CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Strategy for Cross-Project Defect Prediction Models in Industry

Master's thesis in Software Engineering

David Gustafsson and Erik Pihl

Department of Software Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018:NN

A Strategy for Cross-Project Defect Prediction Models in Industry

David Gustafsson and Erik Pihl

Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

A Strategy for Cross-Project Defect Prediction Models in Industry
David Gustafsson & Erik Pihl

Supervisor: Robert Feldt, Software Engineering
Advisor: Mattias Wasteby, CPAC Systems AB
Examiner: Eric Knauss, Software Engineering

Master's Thesis 2018:NN
Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology and University of Gothenburg

A Strategy for Cross-Project Models in Industry
David Gustafsson & Erik Pihl
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Defects are a costly concern within software development. A way to mitigate this is the use of cross-project defect prediction models that aid quality assurance by pinpointing defect-prone parts of the software.
We have studied the ability to incorporate this kind of prediction models at an automotive company by developing such models. During this development, we applied metrics and modelling techniques from research to create cross-project models that would fill the needs of the company.
We succeed in creating cross-project defect prediction models using data gathered from local data repositories.
Our findings suggest that change-level metrics are of interest in the automotive sector since they allow for quick feedback. Random forest seem to result in the best predictive performance when training models, though these findings have to be considered in relation to a low performance for our models in comparison to similar studies.
At the same time, we gathered interesting insight into which performance measurement is most valued by practitioners. These findings pose that high precision is requested by practitioners while a low recall is acceptable. This is in contrast to the view held by research within the field and should be considered in future studies.

Keywords: Defect prediction, Fault prediction, Cross–project, Automotive industry

Acknowledgements

We would like to thank everyone that assisted us with our thesis work, most notably our supervisor Robert Feldt and our advisor Mattias Wasteby.

David Gustafsson, Gothenburg, June 2018
Erik Pihl, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1

Introduction

It is a well-known truth that software defects can lead to costly losses [1]. The effort of quality assurance is, therefore, crucial to avoid such costs and to ensure that resulting software have sufficient quality. At the same time, the cost of providing such assurance has been estimated to be between 50% and 75% of the total development costs [2]. A solution to this "lose and lose situation" is the use of defect prediction models, that can estimate which parts of the software that are more prone to error. With the knowledge of the more error prone parts, it is possible for quality assurance to cover a significant amount of defects with fewer resources [3].

A particular instance of such defect prediction models are the cross-project defect prediction models that have a higher degree of applicability and value in a commercial setting [4]. The variety of cross-project defect models also provides a multitude of ways of detecting defects at different abstraction levels of the code [5]. However, the abundance makes it difficult for anyone working with quality assurance to choose and incorporate a model that fits their specific context. To address this issue there exists a need to develop a strategy that can guide industry practitioners in how to construct cross-project defect prediction models which are both rigorous to research and applicable to their environment.

To understand how this strategy should be formed it is necessary to study the development of a cross-project defect prediction model that meets the criterion of diligence and relevancy. The pursuit of these criteria in a development process will highlight the areas where these criteria conflict, which will enable us to study applicability and usefulness of practices from research. This overall process will, therefore, make it possible to give an answer to questions regarding what metrics, modelling techniques and trade-offs from research that are usable and relevant to the automotive industry.

To create an object to study, we implemented a cross-project defect prediction model at CPAC, a company in the automotive sector. A cross-project approach was chosen to be able to transfer knowledge from historical data to upcoming projects. The requirements of the model were to be set by the quality management while we planned and undertook development and evaluation of the model. The evaluation of the model is split into a quantitative and qualitative part. The quantitative part seeks to understand the model performance by studying evaluation metrics

commonly used in the research field of defect prediction. The qualitative part will try to understand how the model will succeed in its objective by studying how well its result can contribute to the current development process. The qualitative part will be through interviews and questionnaires. The quantitative and qualitative data will be used to evaluate implemented models and practices.

This study's main contribution to the field of research is testing the applicability of certain reached standards, assumptions and know-how in an industrial setting. To industry practitioners, this study seeks to provide examples of how a defect prediction model can be implemented in a commercial setting, what constraints and decisions points can be found and what result can be expected.

The rest of this paper is split into six major sections: background, related works, method, results, discussion, and conclusion. The first chapter, background, will describe the context of the study and some definitions to help the reader. The second chapter, related work, will describe the existing research and studies within the area of defect prediction and why there is a need for answers to this study's research questions. The third chapter, method, will explain the process undertaken in the study. The fourth chapter, results, will describe more in detail the results of each iteration. The fifth chapter, discussion, is where answering the research questions will be handled. The final chapter, conclusion, will seek to summarise the preceding sections and the findings of the study.

1.1 Limitations

The scope of the study is limited due to being performed at an automotive company and having a focus on cross-project models.

As the study is done at an automotive company, it will be difficult to know if the same values and constraints explored within this study will apply to the software development in other parts of the industry. For this reason, our research questions will be framed around the automotive sector.

The focus on cross-project models will restrict the way that models will be trained and validated. This includes primarily using metrics and modelling techniques that have shown promising results in prior cross-project defect prediction research.

2

Background

This chapter will describe the background of both the industrial setting in which the study was conducted in as well as the information required to understand concepts in the field of defect prediction. Section 2.1 aims to explain the industrial setting of the study. Section 2.2 provides short descriptions of various concepts used in cross-project defect prediction. Lastly, Section 2.3 contains short explanations of terms and abbreviations used in the study.

2.1 The Industrial Setting

This study was done at the automotive company CPAC Systems AB. The following subsections aim to describe the industrial setting that the study was done in order to clarify the environment to the reader.

2.1.1 Organisation

CPAC is a company that develops electronic control systems for marine and commercial vehicles. They are certified for the quality standard ISO9001 and the environmental management systems standard ISO14001. Development at CPAC is split into the following segments: Industry, Construction, and Marine. These segments have their own projects, products, and target markets.

Developers are in general working within one of these segments but are also encouraged to migrate between segments in an effort to share knowledge. Developers are also flexible in how they approach their work and can initiate their own projects.

2.1.2 Products and Projects

Many of the products developed at CPAC are safety critical due to being used in vehicles. A defect in a vehicle part can, therefore, lead to a costly recall of vehicles.

This creates a great incentive to detect issues and defects before the release of a product.

Development projects at CPAC deal with one or more of these products and the same product can often appear in multiple development projects. The latter leads to projects sharing code.

In this study, projects within the industry and marine segments are studied since they are primarily embedded systems. Many of these projects deal with products already on the market with several releases. The products of these projects are mainly developed in C and many follow the MISRA guidelines. The size of the products is in the range of 50k-500k lines of code.

2.1.3 Development process

The development process is influenced by the agile principles, especially people over processes. A scrum-like framework is adopted for development tasks with weekly meetings. Several other agile tools are also loosely adapted to fit the work-flow. An example of this is the use of a Kanban board without a limiting factor for each stage.

Quality assurance within CPAC development is done in several stages. The first check is code reviews for every change by at least one other engineer. Then there are multiple stages of testing before the code is released to customers. Other quality assurance techniques are used within some of the projects, these ranges from ensuring traceability between requirements and implementation to static code analysis.

2.1.4 Tools and Data repositories

Development is done with the assistance of several tools. This section describes the tools that mainly influenced our thesis work.

The issue tracking system (ITS) used at CPAC is used for tracking both defects and tasks throughout the milestones of the projects. For the studied projects the ITS used is Trac. Trac is used to coordinate work on feature development, reviews and issues for the projects. For this purpose, it also contains references to changes in the version control system.

There exist multiple types of defects in the ITS. They can be caused by a failure of the code to execute correctly or failure to meet requirements. However, labelling for the different types of defects are non-existent. This creates a potential error where defects which are caused by a failure in the requirement elicitation process are bundled with defects due to software issues.

Issue tracking systems have previously been used to identify defective parts of the code by linking defect tickets to changes in version control systems [6].

To keep track of changes to files version control systems (VCS) are used. The study is limited to a Subversion repository since Trac is only used together with Subversion. CPACs Subversion repository contains both the code of products as well as related project documentation. VCS has previously been a source of data for defect prediction modelling [7, 8].

A standardised architectural framework for developing automotive applications is used by CPAC. To simplify compliance with the architecture and speed up development, code for a base platform can be generated by a tool. This code generating tool is used by many of the studied products. When making changes to the platform all existing generated files are changed.

Static code analysis is used within CPAC to measure code quality within projects. Code inspection tools are used in the projects to investigate source code complexity and compliance with software development guidelines. Several of the studied projects have constraints on the design of the code according to source code metrics. These constraints relate to the following categories: Lines of code, depth, statements per function and number of branches. The existence of these constraints on the code is a factor that needs to be taken into account when selecting metrics. The reason for this is that they might not offer enough variance to be useful for correlating to defects.

2.2 Theoretical background

This section gives a brief description of the theoretical concepts that are relevant to the field of cross-project defect prediction and will later be used throughout the report.

2.2.1 Metrics

In the field of defect prediction, there are multiple code-related measurements that have been used as predictors for defects. In this study, a standard of measurement that will be used to predict defect is defined as a "metric" while the data of such a metrics will be defined as a "measurement". These metrics are applicable to one or several abstraction levels of the code, such as function, subsystem or file. In this study, the level of abstraction that the metrics are applicable to will be referred to as the granularity level of the metric. Some metrics, such as McCabe's cyclomatic complexity, can be measured on several different levels of granularity. In contrast, other metrics such as the entropy metric proposed by Hassan [9] is only measurable on a single granularity level.

2.2.2 Learners

Learners are software systems used for machine learning. These systems can learn to make predictions using data. In the field of defect prediction learners uses gathered measurements to predict the defective status of the code. In this study classifiers, which can divide the input into classes, are used. When a learner has been taught we refer to it as a model.

A sub-field of machine learning is transfer learning where knowledge learned in one context is used in another. An example of this is a model trained with data from one project and used as a predictor for another project, then the knowledge from the first project has been transferred to the second project.

Models can be evaluated in multiple ways. Explanatory power and predictive power are two common ways of reasoning about the usefulness of a model. Explanatory power is how well a user of a model can understand the relationship between the input and output of the model. For instance, logistic regression models have high explanatory power since it is possible to investigate the size and direction of the effect that the input has on the classification. Predictive power instead describes to what extent the model can be expected to perform accurate predictions, this can be measured in several different ways which are further described in 2.2.3.

2.2.3 Performance metrics

When evaluating the performance of a defect prediction model three different performance metrics often appear in research: recall, precision and AUC.

Recall as used in the area of defect prediction describe how many defective instances that are detected in relation to all defective instances. It is computed using the formula seen in Equation 2.1. A high recall means that the model is able to successfully classify a majority of the defects while a low recall means that most of the defects are not classified as such by the model.

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2.1}$$

Precision in defect prediction is used to determine how often the model is correct when it classifies instances as defective. It is computed using the formula seen in Equation 2.2. A high precision value will, therefore, increase confidence in what the model classifies as defectives truly are defective.

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2.2}$$

Both precision and recall are considered threshold dependent metrics as they are affected by the probability cutoff value. Classification models often have a probability

cutoff value that determines how sensitive they are to classify entities as positive. Increasing this value will generally increase precision and decrease recall and vice-versa if the value is lowered. The probability cutoff determines which likelihood value should correspond to a defect classification. For example, a probability cutoff of 0.5 leads to instances that give a prediction score of at least 0.5 being classified as defective. A high cutoff will, therefore, lead to a lower rate of defective classification while a low cutoff leads to a higher rate of defective classifications.

AUC (Area Under receiver operator characteristics Curve) differs from precision and recall as it is threshold independent. AUC is calculated by measuring the area under the curve of true positive rate and false positive rate as they change with the cutoff values. This area under the curve corresponds with the probability that a true positive will be ranked higher than a true negative. For example, does the model give a higher probability of being defective to a defective file than a non-defective file. AUC values range from 0 to 1 where 0 means all classifications are wrong, 0.5 corresponds with random classifications and 1 is perfect classifications. The drawback of AUC is that is hard to interpret when it comes to the actual performance of the model since a cutoff value needs to be established to classify the input. The benefit is that AUC can be used to compare the overall predictive performance of the model, without having to consider different cutoff values.

AUC is favoured in many studies over other performance metrics when comparing learners in defect prediction [10, 11, 12] as it, due to being threshold-independent, is less affected by class-imbalance and noise in the data sets used by a learner [13].

2.2.4 Class-imbalance

Class-imbalance is when data of the two classes that are to be classified by a model is unevenly divided among the two classes. An example is that for every item of class A there are 99 items of class B. Class-imbalance needs to be taken into account during modelling because it can have effects on the training of the model. An example of this is that a model with a class-imbalance of 1 to 99 would be correct 99% of the time if everything was classified as the most occurring class.

There are some techniques to change the class-imbalance by either adding data of the small class or removing data from the larger class. Adding new data can be done by resampling the existing data or by generating similar data. In the area of defect prediction, removal of class-imbalance have small effects on the AUC of the model while it can have huge effects on precision and recall [14].

2.3 Glossary

- SMOTE - A technique used to address class-imbalance issues in a data set.

- ANOVA - Statistical test to check for difference among 2 or more groups.
- Scott-Knott - A clustering algorithm based on ANOVAs.
- Spearman correlation coefficient - Used to analyse data for correlations.
- Collinearity - Metrics which contain the same information i.e. if one increases the other increases as well.
- Shapiro-Wilk test - Null hypothesis test to check if a sample is normally distributed.
- MISRA C - programming guidelines focused on security and safety.
- 100-repeated out-of-sample bootstrap - Model validation technique.
- GLM - Learner based on linear models.
- SVM - Learner based on kernel functions.
- knn - Learner using the K-nearest-neighbours to predict input.
- Naive Bayes - Learner using Bayes' theorem to make predictions.
- Random forest - Learner using randomly generated decision trees.
- Feature branch - Temporary version tracker for developing a new feature.
- Master branch - Primary version tracker. Usually keeps track of most recent working version.

3

Related work

This chapter describes the previous work in the field defect prediction and how the research questions in this study sought to fill gaps present in the field. Section 3.1 will revolve around the usage of different learners and efforts as pre-processing techniques to predict defects. Following this, section 3.2 will describe the use of metrics in the field of the defect prediction. Section 3.3 aims to describe that state of cross-project defect prediction in research. Section 3.4 presents different views within the field of defect prediction on how precision and recall should be evaluated. Finally, section 3.5 will detail the research questions of the study.

3.1 Learners used for defect prediction

Many learners have been proposed and tried in the area of defect prediction. Rathore and Kumar in their study from 2017 [15] present a division that classifies the models as being either statistical, supervised or unsupervised. Commonly used techniques within these categories are naive Bayes, logistical regression, support vector machine and decision trees. These models have their own benefits and drawbacks in regards to accuracy and their ability to explain variability in the data [16].

Simple modelling techniques such as linear regression, logistical regression and naive Bayes performed well in the early 2000 [12]. Similarly, logistical regression and naive Bayes are also commonly featured in the cross-project defection prediction models up to June 2015 [5]. In recent years, random forest models, a variant using multiple decision trees, have gained traction due to its ability to deal with uncertainty and noise in the data sets [16].

Other studies have explored the use of pre-processing as well as parameter tuning to improve the performance of models. Agrawal and Menzies showed that by pre-processing the data, some performance measurement can be improved regardless of how the modelling is done [17]. Tuning of the parameters of the model has also shown to increase the performance [18].

Our thesis aims to discover how well different modelling techniques apply to the context of an automotive company. How is a model expected to behave to be

considered usable by an industry practitioner. Another aspect of our work is testing the modelling techniques on a commercial data set, which is an area that needs further exploration [16].

3.2 Metrics used for defect prediction

Metrics in the field of software defect prediction is often divided into categories of traditional, object-oriented(OO) and processes metrics. Traditional metrics are the oldest in use and among them are size metrics and complexity metrics such as lines of code and cyclomatic complexity. These metrics are usually easy to gather due to their simplicity or widespread use, but their predictive power is weak according to Radjenovic et al [19].

Object-oriented metrics, such as coupling between object classes and depth of inheritance tree, are commonly used in defect studies [19] and there are several distinct metric suites that have been defined in research [20, 21, 22]. Jurezcko and Madeyski created a suite of OO metrics [23] that have been reused later in other studies [24, 25].

Process metrics use information about how the code is changed to predict where defects are located. Measuring the number of developers, the number of changes and the number of past faults are some of the metrics that have been tested [26]. Nagappan et al. use change bursts, consecutive changes to a file, to predict defects [7]. They achieved precision and recall values above 90% in predicting defects only using data that can be gathered from a version control system.

Metrics collected during test execution can also be used to predict defects. This has been explored by Herzig using data from the development process of Windows 8 [27]. The results of these metrics had better predictive power on binaries than on source code files. Some of the metrics in this study can be hard to reuse in other contexts since they are related to operating systems and Microsoft's development process.

Defect prediction can work on several levels of granularity, files and modules are common. Kamei et al. argue that files and binaries are too coarse-grained and done too late in the development cycle [28]. They suggest looking at changes instead to identify if they introduce new defects or not. Looking at changes allows for quick feedback to developers so that a fix can be made while they still have the change fresh in memory. The design decisions of old code can be hard to remember when looking back at an old change.

What we want to explore further is how easy it is to gather the metrics using available data. Some metrics might be more expensive to gather. In our context, return on investment is an important aspect and might affect what metrics are relevant to use.

3.3 Creating cross-project models

A challenge of creating a cross-project defect prediction model is that all the projects need to have the same measurement available [29]. A solution presented to this problem is Heterogeneous Defect Prediction(HDP), which uses statistical analysis to match different measurement across projects.

Another challenge is the possibility of encountering noisy measurements that have a correlation with defects in the training project but not in the target project. This has in part been tackled by recent studies which use either random forest models [16] or unsupervised learning [30].

Multiple techniques for transfer learning have been devised, some with contradicting results. Menzies et al. claim that the best technique is to use similar projects within the data set to teach a defect prediction model [24]. They also discourage the use of whole data set to make a prediction on a single project due to the complex nature of software engineering projects. A more recent study by Herbold et al. have questioned this and claim that global and local defect tools have similar overall prediction performance [31].

Another technique for transfer learning is using what Krishna et al. call Bellwether [32]. This technique tries to find a single project to use as training data for all the others. This is accomplished by training models on every project and check which of the models have the best performance.

Different techniques for transfer learning have been suggested, but it is not investigated how applicable the techniques are to an industry context. We want to explore the applicability of the different techniques of transfer learning.

3.4 Evaluating model performance

An issue that is less explored in the area of the defect prediction is how industry practitioners want models to perform. Zhang and Zhang argue that defect prediction models require both high recall and high precision [33]. Menzies et al. presented an alternative viewpoint [34] where they argue that precision is too unstable to assess the value of a model. Recall should instead be the primary factor to determine the usability of models. The latter of the two arguments were later used by Kamei et al. to argue for the usability of prediction models built with change-level metrics [28].

In their study of cross-project defect prediction models, Hosseini et al. observed that many studies favour high recall at the cost of precision [5]. They suggest that this trade-off should instead be changed in the favour of precision and cites an earlier study of Herzig and Naggapan, who argue that false-positives have a higher cost for practitioners than false negatives [35].

These argumentations suggest different guidelines when it comes to evaluating the model. However, it is unclear how well these guidelines match up with what practitioners in our study believe crucial for a model to be fit for usage.

3.5 Research questions

The research questions deal with the issue of implementing a cross-project defect prediction model in a company context. RQ1 asks what strategy can be used for such an implementation, while RQ2 and RQ3 deal with details of creating and adjusting such a model in the company context. Finally, RQ4 tries to understand what factors of the models' performance are most valued by the company.

RQ1: *What is an effective strategy for a company in the automotive sector to implement a cross-project defect prediction model?* As the thesis will revolve around the creation of such a model, it will be necessary to develop such a strategy in the study. This strategy can be reused later in the company as well as for other industry practitioners.

RQ2: *Which metrics from research are good predictors of defects for a company in the automotive sector?* There is a need in the automotive industry to understand which metrics have been studied by research and are relevant to their field, easily accessible using available data and have a significant impact on the performance of the defect prediction model.

RQ3: *Which cross-project models from research are useful for defect prediction for a company in the automotive sector?* Similar to RQ2, there is also a need to understand the applicability of available defect prediction models from research. Both the predictive power as well as the usability need to be considered.

RQ4: *What factors of a defect prediction model's performance affects its perceived value and usability within an automotive company?* It's imperative to understand what users of a defect prediction model find useful. Current research provides conflicting arguments to what can be considered valuable, but it is unknown what applies in our industrial setting.

4

Method

The method chapter describes the process that was used throughout the study to produce defect prediction models. Section 4.1 motivates and describes the structure of the iterative approach used in the study. Section 4.2, 4.3, 4.4 and 4.5 motivates and describes the phases of each iteration. Additionally, these sections also describe how the phases differed between iteration 1 and 2.

4.1 Iterations

An iterative approach to developing the defect prediction model was undertaken in the study. These iterations were split into the following phases: a goal phase, a data gathering phase, a modelling phase and lastly an evaluation phase.

The mission of the goal phase was to set an objective to be accomplished by the defect prediction model. Using this goal, relevant measurements were collected in the data gathering phase. These measurements were, in turn, used to build several defect prediction models in the modelling phase. Lastly, the performance of these models was evaluated in the evaluation phase.

The iterative approach was undertaken to account for uncertainties in the implementation of the defect prediction model and to easily be able to reiterate upon and build upon previous iterations. If for instance, an issue would occur during the data gathering stage that would make it difficult to continue with the set goal, it would be easy to return to the goal phase with additional knowledge. Additionally, if an iteration ended with an overall subpar outcome but had achieved a significant result in one of its phases that result could be reused in the next iteration.

The overall performance and effort required to perform this iterative approach will make it possible to give answers to RQ1.

4.2 The Goal phase

The objective of the first phase was to set a goal that was to be achieved by the defect prediction model. This goal had to reflect the needs of industry practitioners, to ensure that the cross-project defect prediction model would be of benefit. An alternative approach was considered, basing the goal on available data. This alternative approach had the benefit of saving time on data gathering. However, this was not possible due to the lack of historical measurement data at the company.

The goal phase was chosen as the starting point for the iterations as a defined goal could be used to support the decision of either including or excluding different metrics and learners. For instance, a goal defined as "determining what characterises a defective module for the purpose of evaluating how well current quality metrics reflect reality" will make it suitable to include metrics that can be gathered on a module level. Additionally, it will favour the use of learners that have high explanatory power since it is crucial to understand how the different metrics affect the learners' classification. As the goal will be used to argue for the inclusion and exclusion of both metrics and learners it will in some part influence the answer of both RQ2 and RQ3.

How well an iteration achieved the goal was evaluated at the end of the iteration. This evaluation also formed the basis towards the goal of the next iteration.

4.2.1 Iteration 1 - Goal

To define the goal for iteration 1 a meeting was held with a quality manager. In this meeting, viable options based on prior research was presented by the researchers and a discussion was held regarding what goal would be suitable.

The choice of establishing the goal using a single meeting was chosen over a more extensive elicitation process, as it was deemed essential to quickly proceed to the other phases. This decision was based on the need to quickly get feedback on the feasibility of the goal.

The goal of this iteration was set to create a model that could predict whether a commit would be bug-inducing or not. This was chosen as it was of interest to have a tool that would enable quick feedback when new code was introduced to the code base. This short feedback cycle was critical since it would allow for defects to be fixed earlier.

4.2.2 Iteration 2 - Goal

After evaluating the performance of the defect prediction models produced in iteration 1 the researchers determined that a significant effort was still required to make it suitable for use. For this reason, the goal of iteration 2 was to improve the performance of iteration 1 defect prediction model and further investigate what performance was requested.

4.3 Data gathering phase

In the data gathering phase of an iteration, we sought to gather the data required for the modelling. A crucial aspect of the phase was to extract defects and measurements that would be in line with the goal that was set in the goal phase. This was done by translating the goal into a granularity level that the measurements and defects should be gathered on. For example, a goal-seeking to understand "what files are most prone to suffer from defects" will set the granularity level to "file".

Due to the lack of local measurement data, there was a need to extract this data from local data repositories. An alternative approach of using transfer learning on publicly accessible data repositories was considered but later deemed unsuitable. This was due to it being difficult to determine to what extent the data was applicable to the local projects. This uncertainty would make it difficult to persuade the company that the defect prediction model would be useful in their development process.

The result, as well as the observations from the data gathering phase, would later be used as a discussion point towards RQ2, as it would provide a foundation of what measurement could easily be accessible using available company data.

4.3.1 Iteration 1 - Data

The data gathering phase in iteration 1 revolved around two tasks: determining which commits introduce defects and gathering measurements on all commits.

To be able to solve the first task a heuristic was used to determine defect inducing changes. A heuristic was chosen as the vast number of commits to ITS made it difficult and time-consuming to manually inspect each commit. This leads us to use the algorithm MA-SZZ that was proposed by da Costa et al. [6]. This algorithm searches through the change history of files connected to a bug fix and estimates where the code in need of fixing was added to the code base. The auto-generated platform files in the VCS were excluded in this search as changes to the platform however small would result in changes to all the generated files. These broad changes would often just change comments or non-executable parts of the files, which were

Table 4.1: Cost analysis of relevant metrics

Metric	Description	Cost
NS	Number of modified subsystem	1
NF	Number of modified files	1
Entropy	How the change is distributed across files	2
Message Length	The length of the commit message	1
Relative Churn	(Lines added+Lines deleted)/ LT	2
Relative LT	LT/NF	2
Age	The average time interval between the current change and last change	5
NDev	The number of distinct commiters that have contributed to the file	5
NbrChangeTypes	The number of unique syntactic expressions found in the commit	8
TotalChangeTypes	The total number of syntactic expressions found in the commit	8
ExpDev	The number of prior commits from the developer	5
SExp	The number of prior commits to the subsystem done by the developer	5
MinRevAndAuth	If the committer's commits are below 5% of total commits to subsystem AND if commit reviews are below 5% of total reviews for the subsystem	8
LT	The total number of lines in the commited files	2

not important to our study and could potentially result in huge quantities of low-quality data if included.

To be able to solve the second task of determining what measurements needed to be extracted in iteration 1, a cost analysis was performed for relevant defect prediction metrics from research. The relevance was determined by if the metrics had seen prior use in cross-project studies or in studies at the specified granularity level. Furthermore, the selection was also influenced by the ability to cover different aspects of the changes that were to be studied. For example, we included metrics that revolved around syntactic expressions, history of the changed files and the size of the commit. The cost of the metrics was determined by estimating how difficult it would be to extract them from the local data repositories.

- 1 - The measurement can be gathered by studying a single commit.

- 2 - The measurement can be gathered by studying the diff of a commit.
- 5 - The measurement can be gathered by studying the entire commit history.
- 8 - The measurement requires a checkout of the commit and/or the use of a third-party tool.

In table 4.1 are the metrics that were considered of interest. These are previously used in the construction of change level cross-project defect prediction models [36], with the exception of Message Length, syntactic metrics and MinRevAndAuth. ExpDev and SExp are not used in the cross-project study due to them not being available to gather in new projects [36]. Though other studies have previously used historic metrics in a cross-project context [37, 38]. They were included in this study since the projects in the study are sharing developers as well as products between them.

The reason for including Message Length was that it has been used in previous studies that have studied change level within-project defect prediction models [39, 40]. The reason for minRevAndAuth was due to it being present from a within-project replication study of code-ownership by Greiler et al. and could easily be applied to the granularity of a code change [41]. Finally syntactic expressions have been studied in a cross-project setting by Mizuno and Hirata [42].

Adjustments had to be made towards the metrics dealing with subsystem as it was difficult to use the definition of the subsystem used by Kamei et al. with the file structure present in the local VCS [28].

When extracting the measurements, commits that were not connected to the ITS were excluded. This was done since the commits with no connection to the ITS could not receive a defect inducing commit status since the MA-SZZ only used commits in the ITS. Consequently, the inclusion of such commits would not aid us as it would be unknown if these commits were defect inducing or not.

In the first iteration, only the metrics with cost 1 or 2 were gathered. The reason for this was problems when gathering defect inducing commits were encountered. Some of the links in the history were missing which caused complications with tracing commits on feature branches. This necessitated that we spent a considerable amount of time to fix the missing links which were done by looking at the commits linked to a ticket in the ITS.

Assumptions also had to be made when dividing the commits into projects. This was because the ITS did not keep track of projects. It instead divided all projects into milestones where the name of the milestone was derived from a project. This required us to group and convert the milestones into projects. Most of the grouping was automated but there were some special cases that were done by hand. Some ITS tickets were also discarded since they lacked a milestone or were connected to milestones that dealt with non-development efforts, such as QA standards or internal policy updates.

Some of the diffs in the VCS were too large to be put into memory to be analysed in an easy fashion. Looking at a subset of these large commits all of them were copies of data from one directory to another. A choice was made to exclude all commits with diffs larger than 1M lines. This cutoff point was chosen since 1M lines were possible to work with and it seemed unlikely that a normal change would be of that size and thus be excluded.

4.3.2 Iteration 2 - Data

In iteration 2 two additional metrics were gathered, experience and subsystem experience, to test if prior unused measurements could affect the performance of the defect prediction model. Because of time-constraint experience and subsystem experience was considered over the other metrics in the same cost-category as they were easier to extract than Age and NDev, as the researchers had discovered gaps in VCS file history.

4.4 Modelling phase

In the modelling phase, the data gathered in the prior phase was used to train several cross-project defect prediction models. This enabled further inspection of the measurements and to later evaluate the class-imbalance issue, learners and transfer learning techniques in the evaluation phase.

The observation formed and decision done in the modelling phase was later used to support argumentation towards RQ2 and RQ3. RQ2 was further explored by investigating collinearity amongst the measurements. RQ3 would be explored in part by studying how investigating the need for class-imbalance techniques on the sample.

The transfer learning techniques that are to be used in the study are simple merge and bellwether. Both of the techniques are selected for their simplicity in comparison to clustering and ensemble models. If the simple techniques work there is a potential benefit of spending less time on the transfer learning.

4.4.1 Iteration 1

The first step was to check for collinearity amongst the gathered measurements, this is crucial because some learners perform poorly on inputs that are highly collinear. A Spearman correlation value of 0.7 was used as a threshold to identify highly correlating measurements, as it has been suggested by prior researchers [43].

To determine the correlation between the metrics a variable clustering analysis was

done. We used the R package Hmisc which provides a function VarClus as recommended by Has [43]. Correlating measurements were removed as some learners suffer when multicollinearity exist in the input.

Following collinearity analysis, the next step was to determine if class-imbalance would become a problem and if so what techniques would be used to handle it. The class-imbalance issue is common to defect prediction studies, as defects usually form a small minority of the sampled data. This creates an issue for many learners since they rely on the sample being evenly split between the predicted classes.

Since there was a class-imbalance of more than 20 to one in the gathered data set it was deemed necessary to apply a class-imbalance technique. SMOTE was selected for this task due to its good performance in previous studies [11]. Due to time constraints, the default settings for SMOTE was used for this iteration.

Following this step, a selection of common learners in research was taken and fitted with the sampled data. To see which learner worked best for this data, performance was measured by using all commits as a single project. The data was partitioned into a training set and validation set with a 70/30 split, a common split within machine learning. The caret package in R was used to create the models. It was selected since it provides a single interface for all the learners we wanted to use. Five learners, random forest, SVM, stepwise glm, knn and naive Bayes, were trained to see which performed best with this data set. 100-repeated out-of-sample bootstrap was used to train the models. 100-repeated out-of-sample bootstrap was selected as it had been recommended by a prior study that evaluated validation techniques [44].

4.4.2 Iteration 2

In iteration 2, the calibration of SMOTE was revised to further test to what extent the precision and recall could be manipulated to better fit what constitutes a good performance for the defect prediction models. To assess this, several calibrations were tested on the defect prediction model created in iteration 1. These calibrations were later used to argue for what precision and recall were possible and to what degree it was suitable to address the class-imbalance at all.

4.5 Performance evaluation phase

The evaluation phase had two objectives: Firstly, evaluating the effect of the measurements on the models. Secondly, how the defect prediction models performed in relation to the needs present at the company.

This phase will form the bulk of argument towards RQ2, RQ3 and RQ4. Reasoning

on RQ2 was based on the comparison of the different measurements in the best performing learner. Arguments for RQ3 were based on the performance reached with the class-imbalance techniques, learners and transfer learning techniques that make up the different cross-project defect prediction models. Arguments for RQ4 will be based on what the software developers believe is crucial for the defect prediction model to be successful.

4.5.1 Iteration 1 - Predictive performance

The first task of iteration 1 evaluation was to select the top performing learner that was created in the modelling phase. The selection of the top performing learners was done by comparing model precision, recall and AUC values.

Following this, the gathered measurements were evaluated by studying their variable importance in the best performing model, since it had provided the best fit for the data.

Thereafter a test was done to determine if simple merge technique or bellwether could function as a good transfer learning technique.

4.5.2 Iteration 1 - Interviews

Semi-structured interviews were conducted in the evaluation phase of iteration 1 to determine what would be required for the model to be used by the company. The length of the interviews varied between 20-40 minutes. The choice of applying a semi-structured approach was done as it was deemed necessary to enable the interviewer to ask follow-up questions and to touch on interesting areas that were not covered by the questions.

A junior developer, as well as a quality manager, were selected for the interview in iteration 1.

The interview design was split into 3 parts: an introduction that presented the goal of the study, a section of warm-up questions and finally the main section with questions about defect prediction and software quality. The introduction was made to explain the design and purpose of the interview, as well as provide guidelines on how to deal with difficulties in answering the questions. The warm-up contained some simple questions that allowed the respondent to ease into answering questions.

The analysis of interviews was done in three steps. Factor derivation, factor clustering and theme identification. Factor derivation was done since the interview answers were long and contained a lot of detailed information. To make the answers easier to analyse they were rephrased using a shorter more consistent phrase.

The phrases were then clustered based on similarities to be able to find common themes. These themes then form the result of the analysis and were used to formulate recommendations for creating defect prediction models and providing directions for our further work.

The resulting themes were later used to determine what needed to be done to calibrate the model to better suit the company.

4.5.3 Iteration 2 - Predictive performance

In iteration 2 the within-project performance of the best performing learner was evaluated to see if within models are better suited compared to the models of iteration 1.

To see if a within-project defect prediction model could be more useful five within-project models were constructed from the five projects with the most defects. 30 instances were created for each project model and the mean AUC was computed. To determine if there is a significant difference between within-project defect prediction models and cross-project defect prediction models ANOVAs were made.

4.5.4 Iteration 2 - Interviews & Survey

A questionnaire was used in the evaluation phase of iteration 2 to study RQ4 on a broader scale than prior interviewees could. The questionnaire was sent out to 79 recipients at the company.

The response rate was measured in order to assess to what degree the result of the questionnaire matched the opinions in the company.

Prior to the release of the questionnaire, it was tested on five software engineers. They were asked to review the questions and point out any question that was hard to understand. Their answers were not part of the final survey.

The questionnaire contained both quantitative and qualitative answers. Quantitative answers can be interpreted as is while qualitative answers need to be coded into categories. The distribution among the categories is then used as the basis for the analysis of the responses to these questions.

Iteration 2 also included an interview using the same design as described in 4.5.2 but with another protocol to better complement the result of the survey. A senior developer was selected for this interview.

The analysis of the iteration 2 interview was added to the survey and the result of the prior interviews provide arguments for RQ4.

5

Results

This chapter describes the results of each iteration, it is split into two sections: section 5.1 which describes the result of iteration 1 and section 5.2 which describes the result of iteration 2.

5.1 Iteration 1

This section describes the results gathered for the modelling phase and evaluation phase of iteration 1. It is split into the following subsections:

- Subsection 5.1.1 describes the data set gathered in iteration 1 and the correlation of the measurements within them.
- Subsection 5.1.2 describes the result of comparison of the different learners and the selection of the best performing one.
- Subsection 5.1.3 describes importance of the measurements.
- Subsection 5.1.4 describes the result of the transfer learning techniques.
- Subsection 5.1.5 describes the result of the interviews that were held to evaluate the models.

5.1.1 Data set

Using the method described in iteration 1, a data set of 20924 commits were gathered, these commits belonged to 48 projects of varying sizes. The measurements that were gathered are the following, LT, LA, LD, NS, NF, Entropy, Relative LT, Relative Churn, Message Length.

Two pairs of correlating measurements were found, as can be seen in figure 5.1. Since some of the learners were sensitive to correlating measurements, Entropy and Relative Churn were not used when the learners where compared. Entropy and Relative Churn were excluded in favour of NF and LD due to the latter being easier to gather from a commit.

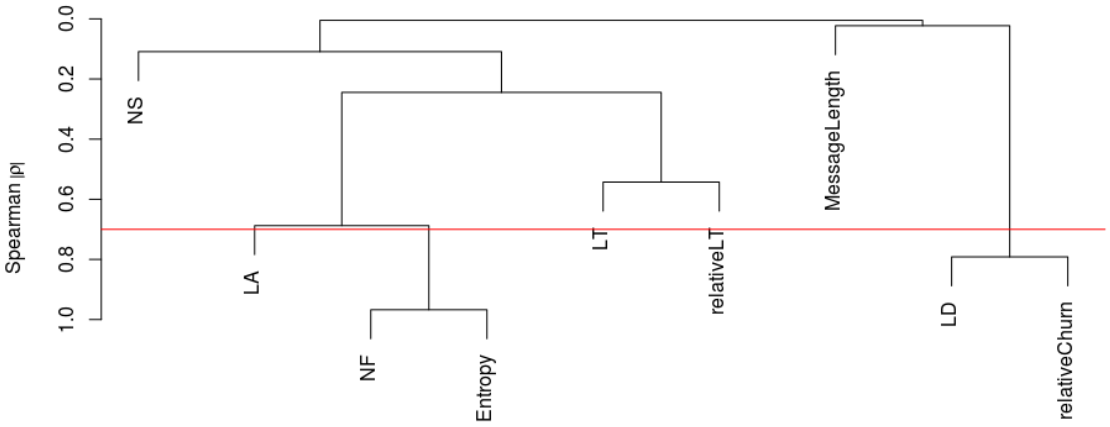

Figure 5.1: The result of the Spearman correlation test showing that there are two pairs of highly correlating measurements in the data set of iteration 1. NF, Entropy, LD and Relative Churn have correlation values above 0.7 which is signified by the horizontal line in the figure.

5.1.2 Learners

To compare the performance of five learners, random forest, knn, SVM, stepwise GLM, and naive Bayes, were trained using the entire data set as a single project. Figure 5.2 shows how the five models perform. Naive Bayes performs poorly when looking at recall values, where the mean is very close to zero while the others show a more promising performance. Random forest has the highest score when looking at the AUC values even if the four best performing models are rather equal. Since random forest performs best a decision was made to further evaluate the random forest learner. Since random forest is less affected by multicollinearity Entropy and Relative Churn was included in the metric selection again.

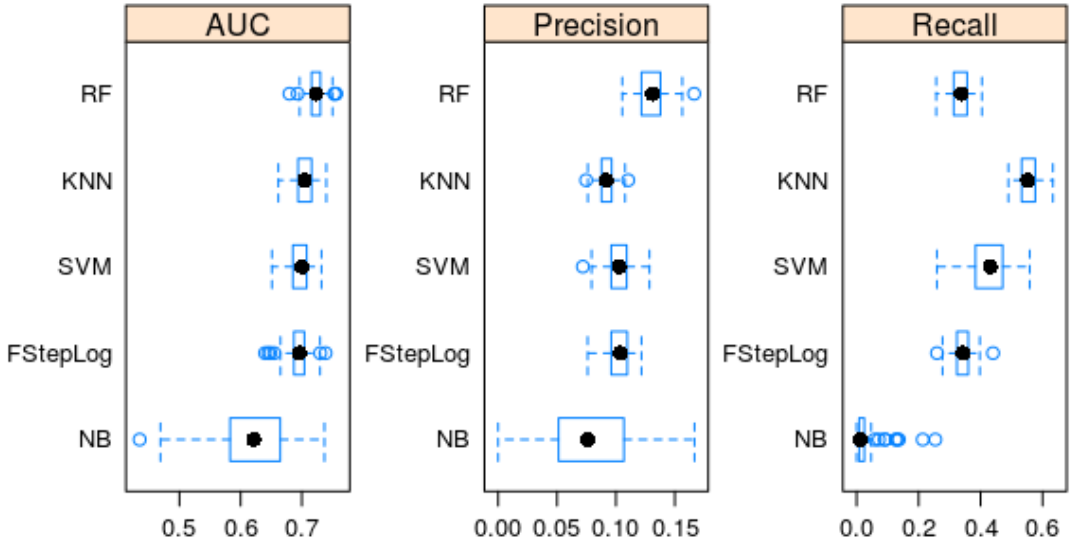

Figure 5.2: Boxplots of the performance of the five learners on our data set using precision, recall and AUC. Precision and recall are computed using the default threshold of 0.5.

5.1.3 Measurements in learners

A variable importance analysis was done on the random forest model to understand which input measurement had the most impact on the model. Due to how random forest works it is hard to determine the size of the differences in the importance of the variables, but the ordering can give an indication of which variable is most important for the accuracy of the model. It shows that the code size related measurements are the most important for the models' accuracy while counting the number of files or subsystems is of less importance. The variables were ordered in the following way:

1. LT
2. LD
3. LA
4. Relative LT
5. Relative Churn
6. Message Length
7. Entropy
8. NF
9. NS

5.1.4 Transfer learning techniques

The result of using simple merge to build models for five of the projects can be seen in table 5.1. The AUC values are the mean of 30 models trained for each project.

Table 5.1: AUC values for 5 models created using simple merge.

P1	P2	P3	P4	P5
0.67	0.65	0.61	0.66	0.59

Table 5.2 shows the AUC values when using the bellwether technique for training models. The bellwether test was done using the same five projects as in the simple merge. 40 random forests were generated using every project as training data. The highlighted cells are the best training projects for each project, the AUC values are clustered using Scott-Knott to determine statistical significance in their differences. There is no single project that generates a better predictor for all other projects, a fully highlighted column, in contrast to the result of Krishna et al. [32].

Table 5.2: Bellweather performance of random forest models. The highlighted cells signifies the best result for each of the projects.

Validation\Training	P1	P2	P3	P4	P5
P1	-	0.67	0.67	0.68	0.58
P2	0.64	-	0.64	0.64	0.54
P3	0.62	0.68	-	0.66	0.59
P4	0.74	0.64	0.72	-	0.59
P5	0.56	0.60	0.56	0.58	-

5.1.5 Interviews

The analysis of the interviews resulted in a series of common themes that can be seen in table 5.3. These themes present different views on both what results are of crucial concern and what role the models might play in the current development process.

Table 5.3: Interview themes of iteration 1

ID	Description
1	Developers will likely ignore the model if it has too low precision.
2	The perceived value of recall is influenced by the release frequency of the projects.
3	The defect prediction model should be an additional source of information that aids developers when determining code quality.
4	The primary user of the defect prediction model is the developer who wants to know if his or her commit is more likely to introduce defects.
5	Developers want to understand what factors contribute to a change being deemed risky by the defect prediction model and why a particular change is deemed such.
6	There are multiple reasons to investigate the code, finding defects is one of them.

ID1: This warning has previously been mentioned in a cross-project context [5, 45], and was one of the prime concerns that interviewees had concerning the ability to use the model. The factors within this theme also involved the concept of "trust" in relation to the precision which also opens up the avenue for other factors affecting trust to be valued by the model.

ID2: This theme poses an additional insight into what influences the perceived value of recall. The theme was based on a single factor which expressed that it was less important that defects were detected by the model if updates could be delivered quickly to the end user.

ID3: This theme describes the role of prediction in relation to overall quality of software and describes the prediction as an information source. This theme also shows a interesting idea in that the prediction model does not act in isolation, but in tandem with other tools. Many of the factors within this theme express the idea that the prediction should also be coupled with a judgement of the developers making the commit. Further on, they all stress that judgement of the developer should have a higher value than the prediction.

ID4 This theme expresses the notion that the primary use case of the model should be a developer that inspects their own commit for defects. Factors within this theme both discuss the importance of doing the prediction early on and that a developer should be responsible for fixing their own commits.

ID5 This theme shows an interest in understanding what affects the decision to classify a commit as defective or not. This suggests that the properties of an explanatory model could be of high value when making use of the prediction. The factors within this theme describe how the interviewees would act and feel when the prediction was made on their own or others code.

ID6 This theme describes that code-inspection effort overall is perceived to contribute to multiple different parts of the development and quality. This theme and ID3 express arguments against using the prediction to increase the focus of inspection efforts by reducing coverage. Factors within this theme describe both the importance of code-inspection to increase developer knowledge and the judgement posed by peers.

The interview protocol can be found in Appendix A.

5.2 Iteration 2

This section describes the results of the modelling phase and evaluation phase of iteration 2. It is split into the following subsections:

- Subsection 5.2.1 describes the within-project performance of the defect prediction models.
- Subsection 5.2.2 describes how the performance changed by including experience measurements.
- Subsection 5.2.3 describes the result of tuning SMOTE to achieve higher precision values.
- Subsection 5.2.4 describes the result of the interview held in iteration 2.
- Subsection 5.2.5 describes the result of the survey held in iteration 2.

5.2.1 Within-project defect prediction

The within-project defect prediction performed better than its cross-project counterpart, as seen in table 5.4. The table shows a comparison between 30 random forest models trained using simple merge and within-project data.

Shapiro-Wilk tests were used to check if the data is normally distributed. The resulting p-values were all above 0.05 and thus we cannot reject the hypothesis that the samples are normally distributed.

To test the significance of the differences between the results ANOVAs where done, as normality could not be rejected. The results from ANOVAs were all lower than 0.05 and proves the significance of the difference between the cross-project and within-project models. The AUC for the projects is between 7% and 19% lower in all the cross-project models.

Table 5.4: Comparison of within-project (WP) defect prediction models and cross-project (CP) defect prediction models.

Project	CP AUC	WP AUC	Pr(>F)	Shaprio-Wilk CP	Shapiro-Wilk WP
P1	0.67	0.72	4.06e-11	0.90	0.85
P2	0.65	0.69	1.25e-11	0.78	0.56
P3	0.61	0.75	<2e-16	0.97	0.43
P4	0.66	0.76	<2e-16	0.78	0.31
P5	0.59	0.64	2.36e-05	0.80	0.37

5.2.2 Experience measurements

A Spearman correlation test with the added experience measurement, as seen in figure 5.3, shows no additional correlations.

Adding the experience measurements to the data improved the AUC of the model, which can be seen in figure 5.4. A Shapiro-Wilk test on the two samples returns the p-values 0.46 and 0.16 for with and without experience models respectively. This means that we cannot disregard the possibility that the samples are normally distributed. To check for the difference between the two samples an ANOVA was used, it gave a p-value of <2e-16 which means that the difference is significant. To see how important the added metrics are in comparison to the other metrics a variable importance analysis was done. It shows that the newly added measurements contain the most information in the model. It orders the measurements in the following way:

1. Experience
2. Sub-system experience
3. LD
4. LT
5. LA
6. Message Length
7. Relative LT
8. Relative churn
9. Entropy
10. NF
11. NS

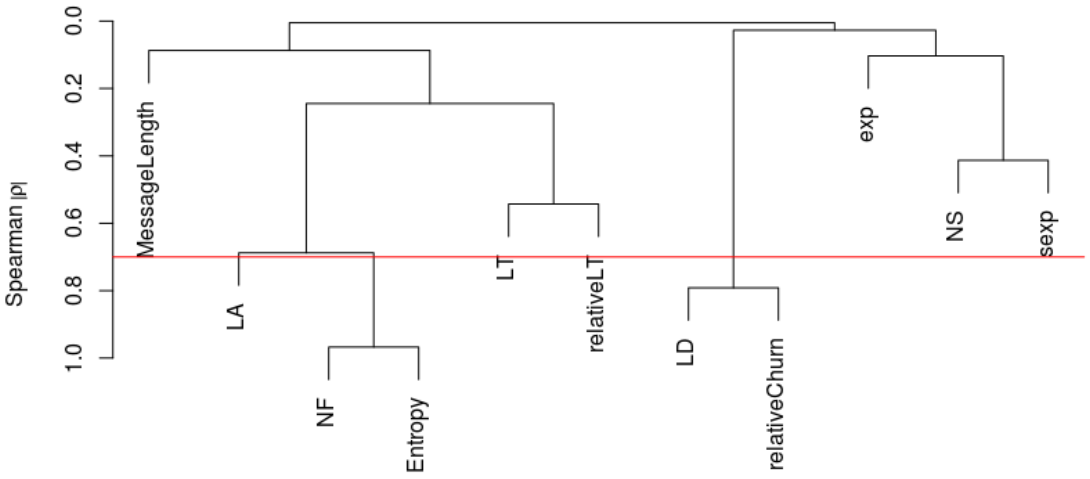

Figure 5.3: The result of a Spearman correlation test shows that the experience measurements are neither correlated with each other or the other measurements in the data set of iteration 2.

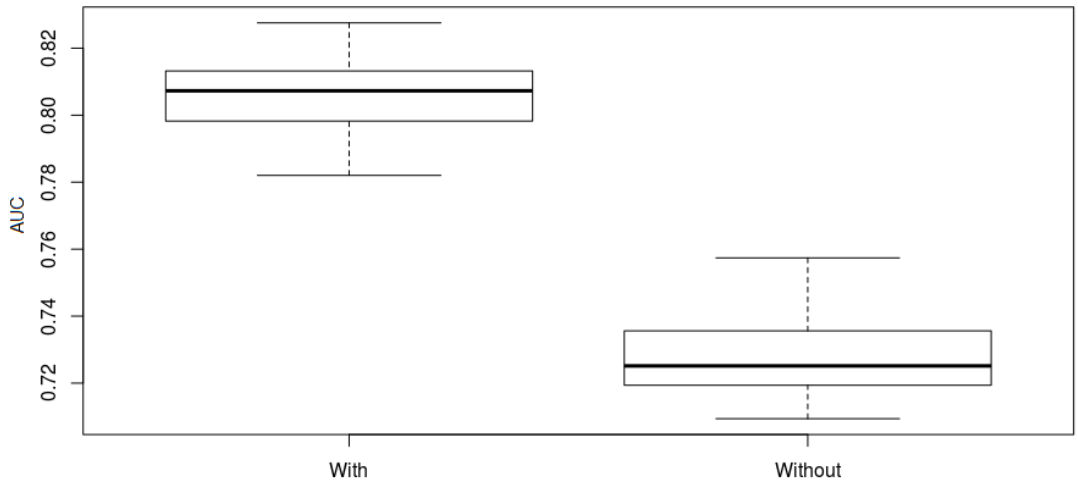

Figure 5.4: Boxplot of the difference in AUC of models trained with and without experience and sub-system experience data on the whole data set.

5.2.3 Tuning SMOTE

Experimentation with the parameters for SMOTE was done. The most promising configuration can be seen in figure 5.5. The precision is about twice as high as the default configuration at the threshold 0.5. A model was also trained without the use of SMOTE. The results of this can be seen in figure 5.6. This model has a higher precision at most threshold levels than the model trained with SMOTE, but the recall is lower.

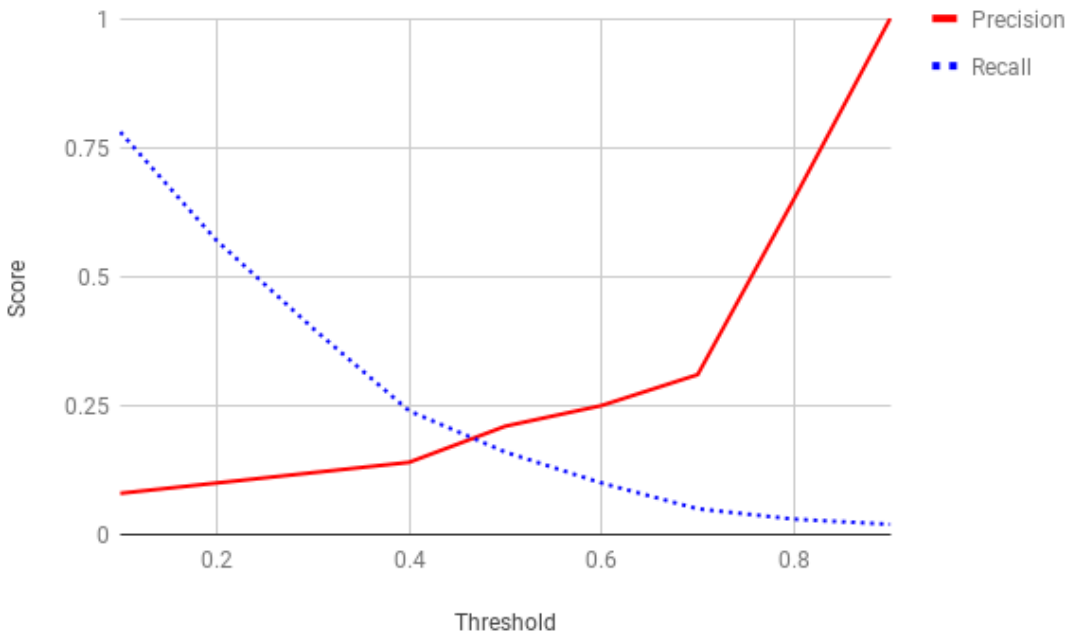

Figure 5.5: The SMOTE configuration with highest precision.

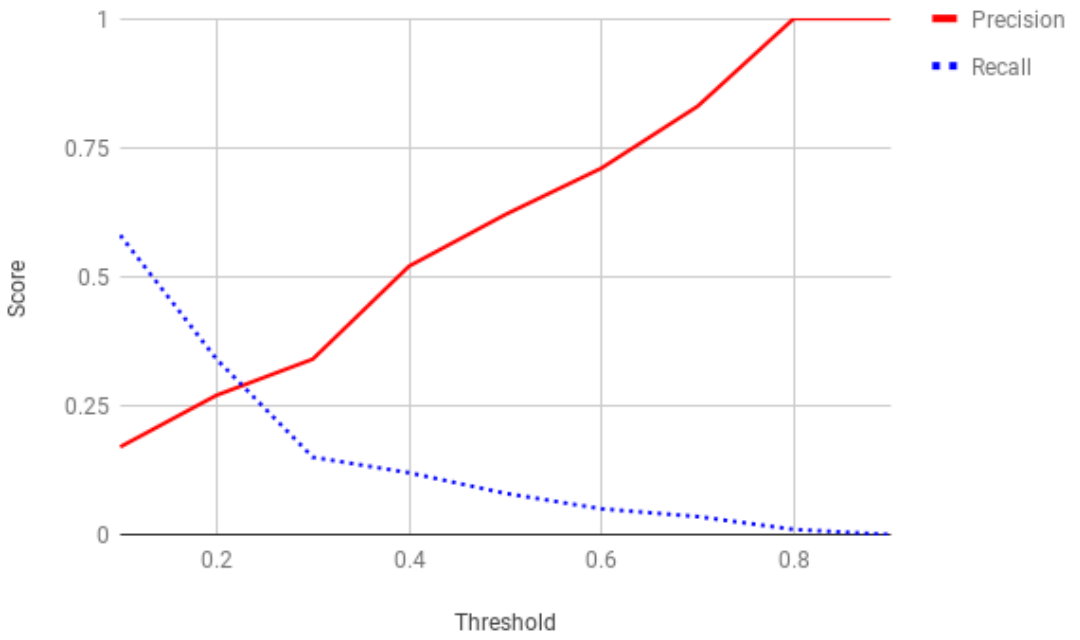

Figure 5.6: Precision and recall of a model trained on data that was not pre-processed with SMOTE.

5.2.4 Interview

A interview was held with a senior engineer. The result of the interview in iteration 2 can be seen in table 5.5.

ID	Description
7	Understandability is preferred since it is a base for action
8	High precision is requested since false positives are expensive and decreases trust in the predictions
9	Defect prediction models are just one more tool with the potential to catch defects.

Table 5.5: Interview themes of iteration 2.

ID7: This theme further shows that understandability is wanted for the defect prediction model as earlier expressed in ID5. Important to note is that factors within this theme express a want towards a more explanatory model but does not deem it crucial.

ID8: This theme connects with ID1 and gives further argumentation towards precision as a deal breaker when using the model. Though in contrast to ID1 this also connects the drawback to effort rather than just trust in the model.

ID9: This theme further contains one of the ideas present in ID3 and ID6 where

the prediction is used in tandem with other code-quality efforts. Factors within this theme express the notion that even a small amount of defects caught by the predictions are valuable since they contribute to overall quality of the system.

The identified themes are in line with the themes of the first round of interviews. All the interviewees seem to agree that precision is important (ID1 and ID7) and that the model is one of many tools for preventing defects (ID3 and ID8).

The interview protocol for iteration 2 can be found in Appendix B.

5.2.5 Survey

A survey containing 14 questions was sent out to personnel at the company. The overall response rate was poor as out of 79 recipients only 28 responded, this sets the response rate to 35%. The survey contained several questions that discussed warnings on commits, which was explained as a sign that a commit was likely to induce a defect.

The following questions of the survey will be presented in this subsection as they indicate a result that is of interest in relation to interview themes and have a high coverage amongst the respondents:

- Should a developer be allowed to ignore warnings when submitting a commit?
- Imagine the warnings only are shown for a small subset of all defect inducing commits. Would such warnings be useful?
- Would the warnings be useful if there was a high rate of false-alarms?
- Which of the following changes would make the warnings more useful?
- How important is it for you to understand what caused a particular commit to receive a warning?

The remaining responses and the structure of questionnaire can be seen in Appendix C.

In figure 5.7 we can see that respondent to the survey believed it important to not be restricted by the model, which hints towards the sentiment expressed in ID3 of 5.3.

Following this, we can see that figure 5.8 connects to ID1 and ID8 in that a high rate of false alarms would be perceived as a deal-breaker for respondents. Figure 5.9 in contrast shows that respondents can manage with a less extensive coverage of defect inducing commits.

In figure 5.10 it can be observed how the respondents would prefer to tune the model.

The majority favours higher precision at the cost of lower recall when tuning the model. This answer corresponds with a focus of high precision within the themes of both ID1 and ID8. Further on, this result also corresponds with the sentiment expressed in the responses shown in figure 5.9 and 5.8.

The final figure shows that respondents want to know more than the defective status as seen in figure 5.11. This is similar to ID5 and ID8 that also stresses the importance of understandability within the models.

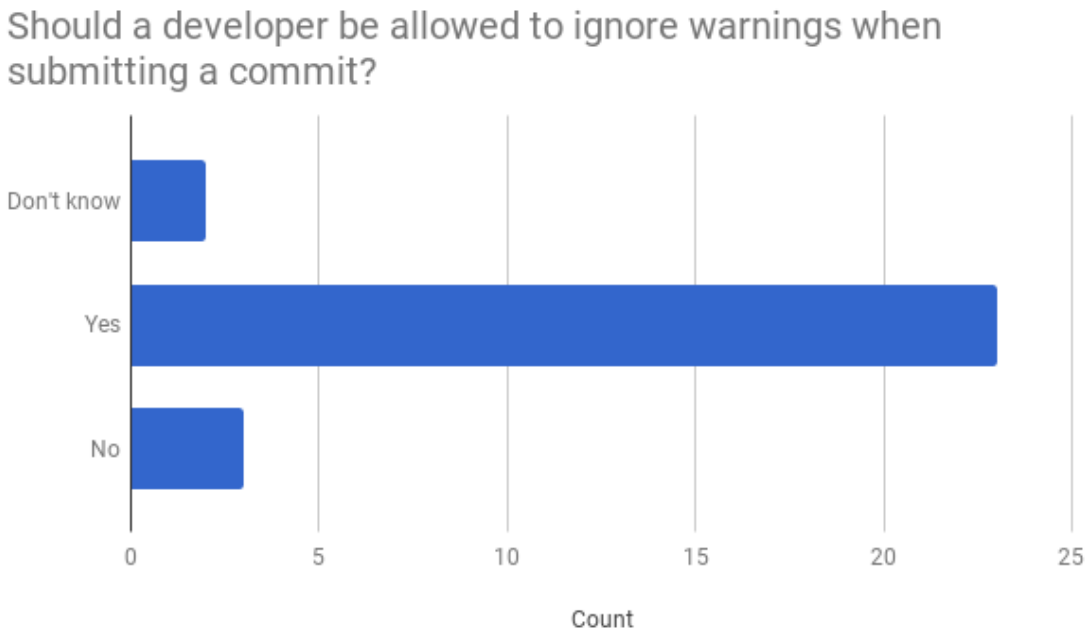

Figure 5.7: Respondents view on whether developers should be allowed to ignore warnings about a commit being defect inducing.

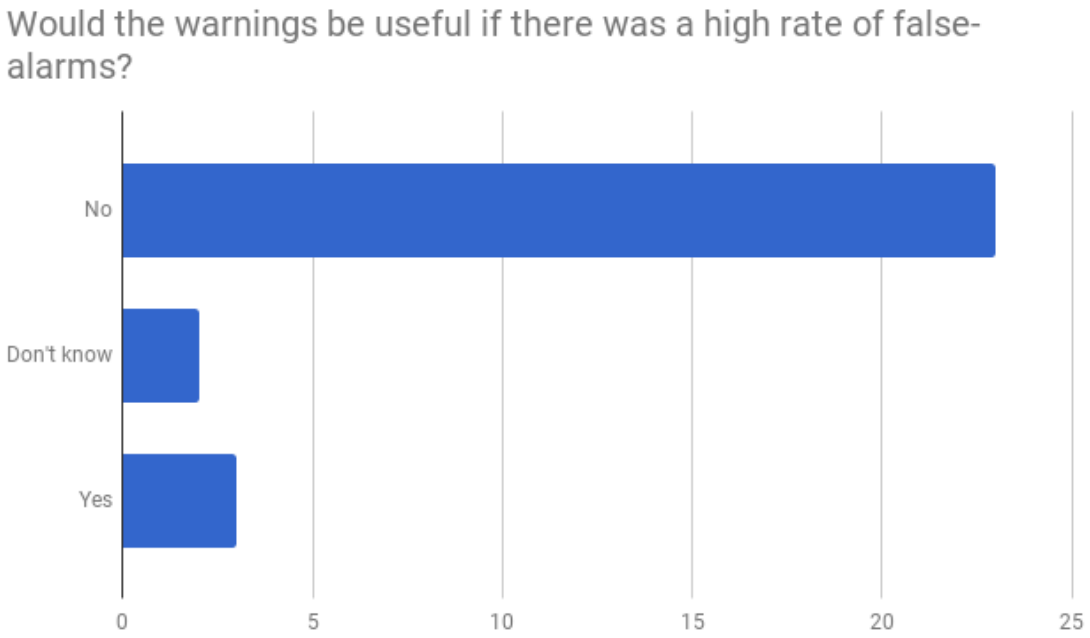

Figure 5.8: Respondents view on the importance of precision, when it comes to warnings about a commit being defect inducing.

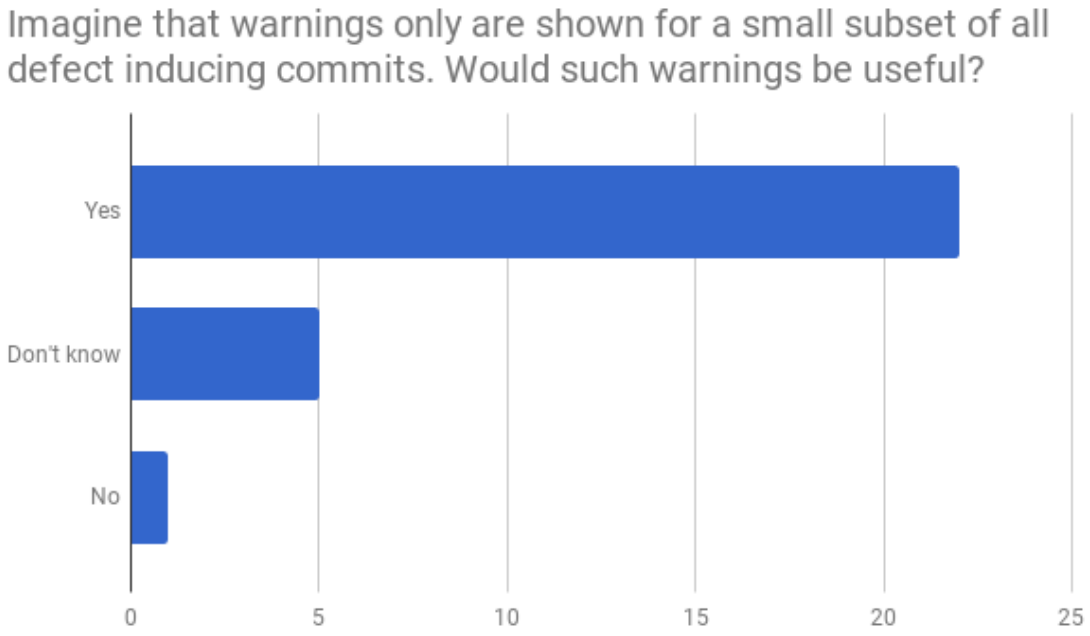

Figure 5.9: Respondents view on the importance of recall, when it comes to warnings about a commit being defect inducing.

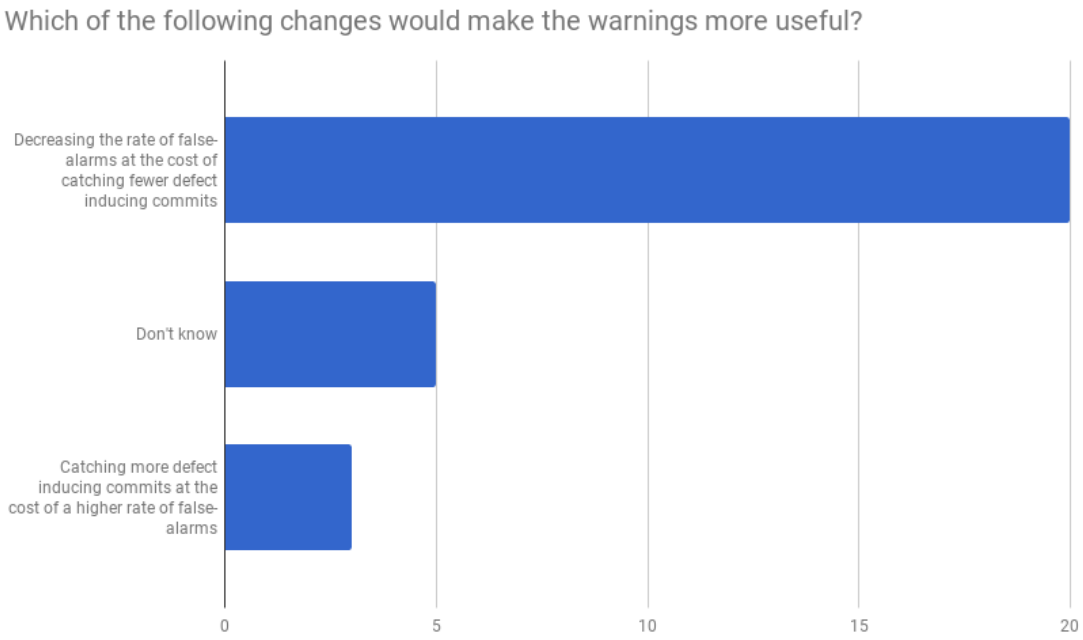

Figure 5.10: The preferred tuning option, when it comes to warnings about a commit being defect inducing.

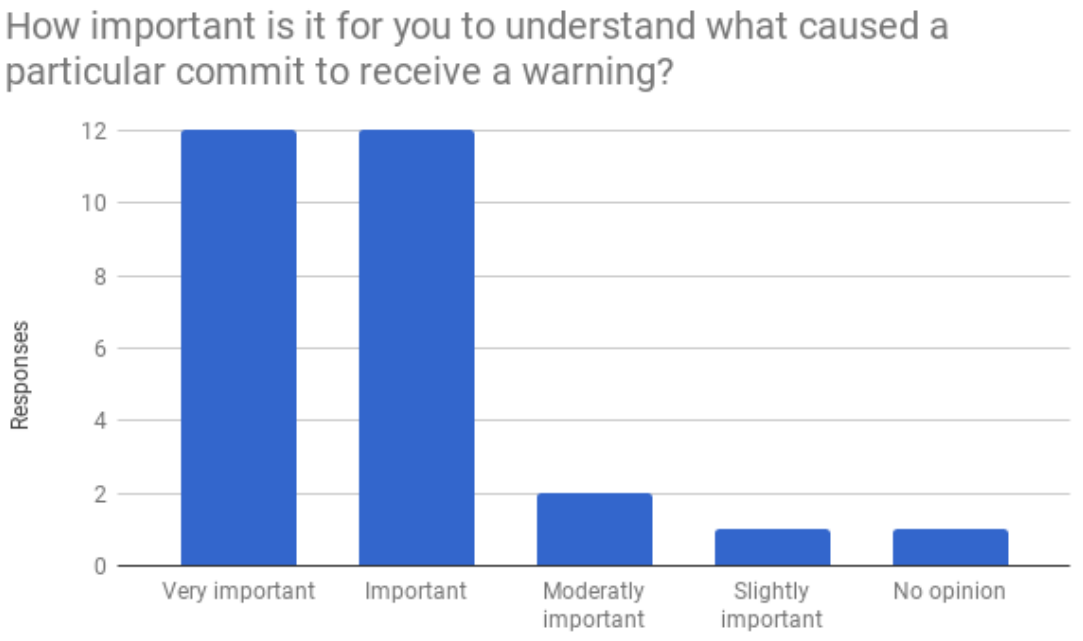

Figure 5.11: Importance of understandability among respondents, when it comes to warnings about a commit being defect inducing.

6

Discussion

This chapter discusses the result in relation to the research questions, as well as the threats to the validity of the study. Section 6.1 discusses what the study can say about what strategy is effective to implement cross-project defect prediction models within the industry. Section 6.2 examines the performance and ease of use of the metrics used in the study. Section 6.3 touches upon the result of the different pre-processing, learners and transfer learning techniques. Section 6.4 discusses what the result of the interviews and surveys can say about what is wanted in the performance of such a model. Section 6.5 discusses other areas that were encountered in the study but not tied to the research questions. Section 6.6 details the threats to internal as well as external validity that are present in the study.

6.1 RQ1 - Effective Strategy

The first questions in the study sought to determine how cross-project models could effectively be implemented within a company in the automotive sector.

The following two questions will be discussed to understand benefits, drawbacks and areas of improvement for the approach used in this study.

- Did an iterative process benefit the development of the model?
- Did the phases benefit the development of the model?

6.1.1 Did the iterative process benefit the development of the model?

We perceive the use of an iterative process as positive as we believe it to be crucial to be able to reevaluate prior results throughout the project.

Throughout the iterations, assumptions of what could be achieved, what metrics that were applicable and what constitutes a good performance changed. These

changes were done by evaluating upon prior results and testing the assumptions given. One example of this was our goal of detecting bug-inducing commits. Setting the goal early on enabled us to structure the tasks of extracting the data and how to evaluate the model. At the same time, these tasks played a crucial role in assessing if the goal was possible and relevant. In our case, if we were able to spot defect inducing commits and if developers would use such a model.

Performing a similar estimation in a plan-driven development process would require expertise regarding the pitfalls and challenges ahead or a more extensive inspection of data and elicitation of requirements. This would not be applicable for industry practitioners seeking to try out cross-project modelling, as they may not possess the required knowledge or have the time required to perform extensive pre-studies.

6.1.2 Did the phases benefit the development of the model?

In our approach, we elected to split the development into four phases each carrying out different sets of tasks for the implementation of the defect prediction models. In these phases, we aimed to achieve different objectives within the development of the models. For example, the objective of the data gathering phase was to collect a data set of measurements on commits.

We believe that the four phases are a useful concept to structure the tasks required to implement defect prediction models. As concepts, they illustrate the dependencies that tasks have on each other. For instance, it is difficult to evaluate a model performance without a model.

As concepts, they are also adaptable to new demands and situations. Most of the strategy is loosely defined to make it more generalisable which allows for greater adaptation.

The phases loose definition poses little to no clarification of how tasks should be completed. This is somewhat warranted as opinions and situations might differ across companies. For example, the act of data gathering in our case was inspecting and extracting data, where it could otherwise have been solved by the use of a third-party data set.

6.2 RQ2 - Metrics

The second research question in this study was to understand what formed a good metric for a company that works within the automotive sector. A "good metric" needed to have the three following attributes:

- The metric was relevant to a need expressed by the company.

- Measurements based on the metric could be gathered without too much difficulty using available data.
- The inclusion of measurement based on the metric had a significant impact on the final cross-project defect prediction model performance.

The pursuit of the first two attributes limited the metrics that were considered in the study and the measurements that were gathered, which in turn limits the generality of our findings. The extent of this limitation is hard to determine as the factor that influences the attributes might be similar to other automotive companies.

6.2.1 Are the metrics relevant to practitioners in the automotive industry?

The metrics that were used in this study tailored to the needs of the company in two ways: the metrics are all applicable on a change-level granularity level and work on commits in the language C.

The criterion of being applicable on change-level granularity was derived from the goal that was set in cooperation with quality management, which was to detect defect inducing commits. They motivated this as a way to decrease the time bugs were present in the system, as the developer would become aware and fix the bug quickly. This need to make predictions early on is echoed in the Kamei et al. study of change level metrics where they present it as one of three major benefits of change-level metrics [28].

The choice of ensuring that all metrics would work on C code was done since C is the main programming language within the studied projects. The major effect this had on our metric selection was the exclusion of several source code metrics from our consideration. The excluded metrics had either been shown to have low predictive power in procedural language [19] or were only applicable to object-oriented languages. At the same time, this exclusion did not affect the possibility to use the model for measuring changes in object-oriented languages as the only source code related metrics is lines of code.

Practitioners in the industry will find similar benefits of change-level granularity as the benefits of change-level metrics are not tied to any specific development process in the automotive industry. The focus on C code will also not impact the ability to use the measurement for other practitioners within the industry but does not cover all metrics that can be considered for object-oriented languages.

6.2.2 Is it possible to gather measurements from available data?

The answer to the question wherever a measurement could be gathered easily depends on whether third-party data sources can be considered. If the latter would be deemed suitable, all measurements within a third-party data set could be gathered with minimal effort provided that the metrics are relevant to the needs expressed by a company. The existence of data sets such as PROMISE[1] opens up the possibility to consider several measurements as easy to gather.

In this study we consider the choice of either extracting the data from the ITS and VCS or using a third-party data set and decided upon the former. This turns the question into how easy it is to extract the information from the available data repositories.

In our case, the measurements that were present in both the iterations were quite easy to collect from the VCS, with the exception of measurement based on subsystems. The same ease can be expected in other commercial development settings where VCS systems such as SVN or git are used.

The measurements added in iteration 2 (Exp and SExp) are only considered available in our case due to a high rate of sharing both developers and product code across the projects. Other practitioners in the automotive sector will likely be able to extract the Exp metric from their own VCS without considerable effort. The effort of the SExp metric is difficult to estimate since some modifications to the definition of subsystems were done during the study.

When it comes to metrics that were considered but not collected, we noticed severe difficulties in applying metrics regarding the history of files. Practitioners within the automotive industry may encounter similar problems depending on the VCS in use.

6.2.3 Which measurements contribute most to the outcome?

The usage of the random forest learner in this study limits what can be said about the relationship between the measurements and the outcome of the prediction. This is due to the random forest learners lower explanatory power. At the same time, it is still possible to answer how large role they play in relation to each other using the results of the variable importance analysis done on the random forest models. We found it difficult to motivate further analysis using learners with more explanatory and less predictive power, due to low AUC values of the models. The poor performance of random forest makes it difficult to draw any conclusion from models with worse performance.

[1] http://openscience.us/repo/code-analysis/

In the variable importance analysis of the measurements, we saw simple size measurements (LD, LA and LT) ranking high in both iteration 1 and 2, compared to more complex measurements. In addition, measurements based on the diffusion (Entropy, NF and NS) of the change scored the lowest. When including measurements that required prior version control history (Exp and SExp), we found that such measurements overall increased the AUC value and ranked highest when compared to other measurements.

The variable importance result has to be considered with the correlating measurement pairs that can be seen in figure 5.3. Both of the measurements within the pair will present similar information towards the model and can, therefore, have a similar degree of importance to the random forest model. This may be a reason why Entropy and NF are ordered next to each other.

As the study is limited to data available at one company and the models also have low AUC values it is difficult to evaluate if these trends are likely to appear in other commercial projects within the automotive industry.

6.3 RQ3 - Models

This research question was to understand the modelling techniques in the automotive company context. We have identified three parts of modelling that we think have a great impact on the result of a cross-project defect prediction model.

6.3.1 What pre-processing techniques are important?

In our data set, we encountered two pairs of correlating measurement as seen in fig 5.3, which made it necessary for us to handle the collinearity when comparing the learners. The correlation between NF and Entropy in our data deviates from what has been previously found to correlate in change-level measurement in a cross-project context [36]. This supports the point that collinearity cannot be assumed based on prior studies and will be necessary to check for future implementations.

The class-imbalance in our data set is high when comparing to data sets used in previous studies. An example is, the data sets used by Kamei et al. have between 5% and 44% defect inducing changes [36]. The projects in this study have on average less than 5% defect inducing changes.

Menzies et al. explains it is impossible to have both high recall and precision when the class-imbalance is high [11]. In the beginning, the expectation was that recall was of more interest than precision and SMOTE was used to handle the class-imbalance since it has a positive effect on recall. However, the interviews and survey indicate that the initial expectations were wrong and put precision as the most important

measurement. This makes SMOTE and other class-balancing efforts less useful since they tend to have a negative impact on precision [17].

6.3.2 Which learner performed best?

Random forest seems to have the best predictive power and this is in line with previous research [10]. However, the AUC of both our cross-project models and within-project models is quite poor compared to other studies [36, 5] as seen in table 5.4. Due to a number of unknowns in our data set we do not expect the performance of our models to be a problem with the learners, it is more likely the cause of poor data quality, too little data in the projects or the high class-imbalance of the data.

6.3.3 Which transfer learning technique performed best?

Both simple merge and bellwether have variance in performance across the projects. This makes it hard to estimate how the models would perform on new projects. With a varying performance, there is an unknown in how the model will perform when used in a new project and this might lower the trust of the model.

Using bellwether and P1 to P4 as training projects give similar results to that of the simple merge. But there is no bellwether, a project that produces the best predictor for all other projects, among the projects in the study. What is more noticeable is that P5 seems to be dissimilar to the other projects since it is both the worst training project as well as shows the poorest validation results. This could be the result of differences between the projects and some sort of clustering could improve performance by only using projects that are similar to each other when training.

6.4 RQ4 - Evaluation

Since change-level defect prediction models were constructed as part of the study, the interviews and survey have a focus on such models. This limits the conclusions that can be drawn from them to change-level defect prediction models. How the information from the model is to be used is most likely relevant to understand the results. The indication from the interviews is that a model should provide additional information during the development process. It should not be used to replace any of the current quality assurance efforts, such as limiting reviews to commits identified by the model as suggested by Kamei et al. [28].

6.4.1 How important is precision and recall for practitioners?

Our initial findings from the survey are that precision is important. As seen in 5.10, most respondents wanted to tune the model to be more precise in its prediction as well as disliked high false alarm rates as seen in 5.8. This is supported by the interviews, ID 1 and ID 8, where the importance of having high precision is expressed. There are two reasons for the high precision requirement. The first reason is to be able to trust the model and not just ignore the information it presents. The second is the additional cost of looking for a defect that does not exist. This is similar to results found by Huang et al. when evaluating usability for fault localisation tools [45]. This goes against Menzies et al. who argue against the use of precision for evaluation of models.

Recall is less relevant according to the findings of the survey since most respondents could see the benefit of a low recall model. We believe the reasoning behind this is that the developers do not expect a defect prediction model to replace the current review efforts. Instead, it is to be used together with the existing tools as an extra source of information to assist in quality assurance. When using an array of tools, such as linters, coding guidelines, and tests, there is no reliance on a single tool to assure quality, instead its the combination of the tools that are used to achieve a high-quality product.

There are multiple ways of using a defect prediction model. A use case in an automotive company is to be a fast and cheap filter to remove some of the defects from the system earlier than before. The main goal is not to find every defect but to find some defects faster than before and thus lower the time they are in the system. For such a model to be useful it needs a low overhead since too many false positives would remove the potential gain.

The value placed on precision over recall is interesting when considering the safety-critical focus in the automotive sector. Since recall corresponds to the ability to find defects, it is easy to imagine the importance of high recall in a safety-critical application where software needs to be defect free, due to safety requirements. However, our findings go against this. One possible explanation for this is that the practitioners in our study trusted and relied on their current tools and processes to build safety-critical systems. Defects are already expected to be found using the current workflow and there is limited value in using a prediction model to replace the current tools. Instead, the focus should be on the models' ability for quick and precise feedback where its unique capabilities may provide value alongside other quality assurance measures. Another possible explanation is that prediction models are seen as black box systems i.e. information is put in and a result comes out and what happens in between is partially unknown. Getting someone to trust a model that is not fully understood does not seem likely when working with safety-critical systems as it is a risk. However, by using models in conjunction with the other tools and processes this risk is mitigated while the model provides useful information.

How the value of precision and recall is interpreted in other industrial contexts needs further investigation. Practitioners might have other expectations when there are other qualities that have the primary focus.

6.4.2 What is the role of understandability in defect prediction models?

Both the interviews and the survey indicate that developers want explanatory power to understand the reasoning of the model. As seen in ID7 extra information is useful since it can assist the developer when looking for a defect. Measurements such as experience might be beneficial to know that it had a significant impact on the classification since it would allow the reviewer to look for beginner mistakes. However, measurements relating to sizes, such as LT or MessageLength, seem less likely to simplify the reviewing effort. This raises a new question about the benefits of explanatory power "Is explanatory power always helpful for defect prediction models?".

6.5 Others discussion topics

This section contains the following discussion topics:

- Should more elicitation be done in the goal phase?
- Is it viable to use company data for cross-project defect prediction?
- Was the cost-analysis an effective way to mitigate uncertainty in the data gathering phase?

All of these topics were encountered in our study but are not directly connected to our research questions. Additionally, they all touch on areas that could be of interest to practitioners who want to implement cross-project defect prediction models.

6.5.1 Should more elicitation be done in the goal phase?

The goal phase in both iterations was considerably shorter than the following phases, we motivated this with two factors: Firstly, uncertainties in what a viable goal could best be explored by actually attempting it. Secondly, the goal enabled us to set the scope of the remaining phases.

In hindsight, we see both drawbacks and benefits to the choice of and whether a short goal setting phase would be of best use.

The drawbacks of a short goal phase can be seen in the numerous wrong assumptions that could have been rectified if more time would have been spent on the elicitation phase. The main one to consider is a request for explanatory power that appeared in interviews and the survey. In the result in table 5.3, table 5.5 and figure 5.11 we can see that explanatory power could have been valued higher. But due to the late stage of evaluation and limited time, we had little opportunity to estimate how this trade-off should be considered in earlier stages of data gathering and model creation. It is likely if more focus would have been put into further exploring how developers perceive the role of detecting bug-inducing commits that such knowledge could have been acquired earlier.

The benefits of a short goal phase were made evident when extracting the data required for the defect prediction model. In this, we encountered an issue when applying the MA-SZZ algorithm. This issue which is further discussed in 6.5.2 could easily lead to a highly flawed process to determine what a bug-inducing commit actually was and made the goal unviable. To spot this issue before attempting the MA-SZZ would require knowledge about MA-SZZ, VCS and an extensive check of historic commits.

A viable argument can thus be made to prolong the goal phase with more elicitation if less uncertainty is present in the data gathering phase. This can either be achieved by using existing historic measurements or using a data set collected from a third-party.

6.5.2 Is it viable to use company data for cross-project defect prediction?

In the study, we elected to use only data gathered from local repositories. This choice made it possible to argue that the data collected was representative of the company. The drawback of this choice was the effort required to extract the data and the rather low performance of the cross-project defect prediction models.

The effort required to extract the data for the defect prediction model was shown to be substantial as most of the time of iteration 1 was spent on the data gathering phase. But this time was not mainly spent on extracting the measurements rather it was spent on trying to determine the defective status of commits. This was an issue since there were missing links between the feature branches and the master branch in the SVN system, which made it difficult to use the MA-SZZ. Fortunately, this information could be pieced together using the issue tracker and commit messages which enabled us to construct the chronological order of changes to files. The absence of this information would introduce a significant threat to the validity of the data. Without good data about defect inducing commits, it would be difficult to justify the use of any metrics on the change level.

In the end, this resulted in cross-project models with low performance in relation

to other studies [5]. The data quality presents an issue in using local data for such predictions. At the same time, local data is most likely to reflect behaviours specific to the company.

6.5.3 Was the cost-analysis an effective way to mitigate uncertainty in the data gathering phase?

One way we tried to manage uncertainties within the data gathering phase was to conduct a cost-analysis of the metrics with respect to the difficulty of extracting the measurements from the repositories. This allowed us to limit the metrics that needed to be gathered and therefore reduce the effort. The cost estimation was useful when it came to estimating the effort of extracting the measurements, but was not always accurate.

Our cost estimation of subsystems was incorrect since the definition of a subsystem as presented by Kamei et al. [36] was difficult to apply to file structure present in the local data repositories. To work around this issue we altered the subsystem category to correspond to the local file structure. This lead to the gathering of subsystem measurements requiring more effort than expected.

When it came to the history of files, the issue was that file renaming was not accurately reflected in the version control history. This was caused by the fact that subversion will consider a renamed file as a new file unless the repair command is used. The use of a file-history metrics would, therefore, require us to identify and fix the gaps present in the file history. This made it significantly more difficult to extract the measurements than others of the same cost category.

Overall we still believe that cost-analysis improved upon our efforts to manage issues and difficulties within the data gathering stage and should be considered when attempting to extract data from local repositories.

6.6 Threats to validity

Internal validity: There are some unknowns in the data that was used which might induce errors. First are the tasks labelled as defects in the ITS. Some of them might be requirement errors and should not be part of this study since a requirements error cannot be identified in the code, but there is no way to distinguish the tasks without going through all of them. There was not enough time to do this during the study but we estimate the number of mislabelled tasks to be low.

While gathering data from the VCS some problems were encountered that could have an impact on the results of our study. The primary problem was the tracing of changes on feature branches. The commands available could not present the changes

done to a file in a feature branch and the tracing had to be reconstructed using data from both the ITS and VCS. The scripts constructed to automate the reconstruction of the history may contain bugs. Another problem with the tracing in the history is moving and renaming files. If this is done incorrectly by a developer the history of a file is lost in the VCS. This might have had a negative effect on the quality of defect inducing commit data.

The VCS contain more than source code files. To handle this commits had to be filtered since the study focused on the source code. This filtering might introduce errors in several ways. First is incorrectness by filtering too much or too little. There is also the potential that the non-source code files affect our measurements. For example, writing documentation might increase your experience with the system but documents are excluded from our study and thus this might have been missed.

Heuristics by definition are not exact and there are some known weaknesses with the MA-SZZ algorithm. The most important problem is that bugs that are fixed by only adding code cannot be found by the algorithm. An alternative to using a search algorithm is manually searching but that would have limited the number of defects that we would have time to find and have the potential to introduce other errors due to our limited knowledge of C and embedded systems.

How the qualitative data was gathered is a potential source of errors. The interviews might contain biases introduced by the researchers in the form of how the questions were asked and how the analysis was done. At the same time, the questions in the questionnaire might also be biased due to how they were asked and placed within the survey.

The questionnaire might also be biased in that the questions were difficult to comprehend for the respondents. We attempted to mitigate this risk by both including options as "don't know" as well as testing the questionnaire on five developers that would not be a part of the survey.

The selection of interviewees was not done at random, instead, the three interviewees were selected due to interest in the subject. This introduces bias in their answers and their views might not be that of the entire company. To counter some of this bias the three interviewees had different roles within the development and experience.

External validity: There are a large number of companies in the automotive sector and there could be variance in how they operate. It is hard to determine whether the company where the study was done is a good representation of other companies in the automotive sector.

The alteration to how subsystems may have affected its performance and can make the results found regarding this metrics in the study less relevant for others.

7

Conclusion

In our study, we set out to understand how to incorporate cross-project defect prediction models in a commercial context. We studied this by developing such models at an automotive company through an iterative approach containing four phases: goal setting, data gathering, modelling and evaluation. During this, we attempted to study what metrics and modelling techniques from the research could be applicable in a commercial setting.

We were able to construct a cross-project defect prediction model through our process, which was beneficial to handle the uncertainties and reflection necessary in the process. The performance of the final model is rather limited but we do not believe this to be a result of the process. The poor performance is more likely to be caused by poor data quality in the available data repositories.

Further on, we found that change-level metrics are sought after in the automotive industry and the metrics we used can easily be extracted from local data repositories. In our study, measurements of developers' experience and code size outperformed other measurements, though these results need to be explored further since it was based on models with low explanatory power and poor predictive performance.

In our attempt to model the data, we found that correlation tests are necessary to avoid collinearity within our data set. In our data set, we also encountered class-imbalance which we addressed with SMOTE. Although, later evaluation of the model indicated a need for higher precision which would indicate that SMOTE may not be warranted.

We found that the random forest learner performed best in comparison to other learners, though low in regards to the performance of other studies. The low performance was also found for two transfer learning techniques of simple merge [16] and bellwether [32]. Though this result must be interpreted with caution as learners performed subpar even on within-project data.

When evaluating the model we found that practitioners perceive high precision as important for the usage of the model while recall is less relevant. Further on, our findings suggest that defect prediction models' performance should not be considered as a stand-alone tool for quality assurance. Instead, it should be considered how

it ties into the entire development process and works with other quality assurance tools.

The result of the evaluation is important to consider for the usage of defect prediction models, as it provides an argument for favouring models that have high precision over high recall.

Based on the interesting result of the evaluation, we suggest that further investigation into what performance practitioners want is needed. Further on, we also see a benefit in investigating how defect prediction models can work in unison with other tools and processes used for quality assurance.

Bibliography

[1] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 8 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7390282/

[2] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12, 2002. [Online]. Available: http://ieeexplore.ieee.org/document/5386906/

[3] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: Current results, limitations, new approaches," *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, 12 2010. [Online]. Available: http://link.springer.com/10.1007/s10515-010-0069-5

[4] M. Jureczko and L. Madeyski, "Cross-project defect prediction with respect to code ownership model: An empirical study," *e-Informatica Software Engineering Journal*, vol. 9, pp. 21–35, 2015. [Online]. Available: http://www.e-informatyka.pl/attach/e-Informatica_-_Volume_9/eInformatica2015Art2.pdf

[5] S. Hosseini, B. Turhan, and D. Gunarathna, "A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction," *IEEE Transactions on Software Engineering*, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8097045/

[6] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan, "A Framework for Evaluating the Results of the SZZ Approach for Identifying Bug-Introducing Changes," *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 641–657, 7 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7588121/

[7] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change bursts as defect predictors," in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*. IEEE, nov 2010, pp. 309–318. [Online]. Available: http://ieeexplore.ieee.org/document/5635057/

[8] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceeding of the 28th international conference on Software*

engineering - ICSE '06. New York, New York, USA: ACM Press, 2006, p. 452. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1134285.1134349

[9] A. E. Hassan, "Predicting faults using the complexity of code changes," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 78–88. [Online]. Available: http://ieeexplore.ieee.org/document/5070510/

[10] 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, 7 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4527256/

[11] 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, 1 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4027145/

[12] 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, 11 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6035727/

[13] A. B. de Carvalho, A. Pozo, and S. R. Vergilio, "A symbolic fault-prediction model based on multiobjective particle swarm optimization," *Journal of Systems and Software*, vol. 83, no. 5, pp. 868–882, 5 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121209003367

[14] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *CoRR*, vol. abs/1801.10269, 2018. [Online]. Available: http://arxiv.org/abs/1801.10269

[15] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artificial Intelligence Review*, pp. 1–73, 5 2017. [Online]. Available: http://link.springer.com/10.1007/s10462-017-9563-5

[16] Y. Kamei and E. Shihab, "Defect prediction: Accomplishments and future challenges," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 3 2016, pp. 33–45. [Online]. Available: http://ieeexplore.ieee.org/document/7476771/

[17] A. Agrawal and T. Menzies, ""better data" is better than "better data miners" (benefits of tuning SMOTE for defect prediction)," *CoRR*, vol. abs/1705.03697, 2017. [Online]. Available: http://arxiv.org/abs/1705.03697

[18] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Information and Software Technology*, vol. 76, pp. 135–146, 8 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584916300738

[19] D. Radjenovi, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, pp. 1397–1418, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584913000426

[20] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 6 1994. [Online]. Available: http://ieeexplore.ieee.org/document/295895/

[21] M. Lorenz and J. Kidd, *Object-oriented Software Metrics: A Practical Guide*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

[22] L. Briand, P. Devanbu, and W. Melo, "An investigation into coupling measures for C++," in *Proceedings of the 19th international conference on Software engineering - ICSE '97*. New York, New York, USA: ACM Press, 1997, pp. 412–421. [Online]. Available: http://portal.acm.org/citation.cfm?doid=253228.253367

[23] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:10. [Online]. Available: http://doi.acm.org/10.1145/1868328.1868342

[24] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local versus global lessons for defect prediction and effort estimation," *IEEE Transactions on Software Engineering*, June 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6363444/

[25] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, pp. 170–190, 3 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914002523

[26] T. Graves, A. Karr, J. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, 7 2000. [Online]. Available: http://ieeexplore.ieee.org/document/859533/

[27] K. Herzig, "Using Pre-Release Test Failures to Build Early Post-Release Defect Prediction Models," pp. 300–311, 11 2014. [Online]. Available: https://www.microsoft.com/en-us/research/publication/using-pre-release-test-failures-to-build-early-post-release-defect-prediction-models/

[28] Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 757–773, 6 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6341763/

[29] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect

prediction," *IEEE Transactions on Software Engineering*, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7959597/

[30] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," in *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*. New York, New York, USA: ACM Press, 2016, pp. 309–320. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2884781.2884839

[31] S. Herbold, A. Trautsch, and J. Grabowski, "Global vs. local models for cross-project defect prediction," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1866–1902, 8 2017. [Online]. Available: http://link.springer.com/10.1007/s10664-016-9468-y

[32] R. Krishna, T. Menzies, and W. Fu, "Too much automation? the bellwether effect and its implications for transfer learning," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016*. New York, New York, USA: ACM Press, 2016, pp. 122–131. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2970276.2970339

[33] Hongyu Zhang and Xiuzhen Zhang, "Comments on "Data Mining Static Code Attributes to Learn Defect Predictors"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 635–637, 9 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4288196/

[34] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with Precision: A Response to "Comments on 'Data Mining Static Code Attributes to Learn Defect Predictors'"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 637–640, 9 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4288197/

[35] K. Herzig and N. Nagappan, "Empirically Detecting False Test Alarms Using Association Rules," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 5 2015, pp. 39–48. [Online]. Available: http://ieeexplore.ieee.org/document/7202948/

[36] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A. E. Hassan, "Studying just-in-time defect prediction using cross-project models," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2072–2106, 10 2016. [Online]. Available: http://link.springer.com/10.1007/s10664-015-9400-x

[37] N. Nagappan, T. Ball, and B. Murphy, "Using Historical In-Process and Product Metrics for Early Estimation of Software Failures," in *2006 17th International Symposium on Software Reliability Engineering*. IEEE, 11 2006, pp. 62–74. [Online]. Available: http://ieeexplore.ieee.org/document/4021972/

[38] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2107–2145, 10 2016. [Online]. Available: http://link.springer.com/10.1007/s10664-015-9396-2

[39] L. An and F. Khomh, "An Empirical Study of Crash-inducing Commits in Mozilla Firefox," in *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering - PROMISE '15*. New York, New York, USA: ACM Press, 2015, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2810146.2810152

[40] J. G. Barnett, C. K. Gathuru, L. S. Soldano, and S. McIntosh, "The relationship between commit message detail and defect proneness in java projects on github," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 5 2016, pp. 496–499. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7832934&isnumber=7832751

[41] M. Greiler, K. Herzig, and J. Czerwonka, "Code Ownership and Software Quality: A Replication Study," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 5 2015, pp. 2–12. [Online]. Available: http://ieeexplore.ieee.org/document/7180062/

[42] O. Mizuno and Y. Hirata, "A Cross-Project Evaluation of Text-Based Fault-Prone Module Prediction," in *2014 6th International Workshop on Empirical Software Engineering in Practice*. IEEE, 11 2014, pp. 43–48. [Online]. Available: http://ieeexplore.ieee.org/document/6976021/

[43] "An experience report on defect modelling in practice: Pitfalls and challenges," in *In Proceedings of the International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP'18)*, 2018, p. To Appear.

[44] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "An Empirical Comparison of Model Validation Techniques for Defect Prediction Models," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 1–18, 1 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7497471/

[45] Q. Huang, X. Xia, and D. Lo, "Supervised vs Unsupervised Models: A Holistic Look at Effort-Aware Just-in-Time Defect Prediction," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 9 2017, pp. 159–170. [Online]. Available: http://ieeexplore.ieee.org/document/8094418/

A

Appendix 1

Protocol for the interviews in iteration 1. The interviews were conducted in Swedish. The explanations are translated while the questions are presented in their original form.

A.1 Introduction

Shortly explain the purpose of interview as to evaluate what can be considered as a good performance for the thesis defect prediction model.

Thereafter explain that the interviewee may at any point during the interviewee interrupt and ask further questions if anything is difficult to understand or hard to grasp.

Following this, explain that there are no right or wrong answers to the questions so there is no reason to worry about answering correctly.

Finally explain that it is acceptable to explore sidetracks that are not touched upon by the questions.

A.2 Warm up

- Vad har du för tidigare utbildning??
- Hur länge har du varit på CPAC?
- Vad har du gjort under den tiden?
- Vad har du för erfarenhet av arbete med mjukvarukvalitet?

A.3 Main body

Give a short explanation of how the predictions are done in the current cross-project defect prediction model and the intended goal for such predictions.

A.3.1 Code inspection

- Vilken roll spelar kodgranskning i utvecklingsprocessen?
- Hur mycket tid uppskattar du läggs ner på kodgranskning under en vecka?
- Defect prediction is current workflow
- Om du hade en model som visar risken att en commit introducerar en ny defekt. När i processen vill du ha denna information?
- Innan du gör en commit, när du ska granska någon annans kod?

A.3.2 Scenario describing

- Hur skulle du reagera på att dina commits ibland flaggades som riskfyllda?
- Hur skulle du använda informationen om att en commit har 3 gånger högre sannolikhet att innehålla en ny defekt? Som utvecklare? Som granskare?

A.3.3 Alternative costs

- Hur mycket tid spenderas på att fixa defekter?
- Vad är kostnaden för ett fel som upptäcks under utvecklingen?
- Vad är kostnaden för ett fel som upptäcks hos kund?

A.3.4 Trade-offs

- Eftersom att precision en är 11% betyder det att endast 1 av 9 flaggade commits innehåller en defekt. Hur skulle detta påverka din tolkning av informationen?
- Om man ser modellen från ett annat perspektiv. Att genom att granska 19% av alla commits så kan man hitta 44% av alla defekter.

- Skulle tiden som läggs ner på extra kodgranskning vara kostnadseffektiv?
- Om vi skulle göra modellen bättre. Om man skulle ändra tröskelvärdet i skulle man öka antalet hittade defekter eller precisionen på modellen? På bekostnad av det andra.

B

Appendix 2

Protocol for the interview in iteration 2. The interviews were conducted in Swedish. The explanations are translated while the questions are presented in their original form.

B.1 Introduction

Shortly explain the purpose of interview as to evaluate what can be considered as a good performance for the thesis defect prediction model.

Thereafter explain that the interviewee may at any point during the interviewee interrupt and ask further questions if anything is difficult to understand or hard to grasp.

Following this, explain that there are no right or wrong answers to the questions so there is no reason to worry about answering correctly.

Finally explain that it is acceptable to explore sidetracks that are not touched upon by the questions.

B.2 Warm up

- Vad har du för tidigare utbildning??
- Hur länge har du varit på CPAC?
- Vad har du gjort under den tiden?
- Vad har du för erfarenhet av arbete med mjukvarukvalitet?

B.3 Main body

Give a short explanation of how the predictions are done in the current cross-project defect prediction model and the intended goal for such predictions.

B.3.1 Recall

- I ditt nuvarande projekt hur långt är det mellan olika releases?
- I ditt nuvarande projekt, hur viktigt är det att en majoritet av defekter upptäcks innan den når release?
- Vilka verktyg eller processer används för nuvarande för att motverka att defekter kommer ut i produkterna?
- Finns det en roll som vår modell ska spela i detta skedde?
  - i så fall vad är den?
- Kan denna roll genomföras om endast ett litet antal av defekter snappas upp av modellen?

B.3.2 Trust

- Hur viktigt är det att du kan lita på att modellens varningar?
  - Vad skulle få dig att lita på modellen?
  - Vad skulle få dig att inte lita på modellen?
- Hur mycket extra tid skulle du uppskatta att du skulle lägga ner på att granska en commit med en varning?
  - Vilka krav skulle du ha på modellen för att detta skulle kännas befogat?

B.3.3 Understanding

- Hur viktigt är det att förstå hur modellen fungerar för att använda den?
- Hur viktigt är det att förstå den underliggande orsaken till en varning?
  - Om viktigt. Skulle en förklarande modell vara användbar även om den inte var precis i vad som får en varning.

- Skulle du finna det hjälpsamt att få reda på vad som särskilt flaggat en commit om du redan har kunskapen om vad som kan påverka modellen?

B.3.4 Wrap up

Ask the interviewee if there is any questions they want to ask us and touch upon areas that was touched upon in the interview, but not explored further.

C

Appendix 3

´

C.1 Introduction

As a part of our thesis work we have created a model that can predict whether a commit is introducing a new defect or not. To get a better understanding of how such a model can be used and how users would react to it we have created a survey. The responses will be presented in our thesis in an aggregated form.

The survey is anonymous and we do not collect any personal information about the respondents.

C.2 Questions

Would you want warnings about commits that are likely to introduce defects? Yes: 24, No: -, Don't know: 4
Follow up - if Yes: Where in your current work-flow would you want such a warning? Before commit to master branch: 4, When reviewing: 12, Other: 6

Should a developer be allowed to ignore warnings when submitting a commit? Yes: 23, No: 3, Don't know: 2

Should a reviewer be required to spend extra time reviewing a commit that received a warning? Yes: 17, No: 8, Don't know: 3

Imagine that warnings only are shown for a small subset of all defect inducing commits. Would such warnings be useful? Yes: 22, No: 5, Don't Know: 1

Is there a minimum amount of defect inducing commits that have to be identified for it to be useful? Yes: 4, No: 11, Don't know: 13
Follow up - if Yes: What amount of defect inducing commits have to be identified for the warnings to be useful? 1: 1, N/A: 3

Would the warnings be useful if there was a high rate of false-alarms?
Yes: 23, No: 3, Don't know: 2
Follow up - if Yes: What rate of false-alarms would make it likely that you ignored the warnings? 1-10%: 4, 11-20%: 2, 21-30%: 0, 31-40%: 1, 41-50%: 4, N/A:6

How important is it for you to understand what caused a particular commit to receive a warning? Very Important: 12, Important: 12, Moderately Important: 2, Slightly Important: 1, Not Important: -, No Opinion: 1

How important is it that a commit can be changed to remove the warning? Very Important: 9, Important: 10, Moderately Important: 3, Slightly Important: 2, Not Important: 2, No Opinion: 2

How important is it to provide hints to what can be changed to remove a warning? Very Important: 6, Important: 10, Moderately Important: 7, Slightly Important: 4, Not Important: 1, No Opinion: -

How would you react to one of your commits receiving a warning? Fix: 11, Investigate: 7, Other: 6

How would you review a commit that had received a warning? Reject: 4, Extra time: 17, Other: 4