17, 2) the mean magnitude of relative error MMRE is given in Equation 5.18, 3) the median magnitude of relative error MdMRE, which is given in Equation 5.19, and 4) PRED(x), which is the percentage of estimates that fall within x percent of the actual value, as shown in equation 5.20.

$$MRE_i = \frac{|\text{Actual effort}_i - \text{Estimated effort}_i|}{\text{Actual effort}_i} \tag{5.17}$$

For every i observation.

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i \tag{5.18}$$

Where N is the number of observations.

$$MdMRE = 100 \times median(MRE) \tag{5.19}$$

$$Pred(x) = \frac{k}{N} \tag{5.20}$$

Where k is the number of observations and where MRE is less than or equal to x.

As shown in Table 5.17, we compare our results with those of [100] because like us, they used ordinal regression on ISBSG and Maxwell data sets. MMRE was used in [100], MdMRE was used in [205], and PRED(25) was used in both studies. In our study, we report all of them, and we use the mean and median point of estimates as in [100].

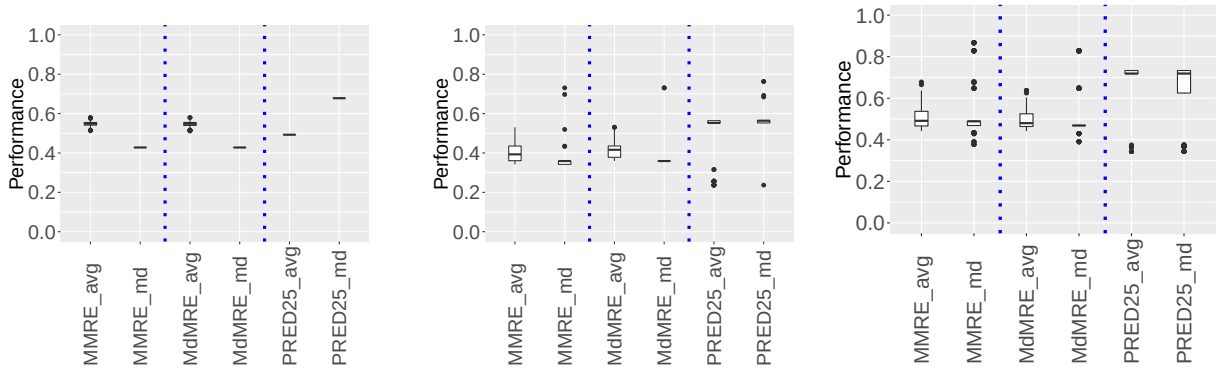
Our MMRE is very close to the MMRE found in ([100]). The PRED(25) results in our study are better than the results found in ([100]). For the Maxwell data set, PRED(25) results are better in [100] than in our study, and our MMRE is better when the mean point of estimate is applied.

Further, we compare our results with the results found in [205], which used sixteen techniques on multiple data sets. Our comparison with [205] shows that our results are either better than the highest performance of all the techniques used in [205] or very close to the best performance. The best MdMRE in [205] is found when OLS+Log technique is used on ISBSG (34.7%). Both MdMREs, using a mean estimate or median estimate, performed slightly better than the [205] model. OLS+Log technique also has the best PRED(25) result, with 36%. In the ISBSG data set, the performance of PRED(25) using a mean or median estimate is much better than the best performance of PRED(25) out of all the techniques in [205]. Additionally, the MdMRE of the mean and median point of estimate of the ISBSG data set is slightly better than the best performance of all the techniques in [205]. Although both the PRED(25) and MdMRE of Desharnais are not better than the best performance of [205] techniques, they are very close to the best performance. MdMRE of the Maxwell data set is very close to the best performance of the techniques in [205]. The PRED(25) results of the Maxwell data set are in the middle of the worst-best range of the performance of the techniques in [205].

Table 5.17: MMRE, MdMRE, and PRED(25) for this work and related works

	Data set	**Our results**			**[100] results**		**[205] results**	
		MMRE	MdMRE	Pre(25)	MMRE	Pre(25)	MdMRE	Pre(25)
Using the mean point	ISBSG	54%	54%	49%	44%	50%	worst-best	worst-best
Using the median point		42%	42%	67%	45%	40%	100-34	19-36
Using the mean point	Desharnais	39%	41%	55%			worst-best	worst-best
Using the median point		35%	35%	56%			47-25	28-49
Using the mean point	Maxwell	49%	48%	72%	59%	37%	worst-best	worst-best
Using the median point		48%	46%	72%	48%	37%	48-32	21-39

Summaries of the answers to research questions **RQ1**, **RQ2**, and **RQ3** are provided in Table 5.18.



(a) ISBSG (b) Desharnais (c) Maxwell

Figure 5.13: MMRE, MdMRE, and PRED(25) for average points and median points of estimate for the three data sets

Table 5.18: Summarized answers to research questions **RQ1**, **RQ2**, and **RQ3**

Research question	Answer
RQ1: Metrics affecting software development efforts	**ISBSG**: Effort increases as the size, time, and team size increase. The presence of faults in projects causes effort increase in ISBSG. **Desharnais**: Efforts increase when time increases and complexity is high. Software development efforts decrease when more 3/4GL programming languages are used. **Maxwell**: Software development efforts increase when staff availability, efficiency requirements, and installation requirements are high.
RQ2: Interactions affecting software development efforts	**ISBSG**: Software development efforts increase with the interaction of project size and team size. Software development efforts decrease with the interaction of Fault and Time and with interaction of time and team size. **Desharnais**: Software development efforts increase with the interaction of management experience and the size of the project. **Maxwell**: Software development efforts decrease with the interaction of staff availability and efficiency requirements.
RQ3: Our models' predictions perform better than others' models	The performance of our models is comparable with other studies and outperforms them in some aspects. We found that MMRE and MdMRE are comparable with the performance reported in [100] and the best performing models reported in [205]. The PRED(25) of our models outperform the PRED(25) reported in [205]. For class performance, we found that performance is high at the highest and the lowest levels. The classification prediction of the middle levels have lower performance. However, the maximum distance of the misclassified instances does not exceed one level.

5.7 Threats to Validity

Construct validity is violated when the study does not measure what it is intended to measure. In this study, we clarified the goals of the research and clearly defined our motivation and what we intended to achieve. The metrics we used in this study are clearly identified to avoid any ambiguousness. In terms of missing values, we used a method that was recommended by many studies and was proven to improve the accuracy of the predictive

models. For data imputation, we applied k-NN method as recommended by recent studies. The imputed data did not exceed 20% of the total number of observations for a single variable. We also tried to minimize metrics that contained missing values and minimized the percentage of missing values inside a single metric. We conducted a correlation test between all numerical metrics. We eliminated two metrics due to the results of the test (i.e., speed of delivery and manpower delivery). Additionally, we conducted correlation tests of the categorical metrics (i.e., development type and language type). Based on the χ^2 test, both metrics were also excluded from the study. In regard data discretization, we applied a statistical method (i.e., ChiMerge) to ensured that every created level was significantly independent. We applied multicollinearity test perturbation to the initial model to ensure that our model was free from any inflation of variances.

Internal threats mainly concern the quality of data. We used the highest class of integrity A and B based on the ISBSG classification. We ignored projects classed C and D because they were considered weak in integrity.

Conclusion validity is the degree to which results based on the data are reasonable. In this study, we used ordinal regression, which was fit for our data types. Although we applied explanatory work, our results showed consistency with other predictive studies. In addition, the model was fit according to the Pearson χ^2 test and the performance metric showed accuracy comparable to other studies.

External validity deals with the generalizibility issue. The projects used in this study were from ISBSG, Desharnais, and Maxwell. The results presented in this study were not necessarily applicable to other data sets. However, by replicating the work of the ISBSG on Desharnais and Maxwell, we found some consistent results that may indicate the generalizibility of these aspects. For example, the length of the project gave consistent results in ISBSG, Desharnais, and Maxwell data sets. Using different metrics for every project makes it difficult to explore the generalizibility further.

5.8 Conclusion

In this paper, we presented a systematic approach for modeling software effort and used the model to explain the effect of different metrics and interactions on the amount of software development effort. Ordinal regression helped us to predict and explain multiple intervals that were caused by changes of different levels of metrics and interactions. The level of prediction was comparable with other studies, which made this model suitable for prediction. Further, we used interactions to determine how to improve prediction. In this study, we found that interactions can explain beyond what main metrics cannot. The answers to the research questions and summary of the main findings are presented in Table 5.18.

The study used data from the International Software Benchmarking Standard Group (ISBSG) release April 2016. Our sample had between 10% missing values for both Faults and Team Size metrics. We applied the k-NN algorithm to impute missing values and create our sample.The process started by analyzing existing metrics and select the potential metrics based on the related works and correlation tests. Some of metrics were eliminated due to high association like Speed of Delivery and Manpower Rate. All numerical metrics were transformed to ordinal data using ChiMerge tool. Then, a χ^2 test was conducted of contingency tables created between categorical and ordinal data to test for association. Based on this test other categorical metrics were eliminated. After preparing data and creating our sample, we built the initial model including metrics that were preselected and interactions of those metrics. Multicollinearity test was conducted using perturbation tool to ensure no multicollinearity existed. Then, we eliminated insignificant interactions and metrics.

Future research can explore additional metrics that exist in the ISBSG data set such as the counting approach, organization, methodology, and programming language. Additional research also needs to be done on data imputation, especially when many measurements have many missing values. Previous works compared some imputation methods but used small data sets. Another potential approach is to try oversampling techniques with the classes of the response variables to gain balanced distribution and to increase the training for the minor classes. Last, we need to replicate this work with other data sets to explore the generalizability of the main empirical observations.

Chapter 6

The Study of Causality

This chapter covers the topic of causality in the area of software fault proneness. The proposed method is applied on Eclipse's Europa release data set. First, this chapter starts with the introduction to the importance of the causal research and the limitation of the current causal studies in Section 6.1. Second, this chapter provides a full explanation of the proposed methodology in in Section 6.2. The methodology is derived from the structural equation modeling SEM technique, which includes data analysis, variable selection and factor analysis, model specification, model estimation, and model validation. The implementation of the methodology and the results based on Eclipse's Europa release are explained in Section 6.3. The threats to validity are discussed in Section 6.4, and the chapter is concluded in Section 6.5.

6.1 Motivation and Background

In the area of software engineering, causality has not yet been widely researched [124], with an exception of a few works that have attempted to implement causal studies using Bayesian networks (BN) [124, 125, 126]. However, some of these studies used BN mainly for prediction and decision-making (e.g., [127, 43, 128, 129, 124, 126]). Additionally, these works incorporated a limited number of static code variables and ignored other types of variables (e.g., change variables).

Earlier explanatory works [3, 4, 5, 6, 7, 8, 9] used logistic regression to explain the dependent variable and measure the contribution of independent variables based on adding one-variable-at-a-time approach. We proposed the use of a case-control method in Chapter 3, which involved confounder selection, multicollinearity, exploring the effect of interactions, and eliminating based on backward hierarchical method. Further, we used categorical regression to explain multi-level confounders and their interactions in Chapter 5. These method deal with a single dependent variable and multiple independent variables. The assumption of these methods is that all independent variables cause the dependent variables with different levels of impact. In other words, the possibilities of the existence of direct and indirect effect and the weight of every effect are not determined. We introduced the interactions of confounders in our models in Chapters 3 and 5, which had a sense of an indirect effect when the estimation of the confounders interaction is different than the estimation of the confounder by its own. However, introducing a higher level of interactions (i.e., three confounders or more) would make the model more complex. It becomes even harder when confounders are in categorical formal as we did in Chapter 5. All these attempts from related works and this study were not focused on causality.

Therefore, there is a need to establish a causal modeling that can support multiple response variables, multiple paths (direct and indirect), and organize and reduce the number of independent variables. We may need multiple response variables instead of dealing with one response variable in case an indirect path exists. For example, if A causes C through B, both B and C are treated as response variables. Thus, the causal model may encounter multiple paths that lead to a response variable. Dealing with many independent variables

may not be needed as some of them may not have a significant effect. Moreover, instead of building a network with many variables, we may reduce that by grouping variables into groups (i.e., latent variables) and measuring the effect of these groups instead of measuring the effect of every individual variable, which leads to a complex model.

Fenton et al. [124] advocated using BN in software engineering and specifically in finding the causal effect on software defects. A Bayesian network algorithm is very useful to estimate the probabilities of multiple outcomes. It can also investigate the effect of different processes at different times throughout the life cycle of the software. Several studies [127, 43, 128] have applied BN to predict and explain causal effects on software faults. BN can be very complicated when there are too many variables. It gets more complicated when they all explain the same process or are extracted at the same time. Therefore, detecting causal relationships might not be very accurate, which leads to many relationships that are hard to explain or predict. Although the BN are very helpful for prediction, learning of the structure of the network may not be accurate unless we rely on our previous knowledge of the development process and how metrics are supposed to be interrelated. One study [43] used BN to learn the structure and causal direction for object-oriented metrics using several data sets. This study did not test the correlation among variables or the multicollinearity of the models, which could introduce threats to the validity of the conclusion and the results. Another problem of this work is that it did not include change metrics, which are essential part as we discussed earlier and as found by related works [21]. Another limitation of BN is that they cannot statistically validate that a group of variables belongs to the same factor [132]. In other words, BN cannot demonstrate (by creating or validating) the existence of latent variables. A latent variable is important to reduce minimize independent variables, especially when all variables belong to the same milestone (e.g., requirements, design).

In this dissertation, we propose an approach for studying causality based on SEM. This method (i.e., SEM) involves statistical analysis and regression analysis between a small and a large set of independent and dependent variables [206]. The first SEM was built by Sewell Wright, who attempted to prove the genetic influence over generations in the 1920s [10]. Social scientists started to use this method in the 1960s [10]. The fields of psychology, marketing, management, accounting, and business have adopted and used the

method in recent years [11, 108, 109, 110, 111, 115, 116, 118, 120]. SEM integrates multiple techniques to obtain the final group of causal diagrams and models. The relationship can be hypothesized and constructed based on earlier knowledge, or it can be statistically analyzed and constructed based on the data. The approach we explain in this chapter consists of multiple analysis and statistical steps to construct the final models and diagrams and to explain the effects and direct and indirect effects of all variables on the response variable.

Two types of variables exist in SEM: observed variables, which can be independent variables (IV) or dependent variables (DV), and latent variables (LV). A latent variable is an unmeasured variable that usually connects more than one correlated independent variable. A latent variable is identified by a group of variables that have the same pattern (i.e, variance). For example, a latent variable can be a phase of software development or a group of variables that measure the same type of objects or define something common (e.g., requirements phase metrics, static code or change metrics). Latent variables are very helpful to reduce the number of variables and the number of a complex relationship in case a large number of independent variables exist. Latent variables are connected with independent variables through two types of relationship: (1) reflective relationship, which means independent variables are caused by the latent variable and the direction of the arrow pointing at the independent variables, (2) formative relationship, which means independent variables causing the latent variable and the direction of arrows pointing at the latent variable. A latent variable can be determined by analyzing the correlation of all independent variables, using statistical tools like principle component Analysis (PCA) or exploratory factor analysis (EFA), and using the knowledge about the data.

SEM causal inference combines between the first and second generation of statistical analysis [207]. Causal inference of the third generation has three essential assumptions: (1) change in an independent variable leads to the dependent variable, (2) the study accounted for the potential variables that may be considered as a spurious relationship, (3) the cause occurs before the effect (i.e., temporal precedence) [207]. SEM can deal with a large number of variables because it estimates separate sets of variables and constructs underlying factors (i.e., latent variables) for every set based on a factor analysis approach [208]. Further, the method involves measurement models which estimate the loading between independent

variables and latent variables. Relationships between latent variables are estimated through regression analysis. SEM allows for multiple outcomes; it can involve multiple phases and direct and indirect effect, similarly as BN. Additionally, SEM can handle a large set of variables because it involves the latent variables [208]. So, instead of dealing with a too complicated network as BN would do, only a few relationships of a few latent variables are estimated. SEM is used to explain the relationship between variables and latent variables and between latent variables themselves. Incorporating BN with SEM is a possible approach when the prediction is desired or to determine the causal direction between two or more variables.

There are two main approaches to implement SEM: covariance-based SEM (CB-SEM) and partial least square path modeling (PLS-SEM). Both approaches yield the same results using good measures [209]. The CB-SEM is restricted to the normal distribution of the data and using a large size sample. Maximum likelihood (ML) is the estimation method used with CB-SEM. PLS-SEM is also known as the variance based or soft modeling approach, which aims to maximize the explained variance of the latent variables. The major difference between the two approaches is that CB-SEM is confirmatory and PLS-SEM is exploratory approach [209]. The PLS-SEM can handle larger size and deal with a complex model (i.e., more independent variables, more latent variables, and more paths) [209]. Further, PLS-SEM handles the variables with nonnormal distribution (i.e., skewed data). The decision of which approach to use in this chapter was based on the fact that we wanted to use exploration study and used as a case study.

Both BN and SEM methods can be used to explore causality in software engineering. We decided to use the SEM approach because (1) the number of variables is large, (2) it is not very clear which metrics go together or whether they measure the same thing, (3) variables are not linked to specific milestones of software development, and (4) data were extracted at a single point. Our case study is based on data extracted from the Eclipse project [38, 2]. The variables include two types: static code metrics extracted from source files of the project, and change metrics extracted from the change history, which explain the changes made to source code files. Though we have two main factors from these data, static code and change variables, we do not have enough information about the detailed

processes for this data. For example, we do not know whether the developer adds lines to fix bugs or for Refactoring purposes. Further, the data were extracted at a single point of the software development. Therefore, we cannot construct the causal network based on the previous knowledge. To solve that we need to involve a statistical analysis that can deal with high dimensional analysis and minimize that to smaller number of dimensions.

6.2 Methodology

In this section, we summarize the main steps to establish an SEM model, which includes sampling, data analysis, variable selection, followed by the model specification, estimation, and validation, as shown in Figure 6.1. In this section, we briefly explain the steps of the methodology. More details are presented in the following section, which discusses the application of this methodology on the case study using Europa release of the Eclipse project.

First, data analysis is conducted to analyze the distribution of variables and measure the skewness and kurtosis of the data. This helped us to decide what modeling technique we should follow (i.e., CB-SEM or PLS-SEM). Additionally, we need to remove any redundant variables that are highly correlated and they have the same definitions (e.g., Maximum Complexity and Average Complexity). At this point, the initial selection of variables should have been achieved. Knowing the number of variables helped us to determine the sample size.

Second, the model specification determines the number of latent variables and independent variables connected to them. This step consists of two sub-steps: creating the measurement models and structural models. The measurement models involve assigning independent and dependent variables to latent variables and determine the type of relationship (i.e., reflective or formative). Structural models involve assigning the path direction between latent variables.

Third, the type of estimation method is determined after analyzing the data and after the model specification. Besides the distribution of the data, other factors are taken into account such as the complexity of the paths in structural models, the number of latent variables, and type of relationship between latent variables and variables. CB-SEM estimate the loading

factors of the measurement models and estimate path coefficients between latent variables using maximum likelihood ML or other similar methods. PLS-SEM also estimates the loading factors and weights of measurement models (outer models) and use the ordinary least squares (OLS) with bootstrapping samples to estimate path coefficients, which determine the direct and indirect effects.

Forth, model validation is conducted to measure the goodness of fit and how good is the model in explaining the data. Both CB-SEM and PLS-SEM use different methods to validate the model. CB-SEM uses measures like the comparative fit index (CFI), the Tucker-Lewis index (TLI), and the root means square error of approximation (RMSEA). PLS-SEM uses other methods of model validation such as (1) examining the multicollinearity in the measurement and the structural models using the variance inflation factor (VIF), (2) measure the average variance extracted for latent variables with reflective relations, (3) measure the R^2 for the endogenous latent variables, and (4) measure the level of significance for the path coefficients based on the t-statistic.

6.3 Case Study Eclipse's Europa Release

Eclipse is an open source integrated development tool IDE that was built using Java programming language. The project is used for developing programs that are written in Java and other programming languages (e.g., C, C++, and C#). Europa is one of the Eclipse releases, which was released in 2007. Europa projects contains a total of 30,862 files. We use a total of thirty-two static code variables and change variables extracted by prior works [38, 2][1].

Our aim in this chapter is to use the variables to build the SEM model and find indirect and direct causes that lead to software fault proneness. To achieve that, we need to follow several steps as described in Section 6.2. The processes described leads to a creation of a causal model that has several latent variables connecting independent variables and have structural paths that lead to the response variable. In this section, we address the following research questions:

[1]More detailed description of the data can be found in Chapter 3

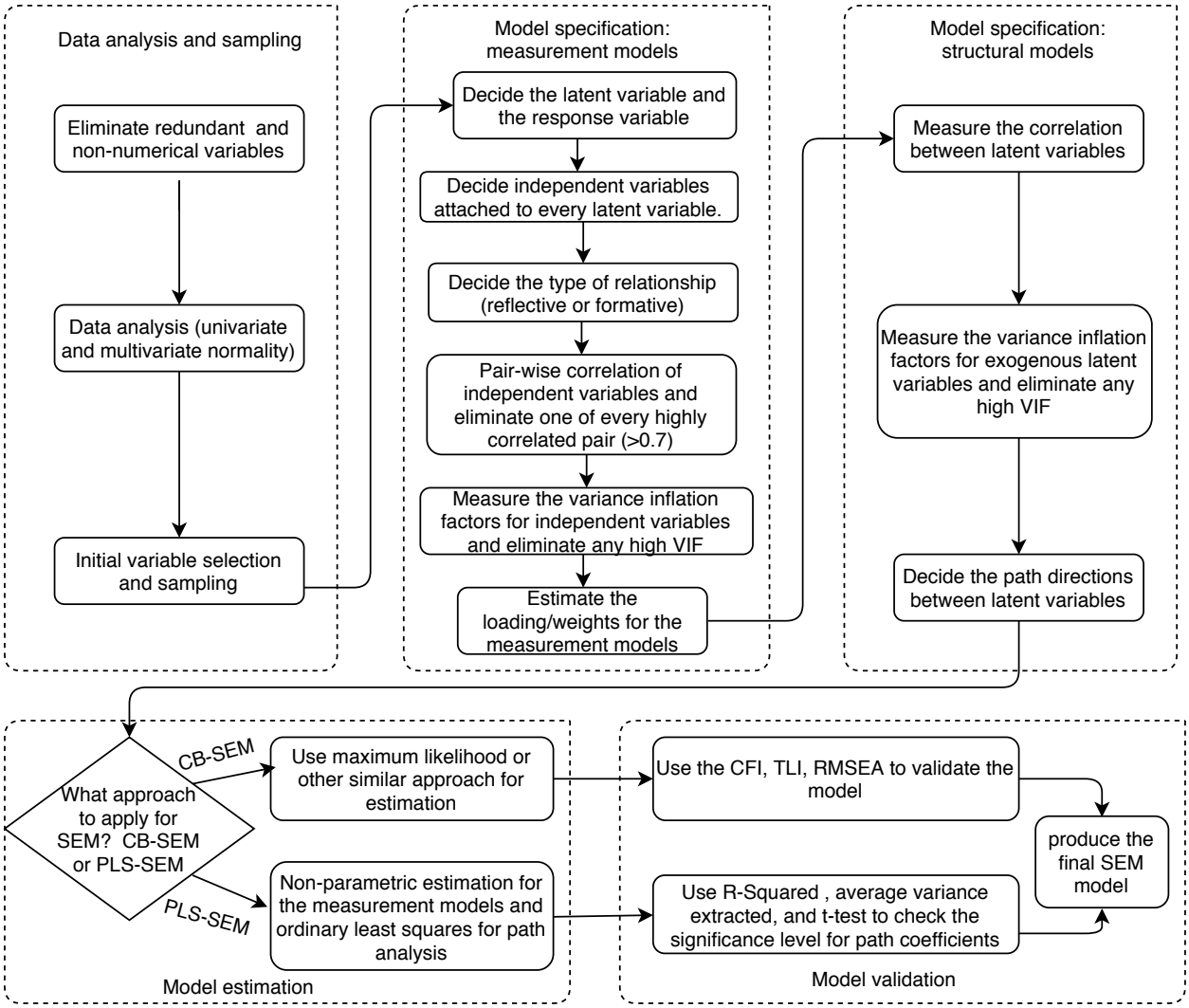


Figure 6.1: Methodology of the causality research

RQ1: How many underlying factors (i.e., latent variables) can be found in the whole set of static code and change metrics?

RQ2: What metrics have significant loading factors (β) linked to every latent variable?

RQ3: What latent variables have the highest direct impact on software fault proneness?

RQ4: Does any latent variable indirectly affect software fault proneness, and what is its impact?

Initial selection of variables

To reach the total number of variables, we need to analyze variables and keep the useful ones. At this stage, we remove non-numerical variables (e.g., file, package, component name). Further, we remove some of the redundant variables. For example, both Average Complexity and Maximum Complexity explain the complexity of a software unit. So, we keep one of them and eliminate the other one. These variables are highly correlated, and they may affect the estimation of the model. As shown in Figure 6.2, the Spearman test wass applied to detect pairs of variables that are highly correlated and one of them was eliminated. Every dark red cell indicates a high correlation between the crossed pair of variables. As we can see, maximum complexity is highly correlated with Average Complexity, Maximum Depth is highly correlated with Average Depth, and Maximum Changeset is highly correlated with the Average Changeset. Therefore, we kept the average of all variables and we eliminated their maximum. Also, maximum and average of LOC added, LOC deleted were deleted because they are highly correlated with each other and all of them are described by the code churn[2]. The total number of the remaining independent variables is eighteen. Definitions of the initial variables (after removing the non-numerical and redundant variables) are presented in Table 6.1.

Assessing non-normality

A normal distribution is used as an assumption for many estimation methods. A high level of non-normality has a great impact upon results of SEM [210, 211]. In the real world, a normal distribution is not always possible. However, with a large sample, estimation can handle non-normal distribution variables at certain levels. Therefore, we need to assess the distribution of variables and the severity of the skewness of the distribution. This can be done using three types of distribution tests: univariate skew, univariate kurtosis, and multivariate kurtosis. A large skewness indicates that more responses occurred at the same point on the axis. Kurtosis measures the level of flatness of the distribution [212]. It is important to note that no agreement has been reached regarding the acceptable level of non-normality for

[2]Code Churn is the summation of the LOC added with the LOC deleted

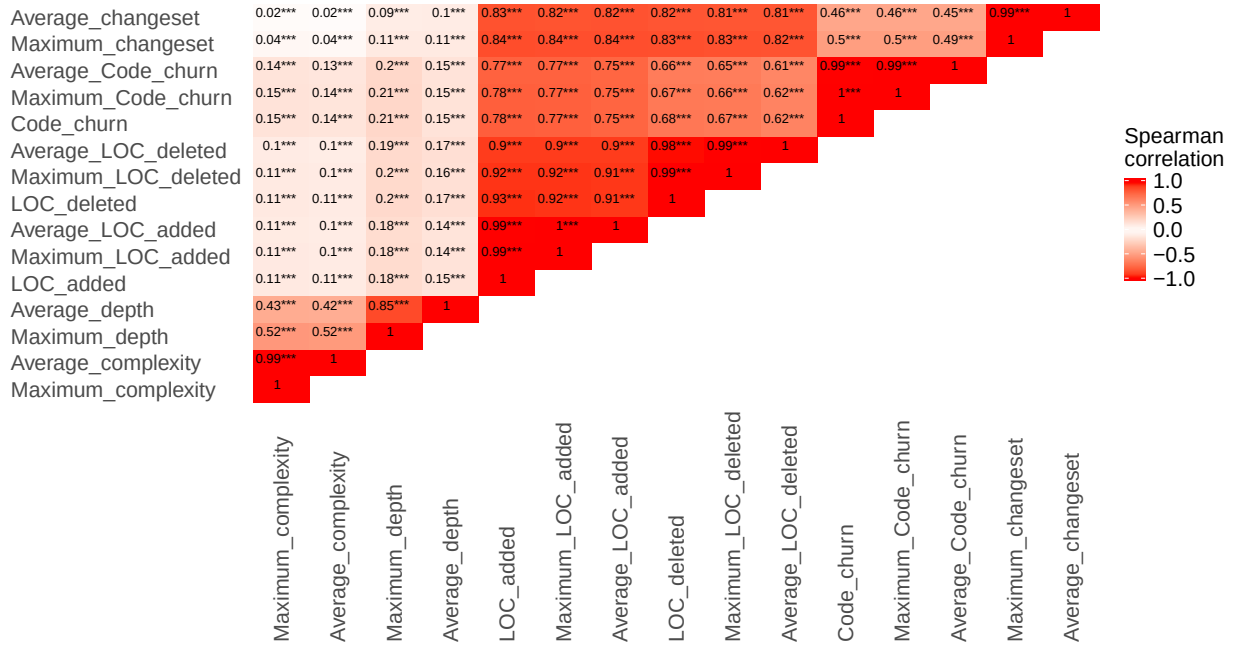


Figure 6.2: Spearman Correlation Test for Redundant Variables

Table 6.1: Static code and change variables definitions [2]

Static code variables	
Variable	**Definition**
LOC	Total number of lines
Statements	Any LOC terminated by ';'
Percent Branch Statements	Percentage of statements causing a break in sequential execution, e.g., if, for, try, throw
Method Call Statements	All method calls, in statements and logical expressions
Percent Lines with Comments	Percentage of comments lines
Classes and Interfaces	Total number of classes and interfaces, including anonymous inner classes
Methods per Class	Total method count divided by the total classes
Ave Statements per Method	Total number of statements found inside of methods divided by the number of methods
Ave Block Depth	Sum of all method block depths divided by the number of of methods
Average Complexity	Sum of all method complexity values divided by the number of methods
Change variables	
Variable	**Definition**
Revisions	Number of revisions made to a file
Refactorings	Number of times a file has been refactored
Bugfixes	Number of times a file was involved in bug fixing (pre-release bugs)
Developers	Number of distinct authors who revised the file
Code Churn	Sum of (added LOC + deleted LOC) over all revisions
Age	Age of a file in weeks (counting backwards from a specific release)
Weighted Age	$\frac{\sum_{i=1}^{n} Age_{(i)} \times LOCAdded_{(i)}}{\sum_{i=1}^{n} LOCAdded_{(i)}}$
Ave Changeset	Average number of files committed together to the repository

univariate or multivariate test [211]. However, the impact of univariate non-normality on the Maximum Likelihood (ML) estimation has been suggested to be 2 for univariate skewness and 7 for univariate kurtosis [213, 214, 212]. In addition, the suggested multivariate cutoff that does not affect the ML estimation is 3 for the kurtosis test [215].

If we face a severely skewed distribution that makes the estimation intolerable, the first option is to discretize the data. Discretizing the data means converting data from a numerical format to a discrete, ordinal type. This method produces events equally distributed on multiple ordered categories, which can significantly improve the distribution and reduce risks associated with non-normality. The second option is to treat data the to achieve normality using the common methods, such as logtransformation or the square root transformation of all data. The third option is to keep the data and use an estimator that does not have restrictions for the normality assumption (e.g., wighted least square WLS for CB-SEM, or partial least squares PLS-SEM).

The result of our data distribution is shown in Figure 6.3. We conducted the multivariate and univariate normality tests, according to Royston and Shapiro-Wilk tests [216, 217]. The results indicated that variables are not normally distributed with 95% confidence level. Some variables showed severe skewness and kurtosis and other variables were within the acceptable range (i.e., Skewness < 2 and Kurtosis < 7). Figure 6.3 shows the histogram distribution of all independent variables. As shown, most of the independent variables are heavily skewed and face a significant amount of Kurtosis (i.e., horizontal spread). Therefore, we need to consider an estimation method that does not require normal distribution.

Sampling

The sampling step determines the number of files that we should use for the study. The sampling method should ensure that observations are independent and the size of the sample is sufficient to get unbiased estimation. Independent observations can be handled through a random sampling [210]. The size of the sample should be at least at the ratio of 5:1 (i.e., five observations for every parameter) [210, 218]. The ratio of 10:1 has been suggested to achieve the best estimation [218, 219]. Barclay et al. [219] suggested using "ten times" rule for sampling, which means that the final set of independent variables should be multiplied

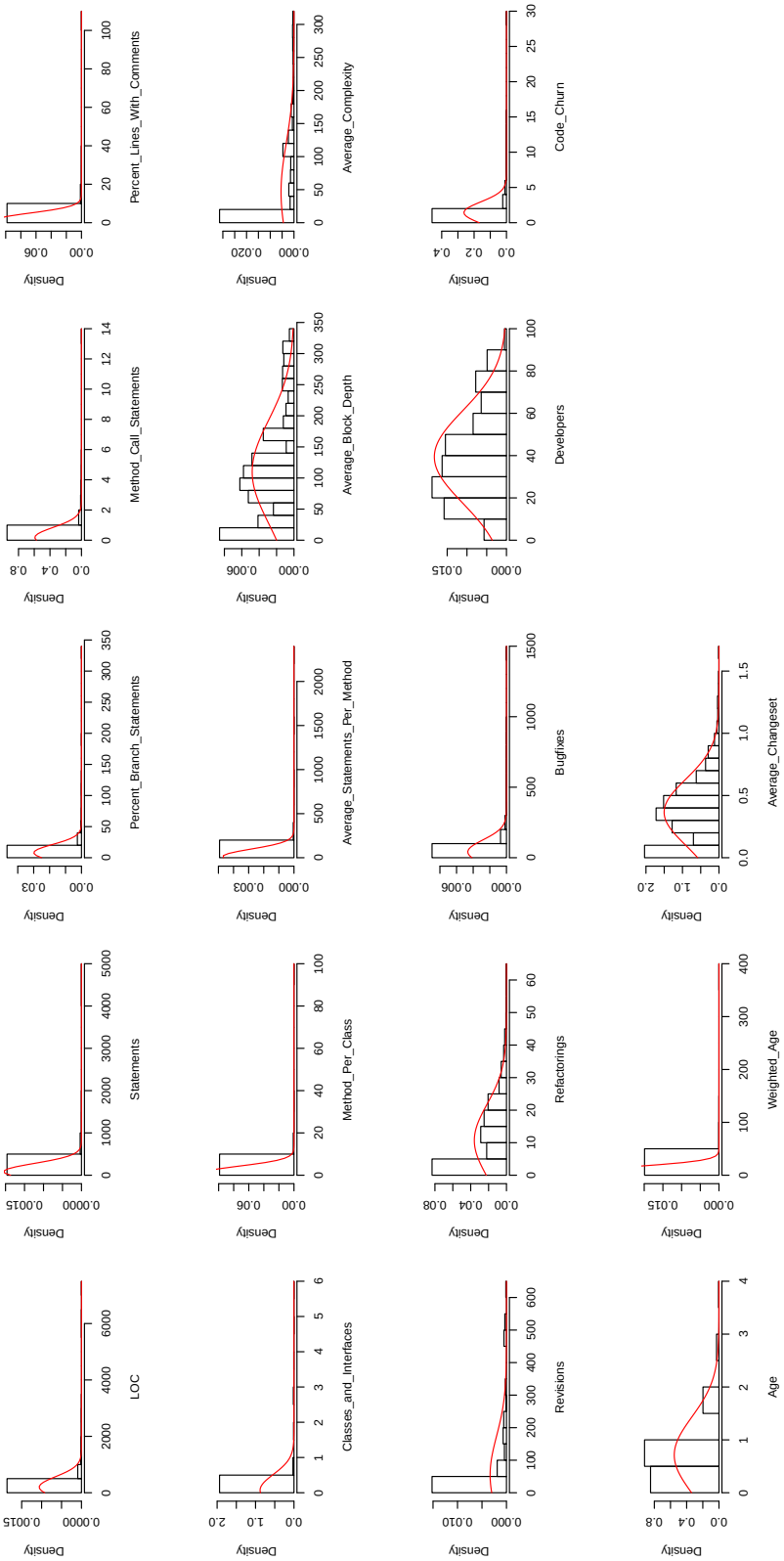


Figure 6.3: Distribution of the initial independent variables

by ten, or the largest number of path directed to a latent variable. In this study, we follow the "ten times" rule to ensure the best possible estimation. Therefore, the size of the sample is determined after the model specification step, when the final set of independent variables is decided and the final paths are specified.

6.3.1 Model specification of Europa

Model specification stage determines the final selected variables, latent variables, type of relationships between independent variables and latent variables, and path directions between latent variables. This is based on the previous knowledge and based on the understating nature of the data. Two types of SEM models exist: measurement models and structural models [215]. A measurement model explains the relations between latent variables and observed variables. A structural model explains the relations between latent variables.

Measurement models

In this section, we decide the number of latent variables and independent variables connected to every latent variable. This process analyzes the loading of all independent variables to the latent variables. Additionally, we measure correlation coefficients of the independent variables under every latent variable.

This step is helpful regardless of what approach we decide to use (CB-SEM or PLS-SEM). Variables with low loading factor are not recommended to be used in the SEM. Adding a variable with low loading results in low covariance gained for the CB-SEM and minor addition to the average extracted variance for the PLS-SEM approach. It is important to mention that both approaches conduct a factor analysis when estimating the measurement models, which allows eliminating variables during building the SEM model. However, building the model with a proper selection of variables helps to achieve the optimal model (with a good fit) early and reduce the effort results from model modifications.

In our study, we have a major response variable, which whether a file has Postrelease bugs. Because we have only a single response variable, this variable is presented in the final model attached alone to a latent variable. We used the eighteen variables from the static code and change variables resulted from the previous step. Static code variables were grouped under a latent variable called *Code Characteristics*. We divided the change variables into two categories: (1) *Change Request* latent variable, which includes a number of Bugfixes, number of Refactorings the file went through, number of revision, and number of developers, and (2) *Change Code* latent variable, which includes amount of change (e.g., Code Churn and Average Changeset). Other independent variables (e.g., Age, Weighted Age) were included with the *change code* latent variable. The initial measurement models are shown in Figure 6.4. The relationship between independent variables and latent variables is formative, which specifies that independent variables caused latent variables. For these variables, we estimate the outer regression weights after we test the correlation coefficients of every group of independent variables under every latent variable. The measurement models are represented by the equations 6.1, 6.2, 6.3, and 6.4, respectively. In the four Equations, the β is the weight regression associated with every independent variable, X_i represents independent variables, and ζ_i and e_1 are the disturbance or the error variance of the the four regression models.

$$\text{Code characteristics} = \sum_{i=1}^{n} \beta_i \times X_i + \zeta_1 \tag{6.1}$$

where n is the number of variables that describe the code characteristic latent variable

$$\text{Change request} = \sum_{i=n+1}^{n+m} \beta_i \times X_i + \zeta_2 \tag{6.2}$$

where m is the number of variables that describe the change request latent variable

$$\text{Change code} = \sum_{i=n+m+1}^{n+m+l} \beta_i \times X_i + \zeta_3 \tag{6.3}$$

where l is the number of variables that describe the change code latent variable

$$\text{Postrelease bugs} = \beta \times \text{Bugs} + e_1 \tag{6.4}$$

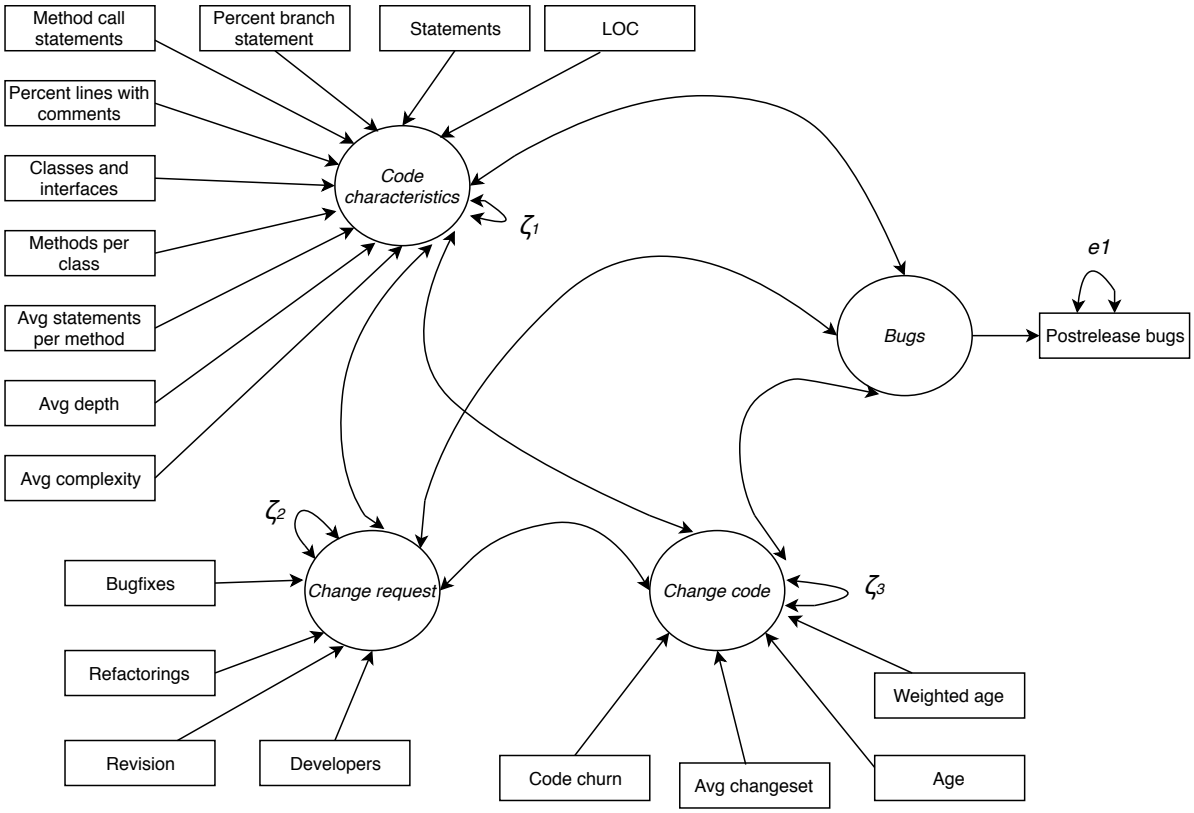


Figure 6.4: Measurement model

Pair-wise correlation test was applied for variables using the Spearman test. Three separate tests were conducted for independent variables of every latent variable as shown in Figures 6.5a, 6.5b, and 6.5c. Spearman correlation coefficient between two variables should not reach 0.7 as the maximum value [140]. The existence of a correlation between any two independent variables under the same latent variable introduces type II error in the model [220].

As shown in Figure 6.5a, LOC and Statements were highly correlated with several other variables. Therefore, both of them were excluded from the measurement models. Method Call Statements and Percent Branch Statements were both highly correlated with Average Statements per Method and Average Depth. We eliminated both Method Call Statements and Percent Branch Statements to keep the other variables, which describe the complexity and size of methods. The number of Classes was also eliminated because it was highly corre-

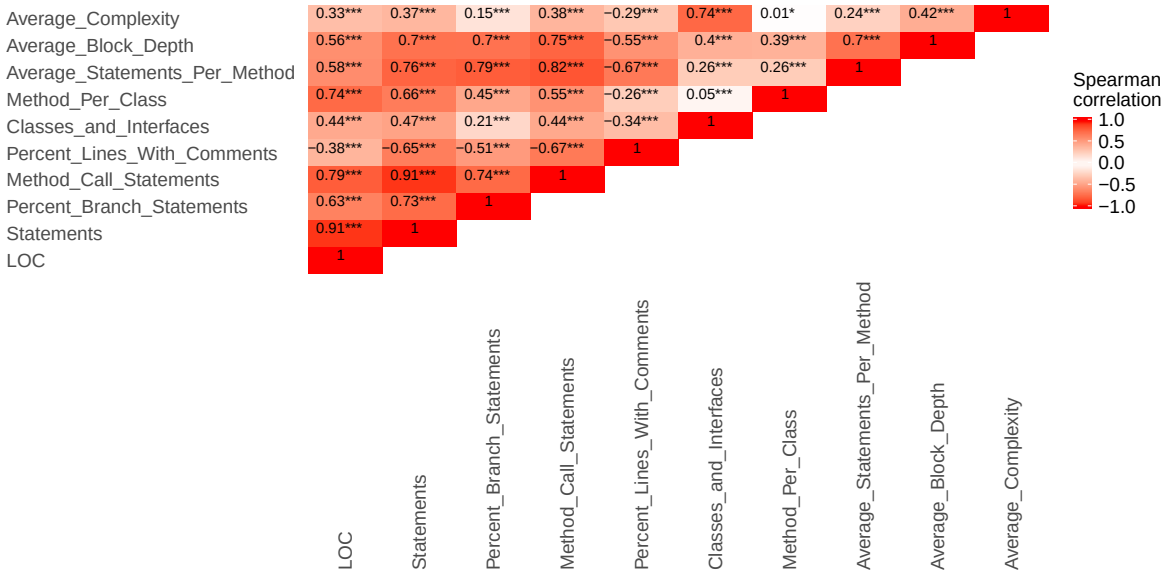
lated with Average Complexity. The Percent Lines with Comments variable was eliminated because it was highly correlated with Average Statement per Method. So, the final selected variables describe the *code characteristics* are Method per Class, Average Statements per Method, Average Depth, and Average Complexity.

Figure 6.5b shows that revisions variable was highly correlated with Bugfixes and number of Developers. Therefore, Revisions variable was eliminated from the *Change Request* latent variable. The change request is described by the number of Developers, the number of times a file involved in Bugfixes process, and the number of times a file was refactored.

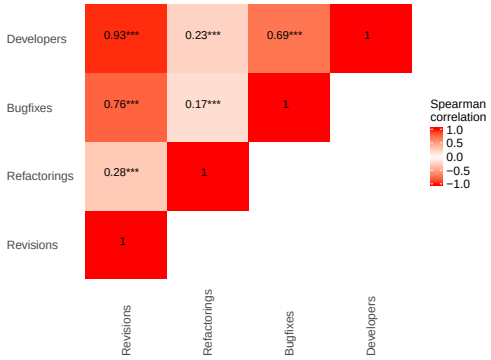
Figure 6.5c shows that Weighted Age was highly correlated with the Average Changeset. Therefore, we eliminated the Weighted Age from the *change code* latent variable. *Change code* is described by the Code Churn, Average Changeset, and Age.

After conducting the pair-wise correlation test between independent variables under every latent variable, our final number of independent variables is nine. Therefore, according to the "ten times" sampling rule, we should have no less than a ninety observation for the estimation. In the next section, we specify the total number of paths and based on that we decide the final sample size for this case study.

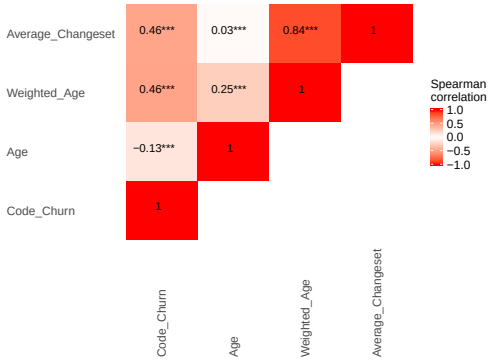
Analysis of the measurement models and the regression weight of every independent variable to the latent variables are presented in Table 6.2. As shown in the Figure 6.4, the relationship is formative between *Code Characteristics*, *Change Request*, and *Change Code* latent variables and the independent variables. Every latent variable is approximated by its assigned independent variables [221]. The latent variable score is calculated based on the weighted sum of its variables, which can help to get the best variance possible from all variables [221]. We report the weight regression for all independent variables (i.e., outer weights) and present them in Table 6.2. The results reported in the table contains the 2.5% lower and 97.5% upper weights resulted from the 500 bootstrap samples. The weights for the independent variables affecting the *code characteristic* LV are Method per Class (weight = 0.75), Average Complexity (weight = 0.46), and Average Statement per Method (weight = 0.43). For change request LV, Bugfixes (weight = 0.92), Developers (weight = 0.20), and Refactorings (weight= -0.01). For the *Code Change*, the weights are Code Churn (weight = 0.98), Average Changeset (weight=0.14), and Age (weight = -0.09).



(a) Code characteristics latent variable



(b) Change request latent variable



(c) Change code latent variable

Figure 6.5: Change request and change code latent variables correlation test results

Along with model specification, we discuss the issue of multicollinearity, which can introduce bias to the estimation of the model. Multicollinearity should be treated without affecting the model specification by ignoring important variables [222]. Several signs indicate the existence of multicollinearity in any regression model [10]. For example, regression weights should not change radically when we include or exclude a single variable or when one or more eigenvalues approach zero. Before estimating the models, we need to ensure that no multicollinearity exists in our models. A model can be affected by multicollinearity when variables under that model are highly correlated. In other words, SEM cannot be affected by multicollinearity if the highly correlated variables are linked to different latent variables.

Table 6.2: Outer weights of the independent variables with 2.5% lower L and 97.6% upper U values of the measurement models

Variable	Code characteristics (L-U)	Change request (L-U)	Change code (L-U)	VIF for measurement models
Methods per Class	**0.75*** ** (0.63-0.84)			1.00
Average Statements per Method	**0.43*** ** (0.35-0.57)			1.00
Average Complexity	**0.46*** ** (0.35-0.57)			1.00
Refactoring		**-0.01*** ** (-0.03-0.01)		1.26
Bugfixes		**0.92*** ** (0.88-0.94)		1.31
Developers		**0.20*** ** (0.17-0.25)		1.11
Code Churn			**0.98*** ** (0.95-0.99)	1.00
Age			**-0.09*** ** (-0.16- -0.05)	1.00
Average Changeset			**0.14*** ** (0.10-0.22)	1.00
*** p value < 0.001 , ** p value < 0.01 , * p value < 0.05, + p value < 0.1				

However, this is not applicable in practice because highly correlated variables are usually linked to the same latent variable. We tested the pair-wise correlation of all metrics under every latent variable to ensure that the correlation did not exceed 0.70. A second diagnostic method to ensure that all models are free from multicollinearity is to measure the VIF.

SEM calculates VIF for the measurement and the structural models. The measurement models calculate VIF values for all independent variables that are connected to latent variables. As shown in the last column of Table 6.2, VIF values for all independent variables are very close to one. Noter that no solid agreement exists regarding the acceptable level of VIF values. Some studies have argued that the VIF values of variables should be less than five (e.g., [8]). Other studies have accepted VIF values that reached ten (e.g., [6, 4]). In SEM, a study [223] accepted the 10 as a rule of thumb for VIF and another study [224] restricted to use 3.3 as a rule of thumb for VIF. All our VIF values are far below these values, which clearly indicates that our SEM measurement models are free from multicollinearity. This also should reflect on the VIF values of the structural models which we present in the next section.

Structural model

In this section, we use the final selected set of variables and specify them in our structural models. Specification of structural models includes determining the relationships between variables and latent variables and between latent variables. This also includes to determining the directions of the model and where arrows should be pointing.

Before determining the directions between latent variables, we present the pair-wise correlation between latent variables. The correlation of the latent variables helps to determine the existence of the cause-effect relationship between the two connected latent variables. The direction of the arrow is hypothesized based on our understanding of the data and the software development processes. As shown in Table 6.3, the highest correlation are between *change request* and *Bugs* and between *change request* and *Change Code*. This indicates number of Bugfixes, Refactoring, and developers involved affect Postrelease bugs and amount of change on the code. Similarly, *Change Code* are correlate with *Bugs*, which indicate amount of change on the code impact the Postrelease bugs. *Code Characteristics* are correlated at the same level with all other latent variables, which indicates that code characteristics have similar direct and indirect affect to Postrelease bugs and to code change.

Table 6.3: Correlation coefficients of latent variables

Code Characteristics			
0.126	Change request		
0.136	0.530	Change Code	
0.135	0.604	0.364	Bugs

The relation between latent variables can be endogenous or exogenous. An exogenous variable is indicated by an arrow which is pointing away from the variable (i.e., independent variable). An endogenous variable is indicated by an arrow pointing to the variable (i.e., dependent variable). In some cases, we may have a latent variable with an arrow pointing in and an arrow pointing out. This is because, in SEM, we could have multiple dependent variables.

We have four latent variables: *Code Characteristics*, *change request*, *Change Code*, and *Bugs*. *Bugs* LV is endogenous where all arrows pointing at, because we believe that all other LV contribute to the Postrelease bugs including the *Code Characteristics*, *Change Code*, and *Change Request* when files experienced bugfixing or Refactoring or when more Developers are involved in the file. *Change request* LV, which includes the number of Bugfixes, Refactorings, and Developers, is affected by the *Code Characteristics*. Complex files with a high number of methods are always in need of Refactoring and Bugfixes and more Developers involved. Similarly, *Change Code* LV is caused by both *Change Request* and *Code Characteristics*. *Change Code* are described by the Code Churn[3], Average Changeset and the Age of the file is believed to be affected by the amount of *Change Request* and *Code Characteristics*. The structural models with all relationships and measurement models with final sets of selected variables are shown in Figure 6.6. The structural models are represented by Equations 6.5, 6.6, and 6.7, respectively. In the three equations, ζ is the disturbance resulted from the three regression models, and β_i is the coefficient of the regression path between two latent variables.

The maximum number of paths to a latent variable in Figure 6.6 is three. According the "ten times" rule, the sample size should be no less than thirty files.

$$\text{Change Request} = \beta_{12} \times \text{Code Characteristics} + \zeta_2 \tag{6.5}$$

$$\text{Change Code} = \beta_{13} \times \text{Code Characteristics} + \beta_{23} \times \text{Change Request} + \zeta_3 \tag{6.6}$$

$$\text{Bugs} = \beta_{14} \times \text{Code Characteristics} + \beta_{24} \times \text{Change Request} + \beta_{34} \times \text{Code Change} + \zeta_4 \tag{6.7}$$

Table 6.4 presents the VIF results of the structural model. In the structural models VIF values for latent variables are estimated in every model. We have total of three structural models and every endogenous latent variable represents a response variable for the structural models. All the VIF values presented are close to one and far below the cut off values discussed in [223, 224].

[3]The summation of lines of code added and deleted

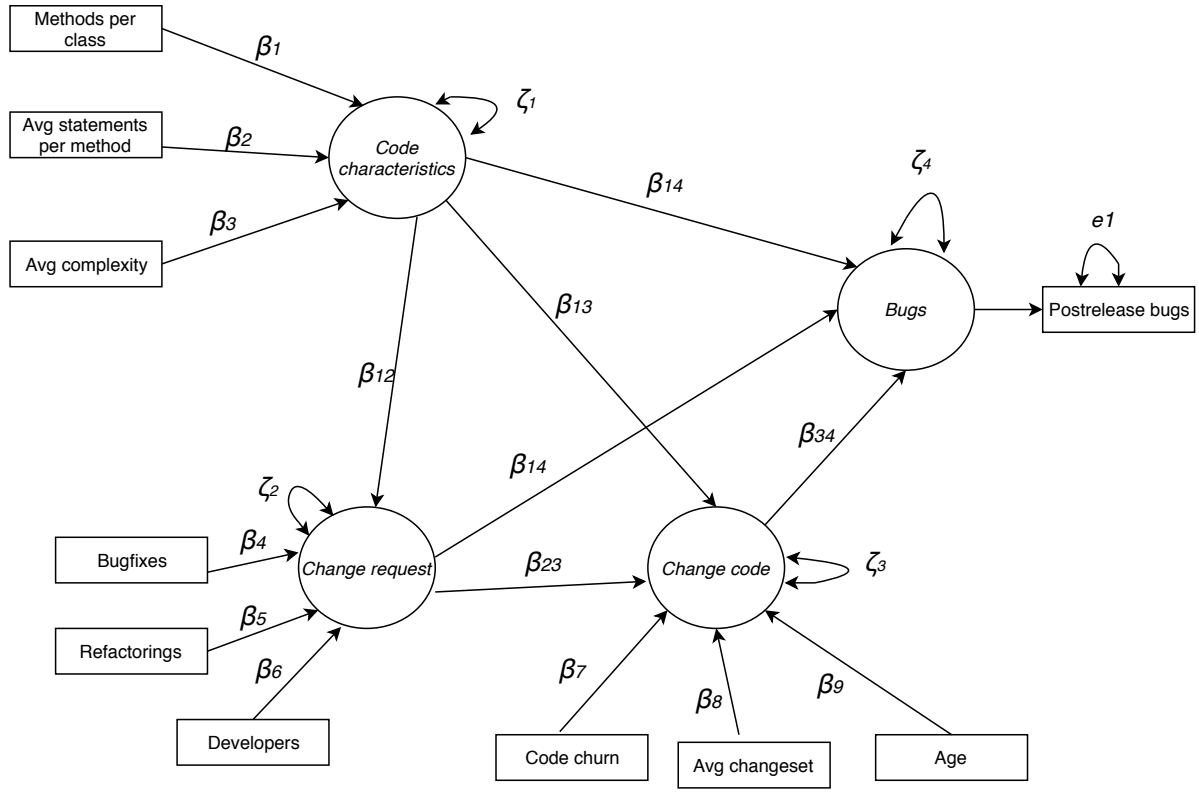


Figure 6.6: Structural model

Table 6.4: Variance inflation factor for the structural models

VIF for the structural models			
Endogenous LV	Exogenous LV1	Exogenous LV2	Exogenous LV3
Change request	Code Characteristics		
	VIF= 1.00		
Change Code	Code Characteristics	Change Request	
	VIF=1.00	VIF=1.00	
Bugs	Code Characteristics	Change Request	Change Code
	VIF= 1.41	VIF= 1.39	VIF=1.02

6.3.2 Model estimation

Model estimation is conducted after the model specification which included, assigning the final sets of variables linked to every latent variable, and after assigning the directions of arrows between latent variables. In this step, both measurement models and structural models are estimated. To achieve that, we have two potential approaches to choose from CB-SEM and PLS-SEM.

The CB-SEM relies more in the normality assumption of the data. The most common method used with the CB-SEM is the ML method [214] because it provides unbiased estimation, consistency, and efficient parameter estimation under the assumption of normality with a sufficiently large sample [225]. The only problem with ML is the strong assumption of multivariate normality [214], which makes it not applicable when the distribution of data is heavily skewed like in our case study. Some studies suggest that ML can tolerate the moderate level of skewness and kurtosis (i.e., when the skew test is less than three, and kurtosis is less than seven) [206]. However, when skewness and kurtosis are larger, then ML can be used with a scaling procedure (i.e., Satorra-Bentler or Yuan-Bentler) [226, 11], or use the weighted least square (WLS) estimation [214]. WLS or diagonally weighted least square (DWLS) require large sample of data and may not work well with large number of variables [227, 212].

PLS-SEM provides alternative method to the covariance based CB-SEM [221, 116, 209]. The main advantage of PLS path modeling is that it supports exploratory and confirmatory research, while CB-SEM supports only confirmatory research [209, 223]. In other words, to implement CB-SEM, a strong theory base is needed and that is not required with PLS path modeling. In the model specification, we used both formative and reflective relationships. *Code Characteristics*, *change request*, *Change Code* latent variables are connected with independent variables through formative relationships as shown in Figures 6.6 and 6.4. The same two figures present the relationship between *Bugs* latent variable and Postrelease bugs in an reflective relationship. Additionally, PLS path modeling does not require strong assumption in data normality, sample size, and measurement scale [228]. PLS-SEM provides a robust estimation of the SEM [209].

As presented in Figure 6.3, the distribution of the final selected variables are heavily skewed (e.g., Average Statements per Method, Methods per Class, Bugfixes and Code Churn). Additionally, we use a total of ten variables and four latent variables. Three latent variables are exogenous and endogenous (i.e., *Change Request*, *Change Code*, and *Bugs*) and one latent variable exogenous (i.e., *Code Characteristics*). Using PLS-SEM is necessary be-

cause of using a combination of both formative and reflective relationships. PLS-SEM handle one or more variable connected to a latent variable [229, 209]. PLS-SEM is recommended with complex relationships in the structural models between latent variables [229, 209]. Thus, PLS-SEM is more appropriate than CB-SEM for our research goal and case study.

PLS-SEM estimates the measurement and structural models separately. PLS-SEM estimation goes through two stages: (1) estimation of latent variables scores and (2) estimation of the loadings of independent variables and path coefficients [229]. Stage one is made through iterative process before moving to the second stage, which produces the final estimation of the measurement model loadings and path coefficients through ordinary least squares regressions OLS [229].

Table 6.5: Results of the structural models

Endogenous LV	Exogenous LV	Estimate	Standard error	t-value	P value	Significance	R^2
Change request	Code Characteristics	0.11	0.00	18.90	< 0.01	***	0.27
Code Change	Code Characteristics	0.09	0.00	19.30	< 0.001	***	
	Change request	0.51	0.00	106.00	< 0.001	***	0.42
Bugs	Code Characteristics	0.04	0.00	10.60	< 0.001	***	
	Change request	0.66	0.00	136.00	< 0.001	***	
	Change Code	0.01	0.00	225.00	< 0.001	***	0.53
*** p value < 0.001 , ** p value < 0.01 , * p value < 0.05, + p value < 0.1							

Table 6.6: Direct and indirect effects, with 2.5% lower and 97.5% upper bootstrap samples

Relationship	direct	indirect	Total effect (U-L)
Code Characteristics $->$ Change Request	0.10	0.00	0.10 (0.08-0.13)
Code Characteristics $->$ Change Code	0.09	0.06	0.14 (0.10-0.20)
Code Characteristics $->$ Bugs	0.04	0.08	0.11 (0.09-0.14)
Change Request $->$ Change Code	0.51	0.00	0.51 (0.43-0.61)
Change Request $->$ Bugs	0.66	0.00	0.67 (0.51-0.77)
Change Code $->$ Bugs	0.01	0.00	0.01 (-0.02-0.04)

The three structural models with t statistics and p values are presented in Table 6.5. Additionally, we analyzed the direct and indirect effects between latent variables, as shown in Table 6.6. The results presented in Table 6.6 contain the 2.5% lower and 97.5% upper values resulted from the 500 bootstrapping samples, which was conducted to validate the model.

Code Characteristics affect both *Change Request* and *Code Change* (path coefficients β =0.10 and 0.09) 100% higher than the direct impact of code characteristics (path coefficient $\beta = 0.04$) on the Postrelease bugs. This suggests that Postrelease bugs occur when the code is modified for bug fixing or when more Developers are involved more than the faults due to the initial design of the code. The model shows that indirect impact (path coefficient β = 0.08) of code characteristics on Postrelease bugs is 100% higher than the direct impact, which supports the claim that says that the bugs are due to the changes of the code.

Change request cause the highest impact on Postrelease bugs (path coefficient $\beta = 0.66$). Also, Change request causes high impact on the *Code Change* (path coefficient $\beta = 0.51$). The indirect cause of *Change Request* through the *Code Change* is very minor. This is because *Change Code* had a very small effect (path coefficient $\beta = 0.01$) on Postrelease bugs, although change request has a major impact (path coefficient $\beta = 0.51$) on changing the code. However, it seems that the involvement of the code change were mostly successful because the number of faults caused by this process were considerably lower. This suggests that involvement of the Code Churn, and Average Changeset are good remedies for any request for changing the code. Also, involving more Developers, and more Bugfixes (i.e., Prerelease bugs) had a significant impact on Postrelease bugs. Refactorings have a minor regression weight (β = -0.01), which suggests that the major issue is through the Bugfixes and the number of Developers.

A summary of the answers to the research questions is given in Table 6.7.

6.3.3 Model validation and goodness of fit

The reliability and discriminant validity of the formative (i.e., arrows are pointing to the latent variables from the independent variables) measurement models are validated [207, 230, 220] using: (1) examining the variable's weight and its significance level (variables with insignificant weight should be removed from the model), as presented in Table 6.2, (2) VIF values of all independent variables to assess the multicollinearity, as presented in Table

Table 6.7: Summary of the research questions

RQ	Description	Result	Evidence
RQ1:	How many underlying factors (i.e., latent variables) can be found in the whole set of static and change metrics?	4	We introduced and tested the existence of four latent variables: (1) *Code Characteristics* described the static code variables, (2) *Change Request* described the number of time the file was involved in bugfixing issue, or Refactorings, and the number of Developers were involved in modifying the file, (3) *Change Code* described the amount of change in the code of the file, amount of files changed at the same time, and the age of the file, and (4) *Bugs* described the number of time the file experienced Postrelease faults.
RQ2:	What variables have significant loading factors linked to every latent variable?		In the *Code Characteristics*, variables are methods per class (weight regression β= 0.75), Average Statements per method (β= 0.43), and Average Complexity (β= 0.46). In the *Change Request*, variables are Bugfixes (β= 0.43), Refactorings (β= -0.01), and Developers (β= 0.20). In the *Code Change*, variables are Code Churn (β= 0.98), Average Changeset (β= 0.14), and Age (β= -0.09).
RQ3:	What latent variable has the highest direct impact on software bugs?	*Change Request* LV	Change request LV has the highest impact on Postrelease bugs (β = 0.66). Also, the *Change Request* LV impacted the *Code Change* with a path coefficient = 0.51.
RQ4:	Does any latent variable have an indirect effect on software bugs, and what is its impact?	Yes	*Code Characteristics* LV has an indirect effect on Postrelease bugs through change request LV, estimated at 0.08.

6.2, and (3) highly correlated under every latent variables were removed and all correlation coefficients are under 0.7. Also, we used 500 bootstrap samples to present the 2.5% lower and 97.5% upper values of regression outer weights of all independent variables, as shown in Table 6.2.

Structural models are validated by checking the multicollinearity level of VIF of all exogenous latent variables. Low VIF of independent variables lead to low VIF at the structural models. All VIF values of the structural models are reported in Table 6.4, which shows that all VIF are very low and suggest that no multicollinearity issue is detected. Additionally, structural models require measuring the R^2 of all endogenous latent variables. As shown in Table 6.5, R^2 values are presented for each structural model, which indicates the variance explained by every endogenous latent variable. The number increases as the more latent variable and more independent variables are added to the model. Additionally, the bootstrapping is required to be used to assess path coefficients with a minimum number of bootstrap sample of 500.

6.4 Threats to validity

The objectives of this research were clearly defined at the beginning. We measured what we intended to measure in this chapter. Also, all terms, variables, and methods were clearly defined for this study. Additionally, we analyzed the data to remove redundant variables to use the effective ones and the proper data for the study. These steps were taken to avoid threats to construct validity. Other variables can be included in the SEM. However, with included variables we achieved a good fit to the observed data.

The objective of internal validity is to ensure the quality of the data. The data were extracted in the lab, and all processes that could ensure the quality of the data were taken into account, including a sanity checks of different variables and manual inspection of randomly selected files to check whether the metrics are correct fo these files.

Conclusion validity can be violated by not applying statistical tests properly or using the wrong test. For that reason, we ensured that all tests were used properly to ensure the correctness of the results. We assessed the level of normality of our data, and based on that, we selected the proper estimation method using the PLS-SEM path modeling that can tolerate the level of skewness of the data. PLS-SEM allows more importantly exploratory study.

Threats to external validity can happen when generalzability is claimed. The results of this work hold for Eclipse project Europa release. Our future work will include building SEM models using other datasets to explore the generalzability of the results.

6.5 Conclusion

In this chapter, we introduced the causality method to the field of software fault proneness for the first time. We used a combination of statistical and regression approaches to building the causal model for the Eclipse project using a static code and change variables. We applied the PLS-SEM due to the nature of our data and the need to use a formative variable, and to the exploratory support for studies that exist in the PLS-SEM path modeling. We presented measurement and structural models based and find the direct and indirect cause

from several paths led to the Postrelease bugs. We found that code characteristics latent variable has the direct and indirect effect to Postrelease bugs. The indirect effect of the code characteristics has 100% higher than the direct effect to Postrelease bugs. Change request, represented by Bugfixes, Refactorings, and the number of Developers, had the largest effect to the Postrelease bugs. Code change (represented by Code Churn, Average Changeset, and Age) had a small effect on the Postrelease bugs.

Future work in this area includes applying this method on other projects to explore the generalzability. Additionally, applying the same methodology in other areas of software engineering, such as effort estimation, is necessary. Further, applying the method to different sets of variables is important, as is applying this on different phases of the software development cycle. Additionally, the future work will explore the applicability of BN to this type of work.

Chapter 7

Conclusion

This dissertation is focused on explanatory, prediction, and causality approaches in software engineering. Specifically, we developed a methodology that can be used by software practitioners to explain and quantify the effect of confounders and their interactions on post release fault proneness. Besides their capabilities to explain software fault proneness, we explored the prediction performance of the explanatory models. Furthermore, we proposed a methodology to deal with software development data, building categorical models to understand and predict the software development efforts. Last but not least, we used a structural equation modeling (SEM) to explain causal relationships between independent confounders and software fault proneness.

For the first explanatory work, we used a case-control study for the first time in the field of software engineering, on software fault proneness data sets. We used static code and change confounders to model the Eclipse and Apache projects. We ensured that no high correlation existed among all independent confounders considered in the explanatory model. We also included the interactions of confounders in the model and ensured that multicollinearity did not exist. Then, we eliminated insignificant interactions and moved on to remove insignificant confounders based on backward hierarchal elimination process. The final models of all Eclipse and Apache projects were statistically tested for goodness of fit. Change confounders, such as Bugfixes, Age, Developers, and Code Churn, were statistically significant. In contrast, static code confounders were not statistically significant in most software projects. We compared the results across projects for consistency and exploring

generalizability. We found consistent results of some confounders and interactions across many projects. The following main results are of interest to software project managers. Since bugfixes significantly affect the Postrelease bugs, software practitioners should pay attention and improve the pre-release bug fixing process. Also, more Developers seemed to increase the risk for Postrelease bugs. However, more Developers assigned in bug fixing process seemed to be helpful and reduce the risk of Postrelease bugs. Also, new files with more Developers seemed to increase the risk of software faults.

It was important to measure the prediction performance of the models that were initially built for the explanatory purpose. Therefore, we measured and reported the performance of all explanatory models. Furthermore, we compared their performances with other top-performing, widely used classifiers used for software fault proneness prediction. We found that the prediction of the explanatory models was not statistically significant different than other classifiers. In addition, we used a classifier based on group lasso regression for the first time in this area. We found that this classifier performed similarly to the most other classifiers.

Second, this dissertation provided an explanatory work in the area of software development effort. The work involved confounder selection based on earlier findings, and correlation and association tests. Confounders were discretized to multiple levels. The study applied the categorical regression algorithm and included confounders and interactions of confounders. The models were tested for multicollinearity before proceeding to eliminate insignificant interactions and confounders based on hierarchal backward elimination methodology. The final model was statistically tested for goodness of fit. In addition, performances of the final models were measured and compared with related works. Also, we measured the classification performance of each level of software development effort, which measured the likelihood of the effort levels to be predicted at the correct class. We used ISBSG, Desharnais, and Maxwell data sets for the software development effort. Our results in the area of software development effort are useful to software projects managers and practitioners. In ISBSG dataset, a reasonable increase in the effort was observed as the size of the project changed from level 1 to 2, from level 2 to 3, and from level 3 to 4. However, the time spent to develop the project (i.e., duration) considerably increased the effort when it changed from level 2 to

3 (i.e., from 4 to 6 months duration to 7 months and higher duration), which indicated that the duration is more critical than the size of the project. Similarly, we found that assigning a team with a high number of Developers considerably affected the effort, especially when the team size exceeded 15 persons per task.

Lastly, we conducted a causality study based on the structural equation modeling technique, which involved a combination of statistical tests and regression analysis to find the causal effect to software fault proneness. We used Eclipse's Europa release as a case study for this work and involved static code and change confounders. Confounders were selected and latent variables were constructed based on exploratory factor analysis. We ensured the correlation among confounders within the same latent variable, and we tested the variance inflation factors of all confounders to eliminate multicollinearity. The final structural models contained four latent variables and eight confounders. The final causal model was statistically tested for goodness of fit using the most common statistical tests of this area. In addition, direct and indirect effects were observed from latent variables to the response confounder. The results from the causality analysis are very important because they are related to latent variables and measured the direct and indirect causes of many latent variables. Specifically, we found that code characteristics metrics did not heavily contribute directly to Postrelease bugs. The indirect effect between code characteristics and Postrelease bugs was higher than the direct effect. The results also indicated that changing the code (adding and deleting lines to the source file) did not heavily contribute to Postrelease bugs. The largest cause of Postrelease bugs was coming from the change requests, especially from the number of Bugfixes and number of Developers.

Future work will include extracting more and different types of confounders for the areas of software fault proneness and software development effort. Furthermore, we will explore the possibility to apply categorical regression on software fault proneness by discretizing the number of faults to several levels. In addition, we plan to explore the use of causality studies on other software projects of software fault proneness, as well as to apply the causality

analysis in software development effort area and in other areas of software engineering. Last but not least, future work may consider data sets from software projects that use different life cycle models and how these models affect the software fault proneness and software development effort.

Bibliography

[1] A. B. Nassif, L. F. Capretz, and D. Ho, "Analyzing the non-functional requirements in the desharnais dataset for software effort estimation," *arXiv preprint arXiv:1405.1131*, 2014.

[2] T. Devine, K. Goseva-Popstojanova, S. Krishnan, and R. R. Lutz, "Assessment and cross-product prediction of software product line quality: Accounting for reuse across products, over multiple releases," *Automated Software Engineering*, pp. 1–50, 2014.

[3] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, 2009.

[4] N. Bettenburg and A. E. Hassan, "Studying the impact of social interactions on software quality," *Empirical Software Engineering*, vol. 18, no. 2, pp. 375–431, 2013.

[5] E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan, "Understanding the impact of code and process metrics on post-release defects: A case study on the eclipse project," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 4, 2010.

[6] N. Bettenburg and A. E. Hassan, "Studying the impact of social structures on software quality," in *IEEE 18th International Conference on Program Comprehension (ICPC)*, pp. 124–133, 2010.

[7] P. Tourani and B. Adams, "The impact of human discussions on just-in-time quality assurance: An empirical study on openstack and eclipse," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2016*, vol. 1, pp. 189–200, IEEE, 2016.

[8] M. D. Syer, M. Nagappan, B. Adams, and A. E. Hassan, "Studying the relationship between source code quality and mobile platform dependence," *Software Quality Journal*, vol. 23, no. 3, pp. 485–508, 2015.

[9] W. Shang, M. Nagappan, and A. E. Hassan, "Studying the relationship between logging characteristics and the code quality of platform software," *Empirical Software Engineering*, vol. 20, no. 1, pp. 1–27, 2015.

[10] G. Maruyama, *Basics of structural equation modeling*. Sage, 1997.

[11] J. B. Ullman and P. M. Bentler, "Structural equation modeling," *Handbook of Psychology, Second Edition*, vol. 2, 2012.

[12] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 797–814, 2000.

[13] C. Andersson and P. Runeson, "A replicated quantitative analysis of fault distributions in complex software systems," *IEEE Transactions on Software Engineering*, vol. 33, no. 5, pp. 273–286, 2007.

[14] T. Galinac Grbac, P. Runeson, and D. Huljenić, "A second replicated quantitative analysis of fault distributions in complex software systems," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 462–476, 2013.

[15] T. Zimmerman, N. Nagappan, K. Herzig, R. Premraj, and L. Williams, "An empirical study on the relation between dependency neighborhoods and failures," in *IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST)*, pp. 347–356, 2011.

[16] M. Hamill and K. Goševa-Popstojanova, "Common trends in software fault and failure data," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 484–496, 2009.

[17] M. Hamill and K. Goseva-Popstojanova, "Exploring fault types, detection activities, and failure severity in an evolving safety-critical software system," *Software Quality Journal*, vol. 23, no. 2, pp. 229–265, 2014.

[18] A. Tosun Misirli, B. Murphy, T. Zimmermann, and A. Basar Bener, "An explanatory analysis on eclipse beta-release bugs through in-process metrics," in *Proceedings of the 8th International Workshop on Software Quality*, pp. 26–33, 2011.

[19] M. Hamill and K. Goseva-Popstojanova, "Exploring the missing link: An empirical study of software fixes," *Software Testing, Verification and Reliability*, vol. 24, no. 8, pp. 684–705, 2014.

[20] A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller, "If your bug database could talk," in *Proceedings of the 5th International Symposium on Empirical Software Engineering*, vol. 2, pp. 18–20, 2006.

[21] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *ACM/IEEE 30th International Conference on Software Engineering*, pp. 181–190, 2008.

[22] S. Krishnan, C. Strasburg, R. R. Lutz, and K. Goševa-Popstojanova, "Are change metrics good predictors for an evolving software product line?," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, 2011.

[23] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *International Workshop on Predictor Models in Software Engineering, 2007*, pp. 9–9.

[24] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *First International Symposium on Empirical Software Engineering and Measurement*, pp. 364–373, 2007.

[25] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Does measuring code change improve fault prediction?," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, p. 2, ACM, 2011.

[26] T. Zimmermann, N. Nagappan, and A. Zeller, "Predicting bugs from history," in *Software Evolution*, pp. 69–88, Springer, 2008.

[27] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[28] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, 2000.

[29] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pp. 284–292, 2005.

[30] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," in *ACM SIGSOFT Software Engineering Notes*, vol. 29, pp. 86–96, 2004.

[31] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Looking for bugs in all the right places," in *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, pp. 61–72, 2006.

[32] V. R. Basili and B. T. Perricone, "Software errors and complexity: An empirical investigation," *Communications of the ACM*, vol. 27, no. 1, pp. 42–52, 1984.

[33] C. Withrow, "Error density and size in ada software," *IEEE Software*, vol. 7, no. 1, pp. 26–30, 1990.

[34] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012.

[35] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 539–559, 2008.

[36] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Programmer-based fault prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, p. 19, 2010.

[37] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, no. 8, pp. 1040–1058, 2009.

[38] S. Krishnan, C. Strasburg, R. R. Lutz, K. Goseva-Popstojanova, and K. S. Dorman, "Predicting failure-proneness in an evolving software product line," *Information and Software Technology*, vol. 55, no. 8, pp. 1479–1495, 2013.

[39] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *15th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 417–428, IEEE, 2004.

[40] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[41] K. Dejaeger, T. Verbraken, and B. Baesens, "Toward comprehensible software fault prediction models using bayesian network classifiers," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 237–257, 2013.

[42] M. Chen and Y. Ma, "An empirical study on predicting defect numbers.," in *Software Engineering and Knowledge Engineering (SEKE)*, pp. 397–402, 2015.

[43] A. Okutan and O. T. Yıldız, "Software defect prediction using Bayesian networks," *Empirical Software Engineering*, vol. 19, no. 1, pp. 154–181, 2014.

[44] V. Jindal, "Towards an intelligent fault prediction code editor to improve software quality using deep learning," in *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming*, pp. 222–223, ACM, 2018.

[45] A. Arshad, S. Riaz, L. Jiao, and A. Murthy, "Semi-supervised deep fuzzy c-mean clustering for software fault prediction," *IEEE Access*, 2018.

[46] C. W. Yohannese, T. Li, K. Bashir, M. Simfukwe, and A. S. Hussein, "Software fault prediction using data reduction approaches,"

[47] H. B. Yadav and D. K. Yadav, "Early software reliability analysis using reliability relevant software metrics," *International Journal of System Assurance Engineering and Management*, vol. 8, no. 4, pp. 2097–2108, 2017.

[48] P. Zong, Y. Wang, and F. Xie, "Embedded software fault prediction based on back propagation neural network," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 553–558, IEEE, 2018.

[49] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Computing*, pp. 1–17, 2018.

[50] S. Chatterjee, S. Nigam, and A. Roy, "Software fault prediction using neuro-fuzzy network and evolutionary learning approach," *Neural Computing and Applications*, vol. 28, no. 1, pp. 1221–1231, 2017.

[51] T. Wang, Z. Zhang, X. Jing, and L. Zhang, "Multiple kernel ensemble learning for software defect prediction," *Automated Software Engineering*, vol. 23, no. 4, pp. 569–590, 2016.

[52] Z.-W. Zhang, X.-Y. Jing, and T.-J. Wang, "Label propagation based semi-supervised learning for software defect prediction," *Automated Software Engineering*, vol. 24, no. 1, pp. 47–69, 2017.

[53] O. Kohannim, D. P. Hibar, J. L. Stein, N. Jahanshad, X. Hua, P. Rajagopalan, A. Toga, C. R. Jack Jr, M. W. Weiner, G. I. De Zubicaray, *et al.*, "Discovery and replication of gene influences on brain structure using lasso regression," *Frontiers in Neuroscience*, vol. 6, p. 115, 2012.

[54] M. Y. Park, T. Hastie, and R. Tibshirani, "Averaged gene expressions for regression," *Biostatistics*, vol. 8, no. 2, pp. 212–227, 2006.

[55] S. Ma, X. Song, and J. Huang, "Supervised group lasso with applications to microarray data analysis," *BMC bioinformatics*, vol. 8, no. 1, p. 60, 2007.

[56] Y. Shimizu, J. Yoshimoto, S. Toki, M. Takamura, S. Yoshimura, Y. Okamoto, S. Yamawaki, and K. Doya, "Toward probabilistic diagnosis and understanding of depression based on functional mri data analysis with logistic group lasso," *PloS One*, vol. 10, no. 5, p. e0123524, 2015.

[57] J. Mairal, F. Bach, J. Ponce, *et al.*, "Sparse modeling for image and vision processing," *Foundations and Trends in Computer Graphics and Vision*, vol. 8, no. 2-3, pp. 85–283, 2014.

[58] B. Shen, W. Hu, Y. Zhang, and Y.-J. Zhang, "Image inpainting via sparse representation," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 697–700, IEEE, 2009.

[59] M.-D. Iordache, J. M. Bioucas-Dias, and A. Plaza, "Hyperspectral unmixingwith sparse group lasso," in *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 3586–3589, IEEE, 2011.

[60] S. S. Roy, D. Mittal, A. Basu, and A. Abraham, "Stock market forecasting using lasso linear regression model," in *Afro-European Conference for Industrial Advancement*, pp. 371–381, Springer, 2015.

[61] P. E. Engelstad, H. Hammer, K. W. Kongsgård, A. Yazidi, N. A. Nordbotten, and A. Bai, "Automatic security classification with lasso," in *International Workshop on Information Security Applications*, pp. 399–410, Springer, 2015.

[62] C. Catsburg, M. J. Gunter, C. Chen, M. L. Cote, G. C. Kabat, R. Nassir, L. Tinker, J. Wactawski-Wende, D. L. Page, and T. E. Rohan, "Insulin, estrogen, inflammatory markers, and risk of benign proliferative breast disease," *Cancer Research*, 2014.

[63] N. Kontou, T. Psaltopoulou, N. Soupos, E. Polychronopoulos, D. Xinopoulos, A. Linos, and D. B. Panagiotakos, "The mediating effect of mediterranean diet on the relation between smoking and colorectal cancer: A case–control study," *The European Journal of Public Health*, vol. 23, no. 5, pp. 742–746, 2013.

[64] B. Kitchenham, S. L. Pfleeger, B. McColl, and S. Eagan, "An empirical study of maintenance and development estimation accuracy," *Journal of Systems and Software*, vol. 64, no. 1, pp. 57–77, 2002.

[65] A. Pengelly, "Performance of effort estimating techniques in current development environments," *Software Engineering Journal*, vol. 10, no. 5, pp. 162–170, 1995.

[66] R. Kusters, M. van Genuchten, and F. Heemstra, "Are software cost-estimating models accurate," *The Economics of Information Systems and Software*, pp. 155–161, 1991.

[67] M. Jørgensen and D. I. Sjøberg, "Impact of experience on maintenance skills," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 14, no. 2, pp. 123–146, 2002.

[68] L. H. Putnam, "A macro-estimating methodology for software development," in *Proceedings IEEE COMPCON*, vol. 76, pp. 138–143, 1976.

[69] C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Systems Journal*, vol. 16, no. 1, pp. 54–73, 1977.

[70] R. W. Wolverton, "The cost of developing large-scale software," *IEEE Transactions on Computers*, vol. 100, no. 6, pp. 615–636, 1974.

[71] F. R. Freiman and R. Park, "Price software model-version 3: An overview," in *Proceedings of the IEEE Workshop on Quantitative Software Models*, pp. 32–44, 1979.

[72] B. W. Boehm *et al.*, *Software Engineering Economics*, vol. 197. Prentice-Hall Englewood Cliffs (NJ), 1981.

[73] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *Proceedings of the 18th International Conference on Software Engineering*, pp. 170–178, IEEE Computer Society, 1996.

[74] M. Azzeh, "A replicated assessment and comparison of adaptation techniques for analogy-based effort estimation," *Empirical Software Engineering*, vol. 17, no. 1-2, pp. 90–127, 2012.

[75] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung, "Exploiting the essential assumptions of analogy-based effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 425–438, 2012.

[76] L. Radlinski and W. Hoffmann, "On predicting software development effort using machine learning techniques and local data," *International Journal of Software Engineering and Computing*, vol. 2, no. 2, pp. 123–136, 2010.

[77] A. B. Nassif, M. Azzeh, L. F. Capretz, and D. Ho, "Neural network models for software development effort estimation: A comparative study," *Neural Computing and Applications*, vol. 27, no. 8, pp. 2369–2381, 2016.

[78] A. S. Andreou and E. Papatheocharous, "Software cost estimation using fuzzy decision trees," in *23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 371–374, IEEE, 2008.

[79] A. Hira, B. Boehm, R. Stoddard, and M. Konrad, "Preliminary causal discovery results with software effort estimation data," in *Proceedings of the 11th Innovations in Software Engineering Conference*, p. 6, ACM, 2018.

[80] O. Fedotova, L. Teixeira, H. Alvelos, *et al.*, "Software effort estimation with multiple linear regression: Review and practical application.," *Journal of Information Science and Engineering*, vol. 29, no. 5, pp. 925–945, 2013.

[81] L. Angelis, I. Stamelos, and M. Morisio, "Building a software cost estimation model based on categorical data," in *Seventh International Proceedings in Software Metrics Symposium*, pp. 4–15, IEEE, 2001.

[82] L. C. Briand, K. El Emam, D. Surmann, I. Wieczorek, and K. D. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," in *Proceedings of the 1999 International Conference on Software Engineering*, pp. 313–323, IEEE, 1999.

[83] L. C. Briand, T. Langley, and I. Wieczorek, "A replicated assessment and comparison of common software cost modeling techniques," in *Proceedings of the 22nd International Conference on Software Engineering*, pp. 377–386, ACM, 2000.

[84] T. Menzies, Z. Chen, J. Hihn, and K. Lum, "Selecting best practices for effort estimation," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 883–895, 2006.

[85] K. Strike, K. El Emam, and N. Madhavji, "Software cost estimation with incomplete data," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 890–908, 2001.

[86] H. Park and S. Baek, "An empirical validation of a neural network model for software effort estimation," *Expert Systems with Applications*, vol. 35, no. 3, pp. 929–937, 2008.

[87] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Transactions on Software Engineering*, vol. 21, no. 2, pp. 126–137, 1995.

[88] S.-J. Huang, C.-Y. Lin, N.-H. Chiu, *et al.*, "Fuzzy decision tree approach for embedding risk assessment information into software cost estimation model," *Journal of Information Science and Engineering*, vol. 22, no. 2, pp. 297–313, 2006.

[89] B. Baskeles, B. Turhan, and A. Bener, "Software effort estimation using machine learning methods," in *22nd International Symposium on Computer and Information Sciences*, pp. 1–6, IEEE, 2007.

[90] Y.-F. Li, M. Xie, and T. Goh, "A study of the non-linear adjustment for analogy based software cost estimation," *Empirical Software Engineering*, vol. 14, no. 6, pp. 603–643, 2009.

[91] S. Kumari and S. Pushkar, "Cuckoo search based hybrid models for improving the accuracy of software effort estimation," *Microsystem Technologies*, pp. 1–8.

[92] M. Hosni, A. Idri, and A. Abran, "Investigating heterogeneous ensembles with filter feature selection for software effort estimation," in *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, pp. 207–220, ACM, 2017.

[93] M. O. Elish, "Assessment of voting ensemble for estimating software development effort," in *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pp. 316–321, IEEE, 2013.

[94] S. Ezghari and A. Zahi, "Uncertainty management in software effort estimation using a consistent fuzzy analogy-based method," *Applied Soft Computing*, vol. 67, pp. 540–557, 2018.

[95] F. Sarro, F. Ferrucci, and C. Gravino, "Single and multi objective genetic programming for software development effort estimation," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp. 1221–1226, ACM, 2012.

[96] B. Sigweni, M. Shepperd, and T. Turchi, "Realistic assessment of software effort estimation models," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, p. 41, ACM, 2016.

[97] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, 2012.

[98] D. Azhar, P. Riddle, E. Mendes, N. Mittas, and L. Angelis, "Using ensembles for web effort estimation," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 173–182, IEEE, 2013.

[99] P. Jodpimai, P. Sophatsathit, and C. Lursinsap, "Re-estimating software effort using prior phase efforts and data mining techniques," *Innovations in Systems and Software Engineering*, pp. 1–20, 2018.

[100] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris, "Software productivity and effort prediction with ordinal regression," *Information and Software Technology*, vol. 47, no. 1, pp. 17–29, 2005.

[101] M. Tsunoda, S. Amasaki, and A. Monden, "Handling categorical variables in effort estimation," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 99–102, ACM, 2012.

[102] M. Tsunoda, S. Amasaki, and C. Lokan, "How to treat timing information for software effort estimation?," in *Proceedings of the 2013 International Conference on Software and System Process*, pp. 10–19, ACM, 2013.

[103] A. R. Gray, S. G. MacDonell, and M. J. Shepperd, "Factors systematically associated with errors in subjective estimates of software development effort: The stability of expert judgment," in *Sixth International Proceedings on Software Metrics Symposium*, pp. 216–227, IEEE, 1999.

[104] M. Cartwright, M. J. Shepperd, and Q. Song, "Dealing with missing software project data," in *Ninth International Proceedings on Software Metrics Symposium*, pp. 154–165, IEEE, 2003.

[105] P. Sentas and L. Angelis, "Categorical missing data imputation for software cost estimation by multinomial logistic regression," *Journal of Systems and Software*, vol. 79, no. 3, pp. 404–414, 2006.

[106] Q. Song, M. Shepperd, X. Chen, and J. Liu, "Can k-NN imputation improve the performance of C4.5 with small software project data sets? A comparative evaluation," *Journal of Systems and software*, vol. 81, no. 12, pp. 2361–2370, 2008.

[107] A. Idri, I. Abnane, and A. Abran, "Missing data techniques in analogy-based software development effort estimation," *Journal of Systems and Software*, vol. 117, pp. 595–611, 2016.

[108] M. P. Martens, "The use of structural equation modeling in counseling psychology research," *The Counseling Psychologist*, vol. 33, no. 3, pp. 269–298, 2005.

[109] S. J. Breckler, "Applications of covariance structure modeling in psychology: Cause for concern?," *Psychological Bulletin*, vol. 107, no. 2, p. 260, 1990.

[110] D. Kaplan, *Structural equation modeling: Foundations and extensions*, vol. 10. Sage, 2008.

[111] J. B. Grace, *Structural equation modeling and natural systems*. Cambridge University Press, 2006.

[112] B. Roelstraete, Y. Rosseel, *et al.*, "FIAR: an R package for analyzing functional integration in the brain," *Journal of Statistical Software*, vol. 44, no. 13, pp. 1–32, 2011.

[113] X. Li, Y. Zhang, F. Guo, X. Gao, and Y. Wang, "Predicting the effect of land use and climate change on stream macroinvertebrates based on the linkage between structural equation modeling and bayesian network," *Ecological Indicators*, vol. 85, pp. 820–831, 2018.

[114] L. Lee, S. Petter, D. Fayard, and S. Robinson, "On the use of partial least squares path modeling in accounting research," *International Journal of Accounting Information Systems*, vol. 12, no. 4, pp. 305–328, 2011.

[115] J. Henseler, C. M. Ringle, and R. R. Sinkovics, "The use of partial least squares path modeling in international marketing," in *New Challenges to International Marketing*, pp. 277–319, Emerald Group Publishing Limited, 2009.

[116] J. F. Hair, M. Sarstedt, C. M. Ringle, and J. A. Mena, "An assessment of the use of partial least squares structural equation modeling in marketing research," *Journal of the Academy of Marketing Science*, vol. 40, no. 3, pp. 414–433, 2012.

[117] F. Ali, S. M. Rasoolimanesh, M. Sarstedt, C. M. Ringle, and K. Ryu, "An assessment of the use of partial least squares structural equation modeling (pls-sem) in hospitality research," *International Journal of Contemporary Hospitality Management*, vol. 30, no. 1, pp. 514–538, 2018.

[118] D. X. Peng and F. Lai, "Using partial least squares in operations management research: A practical guideline and summary of past research," *Journal of Operations Management*, vol. 30, no. 6, pp. 467–480, 2012.

[119] C. M. Ringle, M. Sarstedt, and D. Straub, "A critical look at the use of PLS-SEM in MIS quarterly," 2012.

[120] J. F. Hair, M. Sarstedt, T. M. Pieper, and C. M. Ringle, "The use of partial least squares structural equation modeling in strategic management research: A review of past practices and recommendations for future applications," *Long Range Planning*, vol. 45, no. 5-6, pp. 320–340, 2012.

[121] J. Jaklič, T. Grublješič, and A. Popovič, "The role of compatibility in predicting business intelligence and analytics use intentions," *International Journal of Information Management*, vol. 43, pp. 305–318, 2018.

[122] L. Kaufmann and J. Gaeckler, "A structured review of partial least squares in supply chain management research," *Journal of Purchasing and Supply Management*, vol. 21, no. 4, pp. 259–272, 2015.

[123] K. Kulikowski, "The model of relationships between pay for individual performance and work engagement," *Career Development International*, 2018.

[124] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on software engineering*, vol. 25, no. 5, pp. 675–689, 1999.

[125] N. Fenton, P. Krause, and M. Neil, "Software measurement: Uncertainty and causal modeling," *IEEE Software*, vol. 19, no. 4, pp. 116–122, 2002.

[126] N. Fenton, M. Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause, and R. Mishra, "Predicting software defects in varying development lifecycles using bayesian nets," *Information and Software Technology*, vol. 49, no. 1, pp. 32–43, 2007.

[127] G. J. Pai and J. B. Dugan, "Empirical analysis of software fault content and fault proneness using bayesian methods," *IEEE Transactions on software Engineering*, vol. 33, no. 10, pp. 675–686, 2007.

[128] N. Fenton, M. Neil, and D. Marquez, "Using bayesian networks to predict software defects and reliability," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 222, no. 4, pp. 701–712, 2008.

[129] C. Van Koten and A. Gray, "An application of bayesian network for predicting object-oriented software maintainability," *Information and Software Technology*, vol. 48, no. 1, pp. 59–67, 2006.

[130] S. Wagner, "A bayesian network approach to assess and predict software quality using activity-based quality models," *Information and Software Technology*, vol. 52, no. 11, pp. 1230–1241, 2010.

[131] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J.-F. Girard, "An activity-based quality model for maintainability," in *IEEE International Conference on Software Maintenance*, pp. 184–193, IEEE, 2007.

[132] S. Gupta and H. W. Kim, "Linking structural equation modeling to bayesian networks: Decision support for customer retention in virtual communities," *European Journal of Operational Research*, vol. 190, no. 3, pp. 818–833, 2008.

[133] R. Scheines, H. Hoijtink, and A. Boomsma, "Bayesian estimation and testing of structural equation models," *Psychometrika*, vol. 64, no. 1, pp. 37–52, 1999.

[134] A. Y.-L. Chong, "A two-staged SEM-neural network approach for understanding and predicting the determinants of m-commerce adoption," *Expert Systems with Applications*, vol. 40, no. 4, pp. 1240–1247, 2013.

[135] A. Y.-L. Chong and R. Bai, "Predicting open IOS adoption in SMEs: An integrated SEM-neural network approach," *Expert Systems with Applications*, vol. 41, no. 1, pp. 221–229, 2014.

[136] L.-Y. Leong, T.-S. Hew, V.-H. Lee, and K.-B. Ooi, "An SEM-artificial-neural-network analysis of the relationships between SERVPERF, customer satisfaction and loyalty among low-cost and full-service airline," *Expert Systems with Applications*, vol. 42, no. 19, pp. 6620–6634, 2015.

[137] I. Fisher and J. Ziviani, "Explanatory case studies: Implications and applications for clinical research," *Australian Occupational Therapy Journal*, vol. 51, no. 4, pp. 185–191, 2004.

[138] N. Paneth, E. Susser, and M. Susser, "Origins and early development of the case-control study: Part 2, the case-control study from Lane-Claypon to 1950," *Sozial-und Präventivmedizin*, vol. 47, no. 6, pp. 359–365, 2002.

[139] C. Mann, "Observational research methods. Research design II: Cohort, cross-sectional, and case-control studies," *Emergency Medicine Journal*, vol. 20, no. 1, pp. 54–60, 2003.

[140] G. Succi, W. Pedrycz, S. Djokic, P. Zuliani, and B. Russo, "An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite," *Empirical Software Engineering*, vol. 10, no. 1, pp. 81–104, 2005.

[141] S. Wacholder, D. T. Silverman, J. K. McLaughlin, and J. S. Mandel, "Selection of controls in case-control studies: II. types of controls," *American Journal of Epidemiology*, vol. 135, no. 9, pp. 1029–1041, 1992.

[142] L. S. Aiken, S. G. West, and R. R. Reno, *Multiple regression: Testing and interpreting interactions*. Sage, 1991.

[143] C. E. Lance, "Residual centering, exploratory and confirmatory moderator analysis, and decomposition of effects in path models containing interactions," *Applied Psychological Measurement*, vol. 12, no. 2, pp. 163–175, 1988.

[144] D. A. Belsley and E. Kuh, "Regression diagnostics: Identifying influential data and sources of collinearity," *Wiley Series in Probability and Mathematical Statistics*, 1980.

[145] D. G. Kleinbaum and M. Klein, *Logistic regression: A self-learning text*. Springer Science & Business Media, 2010.

[146] R. M. O'Brien, "A caution regarding rules of thumb for variance inflation factors," *Quality & Quantity*, vol. 41, no. 5, pp. 673–690, 2007.

[147] D. Hosmer and S. Lemeshow, *Applied logistic regression*. Wiley series in probability and mathematical statistics, 1989.

[148] D. G. Kleinbaum, L. L. Kupper, A. Nizam, and E. S. Rosenberg, *Applied regression analysis and other multivariable methods*. Nelson Education, 2013.

[149] J. Rosenberg, "Some misconceptions about lines of code," in *Proceedings of the 4th International Symposium on Software Metrics*, pp. 137–142, IEEE, 1997.

[150] T.-J. Yu, V. Y. Shen, and H. E. Dunsmore, "An analysis of several software defect models," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1261–1270, 1988.

[151] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*, pp. 78–88, 2009.

[152] R. W. Selby and V. R. Basili, "Analyzing error-prone system structure," *IEEE Transactions on Software Engineering*, vol. 17, no. 2, pp. 141–152, 1991.

[153] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *Journal of Systems and Software*, vol. 51, no. 3, pp. 245–273, 2000.

[154] Y. Shin, A. Meneely, L. Williams, J. Osborne, *et al.*, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, 2011.

[155] T. Illes-Seifert and B. Paech, "Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs," *Information and Software Technology*, vol. 52, no. 5, pp. 539–558, 2010.

[156] A. Ant, "The Apache Ant Project." `http://ant.apache.org/`, 2016. [Online; accessed 10-July-2016].

[157] D. project, "Apache Derby Project Charter." `https://db.apache.org/`, 2016. [Online; accessed 10-July-2016].

[158] X. Apache, "The Apache Xalan Project." `https://xalan.apache.org/`, 2016. [Online; accessed 10-July-2016].

[159] M. J. Ahmad, "A comparison of different learning approaches for software faults proneness prediction," *Qualifying Exam Paper, Lane Department of Computer Science and Electrical Engineering LCSEE, West Virginia University WVU*, 2017.

[160] D. Spinellis, "CKJM - a tool for calculating chidamber and kemerer java metrics," 2009.

[161] N. Serrano and I. Ciordia, "Ant: Automating the process of building applications," *IEEE software*, vol. 21, no. 6, p. 89, 2004.

[162] R. S. Wahono, "A systematic literature review of software defect prediction," *Journal of Software Engineering*, vol. 1, no. 1, 2015.

[163] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K.-i. Matsumoto, "The effects of over and under sampling on fault-prone module detection," in *International Symposium on Empirical Software Engineering and Measurement*, pp. 196–204, IEEE, 2007.

[164] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, vol. 1, pp. 137–144, IEEE, 2010.

[165] X. Yu, J. Liu, Z. Yang, X. Jia, Q. Ling, and S. Ye, "Learning from imbalanced data for predicting the number of software defects," in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pp. 78–89, IEEE, 2017.

[166] K. E. Bennin, J. Keung, A. Monden, P. Phannachitta, and S. Mensah, "The significant effects of data sampling approaches on software defect prioritization and classification," in *International Symposium on Empirical Software Engineering and Measurement*, pp. 364–373, IEEE Press, 2017.

[167] R. Shatnawi, "The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction," *Innovations in Systems and Software Engineering*, vol. 13, no. 2-3, pp. 201–217, 2017.

[168] A. Agrawal and T. Menzies, "Is better data better than better data miners?: on the benefits of tuning smote for defect prediction," in *Proceedings of the 40th International Conference on Software Engineering*, pp. 1050–1061, ACM, 2018.

[169] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

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

[171] A. Liaw and M. Wiener, "Classification and regression by random forest," *R News*, vol. 2, no. 3, pp. 18–22, 2002.

[172] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," *University of Waikato, Department of Computer Science*, 1998.

[173] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society*, pp. 267–288, 1996.

[174] L. Meier, S. Van De Geer, and P. Bühlmann, "The group lasso for logistic regression," *Journal of the Royal Statistical Society*, vol. 70, no. 1, pp. 53–71, 2008.

[175] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. Jan, pp. 1–30, 2006.

[176] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[177] K. E. Bennin, J. W. Keung, and A. Monden, "On the relative value of data resampling approaches for software defect prediction," *Empirical Software Engineering*, pp. 1–35, 2018.

[178] K. Molokken and M. Jorgensen, "A review of software surveys on software effort estimation," in *International Symposium on Empirical Software Engineering, 2003.*, pp. 223–230, IEEE, 2003.

[179] M. Van Genuchten, "Why is software late? An empirical study of reasons for delay in software development," *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 582–590, 1991.

[180] M. Usman, E. Mendes, and J. Börstler, "Effort estimation in agile software development: A survey on the state of the practice," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, p. 12, ACM, 2015.

[181] J. Zhang, E. Bloedorn, L. Rosen, and D. Venese, "Learning rules from highly unbalanced data sets," in *null*, pp. 571–574, IEEE, 2004.

[182] A. L. Lederer and J. Prasad, "Causes of inaccurate software development cost estimates," *Journal of Systems and Software*, vol. 31, no. 2, pp. 125–134, 1995.

[183] A. L. Lederer and J. Prasad, "Information systems software cost estimating: A current assessment," *Journal of Information Technology*, vol. 8, no. 1, pp. 22–33, 1993.

[184] R. Premraj, M. Shepperd, B. Kitchenham, and P. Forselius, "An empirical analysis of software productivity over time," in *11th IEEE International Symposium Software Metrics*, pp. 10–pp, IEEE, 2005.

[185] D. Yang, Q. Wang, M. Li, Y. Yang, K. Ye, and J. Du, "A survey on software cost estimation in the Chinese software industry," in *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 253–262, ACM, 2008.

[186] G. H. Subramanian and S. Breslawski, "An empirical analysis of software effort estimate alterations," *Journal of Systems and Software*, vol. 31, no. 2, pp. 135–141, 1995.

[187] R. Kerber, "Chimerge: Discretization of numeric attributes," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 123–128, Aaai Press, 1992.

[188] J. Hendrickx, B. Pelzer, M. Te Grotenhuis, and J. Lammers, "Collinearity involving ordered and unordered categorical variables," in *RC33 Conference in Amsterdam*, 2004.

[189] D. A. Belsley, E. Kuh, and R. E. Welsch, *Regression diagnostics: Identifying influential data and sources of collinearity*, vol. 571. John Wiley & Sons, 2005.

[190] F. González-Ladrón-de Guevara, M. Fernández-Diego, and C. Lokan, "The usage of ISBSG data fields in software effort estimation: A systematic mapping study," *Journal of Systems and Software*, vol. 113, pp. 188–215, 2016.

[191] D. Rodriguez, M. Sicilia, E. Garcia, and R. Harrison, "Empirical findings on team size and productivity in software development," *Journal of Systems and Software*, vol. 85, no. 3, pp. 562–570, 2012.

[192] D. Abbott, *Applied predictive analytics: Principles and techniques for the professional data analyst*. John Wiley & Sons, 2014.

[193] L. L. Minku and X. Yao, "Ensembles and locality: Insight on improving software effort estimation," *Information and Software Technology*, vol. 55, no. 8, pp. 1512–1528, 2013.

[194] Y. Kultur, B. Turhan, and A. Bener, "Ensemble of neural networks with associative memory (enna) for estimating software development costs," *Knowledge-Based Systems*, vol. 22, no. 6, pp. 395–402, 2009.

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

[196] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 11, pp. 736–743, 1997.

[197] G. E. Batista, M. C. Monard, *et al.*, "A study of k-nearest neighbour as an imputation method," *HIS*, vol. 87, no. 251-260, p. 48, 2002.

[198] J. Dougherty, R. Kohavi, M. Sahami, *et al.*, "Supervised and unsupervised discretization of continuous features," in *Machine Learning: Proceedings of the Twelfth International Conference*, vol. 12, pp. 194–202, 1995.

[199] M. Hamill and K. Goseva-Popstojanova, "Exploring fault types, detection activities, and failure severity in an evolving safety-critical software system," *Software Quality Journal*, vol. 23, no. 2, pp. 229–265, 2015.

[200] H. S. Jagpal, "Multicollinearity in structural equation models with unobservable variables," *Journal of Marketing Research*, pp. 431–439, 1982.

[201] S. R. Lipsitz, G. M. Fitzmaurice, and G. Molenberghs, "Goodness-of-fit tests for ordinal response regression models," *Applied Statistics*, pp. 175–190, 1996.

[202] E. Mendes, C. Lokan, R. Harrison, and C. Triggs, "A replicated comparison of cross-company and within-company effort estimation models using the isbsg database," in *11th IEEE International Symposium Software Metrics*, pp. 10–PP, IEEE, 2005.

[203] Ł. Radliński, "Predicting defect types in software projects," *Polish Journal of Environmental Studies*, vol. 18, no. 3B, pp. 311–315, 2009.

[204] J.-M. Desharnais, "Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction," *University of Montreal*, 1989.

[205] K. Dejaeger, W. Verbeke, D. Martens, and B. Baesens, "Data mining techniques for software effort estimation: A comparative study," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 375–397, 2012.

[206] R. Weston and P. A. Gore, "A brief guide to structural equation modeling," *The Counseling Psychologist*, vol. 34, no. 5, pp. 719–751, 2006.

[207] P. B. Lowry and J. Gaskin, "Partial least squares (PLS) structural equation modeling (SEM) for building and testing behavioral causal theory: When to choose it and how to use it," *IEEE transactions on professional communication*, vol. 57, no. 2, pp. 123–146, 2014.

[208] J. C. Anderson and D. W. Gerbing, "Structural equation modeling in practice: A review and recommended two-step approach.," *Psychological Bulletin*, vol. 103, no. 3, p. 411, 1988.

[209] J. F. Hair, C. M. Ringle, and M. Sarstedt, "Pls-sem: Indeed a silver bullet," *Journal of Marketing Theory and Practice*, vol. 19, no. 2, pp. 139–152, 2011.

[210] P. M. Bentler and C.-P. Chou, "Practical issues in structural modeling," *Sociological Methods & Research*, vol. 16, no. 1, pp. 78–117, 1987.

[211] G. R. Hancock and R. O. Mueller, *Structural equation modeling: A second course*. Information Age Publishing, Inc., 2013.

[212] D. Mindrila, "Maximum likelihood (ML) and diagonally weighted least squares (DWLS) estimation procedures: A comparison of estimation bias with ordinal and multivariate non-normal data," *International Journal of Digital Society*, vol. 1, no. 1, pp. 60–66, 2010.

[213] C.-P. Chou and P. M. Bentler, "Estimates and tests in structural equation modeling," *American Psychology Association*, pp. 37–55, 1995.

[214] P. J. Curran, S. G. West, and J. F. Finch, "The robustness of test statistics to nonnormality and specification error in confirmatory factor analysis," *Psychological Methods*, vol. 1, no. 1, p. 16, 1996.

[215] J. B. Ullman, "Structural equation modeling: Reviewing the basics and moving forward," *Journal of Personality Assessment*, vol. 87, no. 1, pp. 35–50, 2006.

[216] S. Shaphiro and M. Wilk, "An analysis of variance test for normality," *Biometrika*, vol. 52, no. 3, pp. 591–611, 1965.

[217] P. Royston, "Approximating the shapiro-wilk w-test for non-normality," *Statistics and Computing*, vol. 2, no. 3, pp. 117–119, 1992.

[218] H. Baumgartner and C. Homburg, "Applications of structural equation modeling in marketing and consumer research: A review," *International Journal of Research in Marketing*, vol. 13, no. 2, pp. 139–161, 1996.

[219] D. Barclay, C. Higgins, and R. Thompson, *The Partial Least Squares (pls) Approach to Casual Modeling: Personal Computer Adoption Ans Use as an Illustration*. 1995.

[220] S. Petter, D. Straub, and A. Rai, "Specifying formative constructs in information systems research," *MIS quarterly*, pp. 623–656, 2007.

[221] W. W. Chin, "The partial least squares approach to structural equation modeling," *Modern methods for business research*, vol. 295, no. 2, pp. 295–336, 1998.

[222] R. Grewal, J. A. Cote, and H. Baumgartner, "Multicollinearity and measurement error in structural equation models: Implications for theory testing," *Marketing Science*, vol. 23, no. 4, pp. 519–529, 2004.

[223] D. Gefen, D. Straub, and M.-C. Boudreau, "Structural equation modeling and regression: Guidelines for research practice," *Communications of the Association for Information Systems*, vol. 4, no. 1, p. 7, 2000.

[224] A. Diamantopoulos and J. A. Siguaw, "Formative versus reflective indicators in organizational measure development: A comparison and empirical illustration," *British Journal of Management*, vol. 17, no. 4, pp. 263–282, 2006.

[225] K. A. Bollen, "A new incremental fit index for general structural equation models," *Sociological Methods & Research*, vol. 17, no. 3, pp. 303–316, 1989.

[226] A. Narayanan, "A review of eight software packages for structural equation modeling," *The American Statistician*, vol. 66, no. 2, pp. 129–138, 2012.

[227] M. W. Browne, "Asymptotically distribution-free methods for the analysis of covariance structures," *British Journal of Mathematical and Statistical Psychology*, vol. 37, no. 1, pp. 62–83, 1984.

[228] V. E. Vinzi, L. Trinchera, and S. Amato, "Pls path modeling: from foundations to recent developments and open issues for model assessment and improvement," in *Handbook of partial least squares*, pp. 47–82, Springer, 2010.

[229] J. Henseler, C. M. Ringle, and M. Sarstedt, "Using partial least squares path modeling in advertising research: basic concepts and recent issues," *Handbook of research on international advertising*, vol. 252, 2012.

[230] P. Andreev, T. Heart, H. Maoz, and N. Pliskin, "Validating formative partial least squares (pls) models: methodological review and empirical illustration," *ICIS 2009 Proceedings*, p. 193, 2009.

[231] T. A. S. Foundation, "Apache Project Maturity Model." `https://community.apache.org/apache-way/apache-project-maturity-model.html`, 2018. [Online; accessed 01-July-2018].

[232] E. Foundation, "Eclipse Development Process 2015." `https://www.eclipse.org/projects/dev_process/#4_2_Code_and_Releases`, 2018. [Online; accessed 01-June-2018].