Using Machine Intelligence to Prioritise Code Review Requests

Nishrith Saini
This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

The author declares that he is the sole author of this thesis and that he has not used any sources other than those listed in the bibliography and identified as references. He further declares that he had not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Nishrith Saini
E-mail: nisi18@student.bth.se

University advisor:
Ricardo Britto
Department of Software Engineering

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Background. Modern Code Review (MCR) is a process of reviewing code which is a commonly used practice in software development. It is the process of reviewing any new code changes that need to be merged with the existing code base. As a developer, one receives many code review requests daily that need to be reviewed. When the developer receives the review requests, they are not prioritised. Manually prioritising them is a challenging and time-consuming process.

Objectives. This thesis aims to address and solve the above issues by developing a machine intelligence-based code review prioritization tool. The goal is to identify the factors that impact code review prioritization process with the help of feedback provided by experienced developers and literature; these factors can be used to develop and implement a solution that helps in prioritising the code review requests automatically. The solution developed is later deployed and evaluated through user and reviewer feedback in a real large-scale project. The developed prioritisation tool is named as Pineapple.

Methods. A case study has been conducted at Ericsson. The identification of factors that impact the code review prioritization process were identified through literature review and semi-structured interviews. The feasibility, usability, and usefulness of Pineapple have been evaluated using static validation method with the help of responses provided by the developers after using the tool.

Results. The results indicate that Pineapple can help developers prioritise their code review requests and assist them while performing code reviews. It was found that the majority of people believed Pineapple has the ability to decrease the lead time of the code review process while providing reliable prioritisations. The prioritisations are performed in a production environment with an average time of two seconds.

Conclusions. The implementation and validation of Pineapple suggest the possible usefulness of the tool to help developers prioritise their code review requests. The tool helps to decrease the code review lead-time, along with reducing the workload on a developer while reviewing code changes.

Keywords: Code Review, Prioritisation, Bayesian Networks, Gerrit, Machine Intelligence
Acknowledgments

I would like to express my deepest gratitude and special thanks to my thesis supervisor Prof. Ricardo Britto for his continuous support and guidance who despite being a very busy person with his duties took time to guide me throughout my thesis and helped me to stay in the right path.

I also would like to convey my deep sense of gratitude to my Manager, Bodil Jansson, whose guidance, encouragement, suggestions, advice, and constructive criticism helped in the evolution of my ideas. I am also thankful for having the opportunity to meet so many wonderful and skilled people who led me through in my thesis. This thesis would not be possible without the crucial and gracious co-operation of all the employees at Ericsson who took part in the interviews, meetings, and assisted me during my research and development phase.

Finally, I would like to extend our heart full love and gratitude to my parents, family, and friends for showing their moral support and backing me in every situation. This thesis could not be achieved without their support. Thank you very much, everyone!
## Contents

| Section                        | Page |
|-------------------------------|------|
| Abstract                      | i    |
| Acknowledgments               | iii  |
| 1 Introduction                | 1    |
| 2 Aim and objectives          | 3    |
| 2.1 Objectives                | 3    |
| 2.2 Research questions        | 4    |
| 3 Background                  | 5    |
| 3.1 Terminology               | 5    |
| 3.1.1 General                 | 5    |
| 3.1.2 Code Review             | 5    |
| 3.1.3 Machine Learning        | 6    |
| 3.2 Code Review               | 7    |
| 3.3 Bayesian Networks         | 8    |
| 3.3.1 Overview                | 8    |
| 3.3.2 Inference               | 10   |
| 4 Related Work                | 11   |
| 4.0.1 Limitations from Related Work | 12  |
| 5 Method                      | 13   |
| 5.1 Research Method Selection | 13   |
| 5.2 Research Design           | 14   |
| 5.2.1 Case and Unit of Analysis | 14  |
| 5.2.2 Data Collection         | 16   |
| 5.2.3 Data Analysis           | 19   |
| 6 Pineapple Overview          | 25   |
| 6.1 Architecture              | 26   |
| 6.1.1 ETL                     | 26   |
| 6.1.2 ML                      | 26   |
| 6.1.3 Prioritiser             | 28   |
| 6.1.4 Front-End               | 28   |
| 6.1.5 Core                    | 29   |
| 6.2 Development               | 29   |
List of Figures

3.1 Modern Code Review Process [1] .............................................. 7
3.2 Bayesian Network Example [2] .............................................. 9

5.1 Research Design Structure .................................................... 14
5.2 Active Developers involving in creating new change requests ........... 21
5.3 Newly created change requests per month .................................. 21

6.1 Pineapple Overview ............................................................. 25
6.2 Pineapple Architecture Overview ............................................. 26
6.3 Bayesian Network Architecture .............................................. 27

7.1 Q1: Pineapple is useful to prioritise review requests ....................... 37
7.2 Q2: Pineapple is easy to use and understand ................................ 38
7.3 Q3: Pineapple helps in decreasing the lead time of review process .... 38
7.4 Q4: Pineapple helps inexperienced developers in performing code reviews 39
7.5 Q5: Prioritisations done by Pineapple are reliable ......................... 39

A.1 Pineapple web interface with feedback questionnaire .................... 55
A.2 Pineapple web interface with prioritised review requests ................ 56
## List of Tables

| Table | Title | Page |
|-------|-------|------|
| 5.1   | Research data sources | 16   |
| 5.2   | Description of semi-structured interviewees | 18   |
| 5.3   | Scope of data considered for training and development | 20   |
| 7.1   | Factors identified from Literature Review and Interviews | 32   |
| 7.2   | Factor Selection Criteria | 33   |
| 7.3   | Bayesian Network Performance Metrics | 35   |
| 7.4   | Comparison of Model Evaluation Metrics | 36   |
| 7.5   | Response time for prioritisation requests | 36   |
| B.1   | Interview structure for the factor identification process | 57   |
| C.1   | Questionnaire used to do the static validation of Pineapple | 59   |
| D.1   | Statistical Analysis for Age of the changes | 61   |
| D.2   | Statistical Analysis for Size of the changes | 61   |
| D.3   | Statistical Analysis for Number of Patches of the changes | 61   |
| E.1   | Data Structure used in Bayesian Network | 63   |
Chapter 1

Introduction

Software code review is the practice that involves team members to check/critique the changes made to an existing software system before the code changes are integrated into the central development [3]. Code reviews helps in improving the quality of the code and reduce the defects raised post-integration [1] [5] [6]. Performing code reviews has been a primary concept in software development [7] [8]. The conventional code review processes were cumbrous and time-consuming [9], which led to the introduction and evaluation of modern code review.

Modern Code Review (MCR) is a lightweight alternative to a conventional code review process and it is practised widely in open source software projects and large organisations [10]. Dedicated tools such as Gerrit and Github are used to make the modern code review process more manageable, comfortable and effective [8] [11].

Incorporating MCR in the software development process helps in achieving software with fewer bugs and easy maintenance [8]. MCR allows developers to monitor the code changes and give feedback to the developers before approving to integrate the suggested changes [12]. Code reviews also help in better understanding of code and knowledge among the involved developers in a team [7] [8] [10].

As a developer, one may receive a large number of code review requests. Prioritising these requests is considered as one of the biggest concern in their day-to-day work [13].

Having received a large number of code changes, the reviewer has to select the requests manually to integrate them with the existing code. This process of manual selection and prioritisation of review requests is time consuming and additional workload for the reviewer. Considering these notable amounts of time and effort spent on selecting review requests, we can say that it may lead to waste of time and have longer lead times [10].

Generally, the listing of code review requests is done according to the time they are created and not by their relevance. As code review requests are sent to multiple reviewers, it may be the case that another reviewer has already reviewed the code change and it need not be reviewed again.
Some of the code review requests may be minor changes to the code that do not need the utmost attention of the reviewer. There may be review requests which do not consist of any test cases which requires reviewers attention. There may be times that the reviewer wants to review a requested code change expeditiously and does not have time to prioritise them.

According to [13], it was observed that many reviewers do not even prioritise the review requests.

The consequences caused by the problems mentioned above are:

- The time spent prioritising the review requests not only costs development time, but also increases the overall lead time of the code review process.
- A small-time delay in the review process may seem negligible but as they occur several times, it affects the overall performance and development speed of a team or an individual.

To overcome all these problems mentioned above, prioritising code review requests without any manual involvement is required. Automating the prioritisation of code review requests without any manual effort may help to decrease the lead time of a code review process, reduce the workload for the reviewers, increase the development productivity.

The goal of this research is to develop and evaluate an machine intelligence-based tool that helps code reviewers to prioritise code review requests that, in turn, helps in decreasing the overall lead time of code review process along with help reducing the workload on the reviewers.

The sub-goals of the thesis is to, identify factors and relationship between them. Use the identified factors to determine the importance of a given code review request along with validating the tool with user feedback in a live environment.

From existing literature, we can learn that AI-Based, Machine Learning (Machine Intelligence) approaches have been used while solving problems related to prioritisation, classification etc. Based on the motivations, issues and limitations mentioned above, I would like to propose a new solution called Pineapple.

The main contributions of the research work are as follows:

- The identification of factors and the relationship between them, which impact the importance/prioritisation of code review requests.
- A machine intelligence-based tool (Pineapple) that uses a Bayesian Network which help in prioritising the code review requests by calculating the probability of the change being merged, alongside addressing some of the limitations associated with previous and existing solutions.
- Perform an empirical evaluation of Pineapple in an industrial environment.
Chapter 2

Aim and objectives

2.1 Objectives

The main aim of this research is to address the limitations and research gaps associated in the existing literature (as described in section 3) by developing a tool which prioritises code review requests automatically for code reviewers. The developed tool helps to address the limitations in the following ways:

- Pineapple accounts for the inputs provided by experienced developers in relation to the factors and feedback provided, which help in more efficient prioritisations.

- Helps in decreasing time and workload along with decreasing the lead time of review process while accounting for uncertainty.

- Pineapple considers review requests from multiple projects in which a developer is a contributor.

- An evaluation of the tool in a real industrial environment based on the feedback provided by the users along with performance metrics.

Pineapple is a machine intelligence-based code review prioritisation tool which is built using the concept of Bayesian Networks. Here the factors act as the network nodes, and the relationship between them provide a Directed Acyclic Graph (DAG). This Bayesian Network helps us find the conditional probability between the factors and try to prioritise the code review requests.

The objectives of this research are:

OBJ1: Identify the factors considered to be essential by experienced developers and literature to prioritise code review requests automatically.

OBJ2: Implement a code review prioritisation tool (Pineapple) to help code reviewers prioritise their review requests automatically. Pineapple was implemented using the factors identified in OBJ1.

OBJ3: Evaluate the tool (Pineapple) by performing both performance evaluation and user evaluation after deploying on a large scale environment.
2.2 Research questions

In this thesis, we answer the following research questions formulated based upon the aims and objects mentioned in the section 2.1.

**RQ1:** What are the factors that relate to prioritising code review requests in an effective way?
In order to prioritise the code review requests, we need to know the factors which impact/effect the review requests. This RQ helps us find these factors and help in further research.

**RQ2:** How well does the developed prioritisation tool (Pineapple) perform?
After the prioritisation tool is developed, we need to validate the predictions and the results produced by the tool along with validating the approach selected to implement the tool. This RQ helps in performance validation of the prioritisation tool.

**RQ3:** How feasible and useful is the developed Pineapple tool for the code reviewers in a large scale environment?
This RQ helps in evaluating the feasibility, usability and the scalability of the developed prioritisation tool in a real-live large scale environment.

**RQ1** aims to identify and generate a list of possible factors that are considered important by the reviewers to prioritise the code review requests. This developed list will be used as a basis of the information in developing Pineapple.

**RQ2** helps in addressing the performance of the tool along with validating the approach selected to develop the tool when comparing the performance metrics with other existing approaches. Here, to validate performance metrics of the Pineapple, we consider the performance metrics such as the RMSE (Root Mean Square Error) and MAE (Mean Average Error) which help us to know how less is the deviation for the predictions made by the tool to the actual results. These metrics are calculated using the data that is not used for training. The predictions/prioritisation done by Pineapple are also validated by the end-users themselves through the feedback questionnaire present in the Pineapple itself.

**RQ3** helps in examining the feasibility and usefulness of Pineapple from the point of view of its users (developers). The feedback questionnaire provided in Pineapple helps us to collect and analyse the feedback and the perspective of the developers/code reviewers. This RQ helps us to find out whether Pineapple is easy to use; prioritisations done by Pineapple are reliable; and whether Pineapple can be used regularly.
Chapter 3

Background

In this chapter, the key concepts and terminology related with this research are presented.

3.1 Terminology

In this section, we have grouped the terminology associated with this thesis. Further, the terminology is divided into their particular field of use.

3.1.1 General

- **Docker** – a software product that creates and runs containers (a container can be defined as a lightweight Virtual Machine which “contains” the supplied software along with its co-responding dependencies. The result is that containers can be deployed in almost any environment that is running docker with minimal adjustments to the runtime environment.

- **REST-API** – standing for Representational State Transfer, a REST API is a web service API that sticks to the rules of stateless transfer of data between computer machines connected to a network.

- **ETL** – An acronym for Extract, Transform, and Load. It refers to the combined action of extracting and transforming data from a data source and then loading it into a compatible data destination.

3.1.2 Code Review

- **Commit**: The set of changes done to the existing code that needs to be reviewed and later integrated to the main branch.

- **Patch**: A small change done to the existing commit before integrating.

- **Change Request**: A commit or a set of patches that consists of a bug fix, improvement, new feature to the existing software.

- **Review**: Feedback and inspection of the change request which consists of comments, approval ratings and patches.
· **Review Request**: An invitation sent to review the change request.

· **Owner**: The individual who creates the change request which consists of the new changes, modifications to the existing code.

· **Reviewer**: The individual who reviews the change request.

· **Comment**: The feedback on the change request by the reviewer.

· **Approval Rating**: The value given by the reviewers that specify whether the code is good enough to integrate or discard the changes.

· **Branches**: The versions of code created from the main source to perform code changes.

· **Machine Intelligence**: It is the concept of making machines/devices or software interact with environment in an intelligent manner and often help in automating progressive tasks [14].

### 3.1.3 Machine Learning

· **Feature** - A characteristic of the data being considered that can be measured, recorded, or calculated.

· **Model** - An artefact that captures the patterns which map the input features to the output or the target variable.

· **Supervised learning** - It is a learning approach where the desired input/output are mapped.

· **Unsupervised learning** - is a learning approach where the desired input-output are not mapped.

· **Underfitting** – A situation that arises when a model is not able to accurately map the inputs to the required outputs.

· **Overfitting** – A situation when a trained model generated is too specific to the training data and is unable to generalize to new data.

· **Root Mean Squared Error (RMSE)** – It is a measure of root of average of the squares of errors between estimated observations and actual values. Mathematically, it is the root of sum of the differences between the estimated and actual values divided by the total number of observations. It is a measure of quality for an estimator- the closer to zero it is, the better the estimator is.

· **Mean Absolute Error (MAE)** – It is a measure of errors between paired observations. Mathematically, it is the absolute difference between the paired observations divided by the total number of observations.
3.2 Code Review

- **Directed Acyclic Graph (DAG)** – As per the graph theory, a DAG is a graph that consists of a finite number of vertices and corresponding edges that are not only directed but also have no edges loop back to the same vertices.

- **Discretization** – It is a method of transforming continuous values into a set of discrete values.

- **Pickle** – A pickle is a file format that is native to python for storing objects. It is widely used to store machine learning models.

### 3.2 Code Review

Software code review is the practice used to find and identify problems in a given source code along with assuring that new code change has the expected quality [15, 16].

The process of conducting code reviews have been changing many times over the years. The use of code review process to find defects and flaws have been in use since the late 70’s [4]. The current day code review process has changed a lot when compared with the initial ways. When code review is mentioned later on in the report, it refers to the process of modern code review declared below.

As mentioned in section [1] Modern Code Review (MCR) is a lightweight and natural alternative to a conventional code review process, and it is practised widely in open source software projects and large organisations [10]. It is used to identify the defects, problems present in the existing code as well as the new changes, additions of the source code. Dedicated tools such as Gerrit and Github are used to make the modern code review process more manageable, comfortable and effective [8, 11]. Incorporating MCR in the software development process helps in achieving software with fewer bugs and easier maintenance [8]. It also helps in improving the code changes by preventing lousy code from being integrated with the existing software. MCR allows developers to monitor the code changes and give feedback to the developers before approving to integrate the suggested changes. MCR process is not concrete and slightly changes depending on the expertise of the reviewer [10].

Figure 3.1: Modern Code Review Process [1]
The normal work flow of a Modern Code Review process is:

- The developer creates some changes in the existing code, which may consist of bug fix or a new feature.

- The developer then creates a change request which makes him the owner of the change request and later sends review requests to other developers (reviewers) who may or may not accept the review request.

- If the reviewer has accepted the review request, they will be inspecting the change made by the owner and feedback is given in terms of comments and approval ratings.

- If the reviewer finds any problems in the change request, the owner has to do some patches and repeat the process.

- If the reviewers are convenient with the changes and approve them, then the changes are integrated into the main code.

There are many tools that are being used in the industry for conducting code review on a large scale, such as Helix, Github, GitLab, Gerrit and Review Board. In this research, we are going to focus on Gerrit since it is the tool used by the case company.

Gerrit is a popular web-based open-source code review management tool. It is integrated with Git, which is a Version Control System (VCS). VCS helps in easier management of projects, repositories along with the changes done to them. There are a significant number of existing studies which analyse code changes using Gerrit [7, 17, 19, 20]. Gerrit helps in viewing the change requests, adding reviewers to the change requests, inspecting the code change and reviewing with the help of comments and approval ratings. A change request is submitted and merged with the associated master branch if only a reviewer with the required privileges approves the code (+2 approval rating). Based on the reviews given by the reviewers, change requests can be appended with modifications and multiple iterations of the process can be performed before integrating the code change with a given master branch.

### 3.3 Bayesian Networks

#### 3.3.1 Overview

A Bayesian Network, as evident by its constituent words, is a network of nodes where the nodes are connected to each other so as to represent their “conditional” dependencies. Given the dependencies among variables are conditional, the network is modelled by a graph with directed edges and acyclic nature. This models the flow of information along with the cause-and-effect relationships among the nodes [21].
The following are the key points of emphasis for a Bayesian Network:

- Input information is subjective for a Bayesian Network.
- The basis for updating information being Bayes’ conditional probability.
- The difference between causal and evidence-based modes of reasoning.

Figure 3.2 represents an example of Bayesian Network and the way conditional probability between the network nodes are calculated.

Richard Neapolitan has put together and consolidated research in the field of representing uncertainty in artificial intelligence [22]. He also proved why should a Bayesian network be presented by a Directed Acyclic Graph (DAG) and how this DAG comes together as Bayesian Network. It was stated that if and only if the product conditional probabilities distributions is equal to the discrete probability distribution then the Bayesian Network is formed.

Figure 3.2 is a representation of Bayesian Network that includes all the factors and the relationship between them for grass being wet. The tables near each node represent the conditional probability of that particular factor. Conditional probability of a factor depends upon the relationship between other factors. The Bayesian Networks helps in creating a compact joint conditional probability distribution.

Here the factor that the grass is wet depends upon rain and sprinkler. The fact that it has rained or a sprinkler has been used depends upon the cloudy nature. Using all these relations a Bayesian Network is created and a joint probability distribution is calculated.
3.3.2 Inference

The inference in a Bayesian network is the evaluation of the joint probability of any particular set of values for each variable in the network. To calculate inference, we have a factorised form of the joint distribution using which we evaluate the variable using the provided conditional probabilities. If only the subset of variables is considered, we need to marginalise the factors which we are not going to use. In many cases when only the subset of variables are considered, it may result in an underflow. So in such a case we take the logarithm of that product, which is equivalent to adding up the individual logarithms of each term in the product.

The inference for the above example can be calculated by using the equation 3.1 where \( x \) is the chances of grass being wet because of sprinkler. \( e \) is the chances of it being cloudy. \( \alpha \) is a normalization constant and joint probability distribution is denoted by \( Y \).

\[
P(x|e) = \alpha \sum_{\forall y \in Y} P(x, e, Y)
\]  

(3.1)
Chapter 4

Related Work

With the popularity of Modern Code Reviews, many efforts have been made to automate this process and reduce user involvement as low as possible. In the efforts made to reach the goal, many problems and limitations have been encountered.

Reviewers need to have an in-depth understanding and experience related to the changes they are working with [11, 23]. If a reviewer does not have enough knowledge, it makes it difficult for him/her to review the changes. It also makes it difficult for him/her to prioritise review request among the other review requests he/she received, which consumes time.

Jeong et al. (2009) [24] has proposed a set of features which helps to find out if a given change request will be merged or abandoned. They focused on a set of keywords in the patch which are extracted from the bug reports. The limitations of this research are that the keywords are applicable to specific programming languages.

Similarly, in [25], the author has proposed a set of 12 features which helps in predicting whether a pull request will get merged. These features help to understand the characteristics of change requests.

Gousios et al. (2015) [13] has conducted a survey and gathered around 749 responses to find out the challenges faced by the integrators in code review processes. This survey helped in identifying that prioritisation of code review requests is one of the major tasks that developers are concerned. The survey responses led to the identification of some factors which reviewers consider while prioritising review requests, and it was also found that one out of five reviewers do not prioritise their code review requests.

Fan et al. (2018) [26] have tried to prioritise the reviewing tasks by predicting if merge requests will be accepted or abandoned. He has identified some factors and used them in performing the predictions. The study was conducted on meta-data collected from Gerrit systems of Eclipse, LibreOffice, and OpenStack.

Similar research with a different approach was done in [27]; they have used learn-to-rank models to predict whether the pull request will be merged or abandoned.
Van der Veen et al. (2015) [28] have tried to implement a pull request prioritisation tool for GitHub repositories. They tried to identify the factors that would help to prioritise the review requests. The prioritisation tool they developed was focused on supporting the project integrates rather than an individual reviewer. Machine learning models such as Random Forest and Naive Bayes were used to train and predict the expected outcomes.

Many studies [29, 30, 31] have used machine intelligence in their respective fields or problems related to prioritisation. This use of machine intelligence can be surmised as a positive approach for similar issues.

4.0.1 Limitations from Related Work

There have been some studies focusing on automatic prioritisation of the code review requests. The majority of the existing studies have been focusing on the identification of the factors and approaches. The limitations and research gaps identified in the existing literature review are as follows:

- Code review requests have been prioritised considering the review requests per repository instead of considering the review requests per reviewer. Considering code review requests of a repository does not help an individual reviewer, because it does not help the reviewers to focus on different projects/repositories at once.

- Code review requests are prioritised by predicting the merge status of a request or how fast the change request will be submitted. The prioritisations made in such a way may not be appropriate as there may be change requests which are bug fixes and require high attention and needs to be reviewed first.

- None of the identified approaches have been implemented and evaluated in a large-scale project or an organisation.

- The studies have not performed user evaluation of feasibility and usefulness of the implemented approach.

- The existing studies do not account the uncertainty factor while prioritising the review requests.
5.1 Research Method Selection

In the context of software engineering, Experiments [32], Surveys [33] and Case studies [34] are the widely used empirical methods. Case studies help increase knowledge about individual, organisational, and social phenomena [34]. Among the mentioned empirical methods, case study is considered to be more appropriate for this research. The reason for selecting case study over other empirical methods is that it allows for more relevance of the results as it involves analysis/introduction of things in a real uncontrolled environment [35].

Reasons for not selecting an experiment, survey are:

- An Experiment is not suitable for this research because all the factors impacting review requests prioritisation are not known forehand which restricts us from performing experiments [32]. It is also that experiments have high rigor but they lack relevance as the results are produced in a controlled often unrealistic environment [35].

- A Survey is avoided for this research as it would provide with better data collection but not allow us to explore a possible solution for the problem. To collect the required data, literature review, interviews and Gerrit mining will be performed.

**Literature review** can be considered as a step by step process of gathering and aggregating existing data from relevant papers to increase the knowledge in a research area [36]. Literature review helps in answering RQ1, as this method helps us finding the factors that impact the code review request prioritisation and the appropriate approach to develop a prioritisation tool from the existing literature study.

**Case study** is aimed at investigating the contemporary phenomenon of the natural context [34]. Case studies help in increasing knowledge about individual, organisational and social phenomena [34]. Case Studies are essential to evaluate the developed tools in a real/live large-scale projects. Even it helps us to analyse the problem in its real environment. In our research, all the questions are answered with the help of conducting a case study.
5.2 Research Design

In this section, the structure of the research and the way in which the research was conducted is described. An improvement case study has been implemented as the existing theories are tested in confirmatory studies [34]. The whole process in represented in the figure 5.1

[Diagram of the research design process]

5.2.1 Case and Unit of Analysis

In our research, the case company is Ericsson. The case product is Digital BSS (Business Support System). The development unit is located in 10 different centres of excellence such as Germany, Sweden, India, China, Canada, Brazil, and the USA, consisting of over 2500 employees. The teams use agile principles and practices in development (e.g., writing user stories, working in fixed-length sprints, stand-up meetings, and continuous integration). There are 4-6 developers in each team.

Charging is the first subsystem; it is responsible for online charging capability of the BSS (our unit of analysis). Around 30 teams have involved in the development of this subsystem. The teams are located in India and Sweden which practice agile principles in their development (e.g., writing user stories, working in sprints, stand-up meetings).

RMCA (Revenue Management Catalog Adapter) is the second subsystem; it is a technical product catalogue which is creating, governing and persisting business
configuration. There are eight teams involved in the development of this component. The teams are located in Sweden and Brazil. Similar to Charging, RMCA also practices agile principles in their development.

Factor Identification

The first step of the research was to identify the key factors from literature review that impact the prioritisation of code review requests (RQ1). Databases with credible peer-reviewed literature are searched to find the relevant research papers.

Along with the factors identified in the literature, semi-structured interviews were conducted with some of the experienced reviewers at Ericsson to identify/mention new factors (if any) along with giving an opinion on the identified factors via the literature review.

Factor Selection

The next step is to use the factors identified in step 1 and select which subset of factors to consider when implementing the code review request prioritisation tool. The selection of factors depends upon:

- The feedback given by the reviewers from the previous interviews.
- Required data is accessible in Gerrit or other sources.

If the preferred factors are unavailable or not suitable for the tool implementation, they must be reconsidered.

Approach Selection

Selected factors are used to find the possible approaches that are suitable to implement our prioritisation tool. With the help of existing literature suitable approaches are considered. Similar to factor selection, approach selection may also require reconsideration; it might be impossible to incorporate all factors on the preferred approach. The selected approach might require rework of the factor selection.

Tool Performance Evaluation

Predictive performance of the created tool is calculated that helps in answering RQ2. RMSE and MAE are the metrics used to measure the performance. These metrics will be compared between different approaches alongside evaluating the tool as well.

User tool Evaluation (Questionnaires)

After the tool has been deployed, a static evaluation was conducted in the form of questionnaires for the reviewers to answer the usefulness and feasibility questions about the tool after using it for an extensive period. This step will answer the RQ3.
5.2.2 Data Collection

The data required to perform and analyse this thesis work have been collected by performing literature reviews, semi-structured interviews, repository mining and feedback questionnaires. The different data sources are represented in Table 5.1.

| Data Source       | Description                                                                 | Questions |
|-------------------|-----------------------------------------------------------------------------|-----------|
| Literature review | Factors responsible to prioritise review requests identified in related and previous work. | RQ1       |
| Semi-structured interviews | Adds evidence and competence in factor identification from literature along with addition of new identified factors as well as limitations on the already discovered factors. | RQ1       |
| Repository mining | Extraction of data from Gerrit which is required to perform the exploratory analysis, This data is also useful for training the Bayesian Network | RQ2       |
| Performance indicators | Describes the effectiveness, efficiency and the speed of the tool | RQ2       |
| Feedback Questionnaire (Pineapple) | It is the user evaluation of the tool which helps find out the feasibility, usability and usefulness of the tool from the end user perspective. | RQ3       |

Table 5.1: Research data sources

Literature Review

The initial step of this research work is to perform a literature review to identify the key factors that impact the code review process and are key while prioritising the code review requests (RQ1).

A search string was formulated to find the relevant research papers from accessible databases such as Google Scholar, IEEE Xplore, and Scopus which help us find the appropriate factors and approaches. The results obtained from the literature reviews serve as a base for our future steps in the research. The relevant papers can be identified using the snowballing technique where pertinent new articles are found using the citations and references of already found related articles [37].

The literature has been found using the following search string along with the combination of peer-review filtering;
5.2. Research Design

("code review prioritisation" OR "pull request prioritisation" OR "code review prioritization" OR "pull request prioritization" OR "pull request" OR "automated code review" OR "review requests" OR "pull requests") AND "code review"

Along with the help of search string, snowballing was used to find more related papers.

Snowball Sampling

The initial set of papers is identified for starting, after which it is subjected to forward and backward snowballing. A paper which is being subjected to snowball sampling, is completely studied and only then it is included. A rollback is said to occur when a paper is included first and excluded later. These 2 actions are taken into consideration with concern to the sampling methodology of snowball sampling.

Backward Snowballing

Sampling done using the reference list of the included papers is termed as Backward Sampling. The papers that get included are based on the following criteria [38]:

- The title of the referenced paper and its relevance.
- The referenced context of the referenced paper and its relevance.
- The abstract of the referenced paper and how does it factor into the study.

Forward Snowballing

Sampling that includes new papers that attract a lot of citations and are also relevant to the topic of study. The papers that get included are based on the following criteria [38]:

- The title of the referenced paper and its relevance.
- The referenced context of the referenced paper and its relevance.
- The abstract of the referenced paper and how does it factor into the study.

The start set for the snowballing method consisted of one paper [28]. Three iterations have been performed and the selected articles after performing both backward and forward snowballing are 34. However it must be noted that some of the papers that were found during these iterations are redundant and some of them are not related to the study, so they are excluded.
Semi-Structured Interviews

After selecting the initial list of factors from the literature review, a total of sixteen developers/code reviewers at Ericsson were interviewed to help add more evidence to the existing list of factors or identify additional factors which impact code review prioritisation (RQ1).

The interview structure can be found in Table B.1 in Appendix B. All the interviews were conducted face-to-face on-site, the interviewee role and time took for each interview is presented in Table 5.2.

| Interviewee | Role            | Duration | Experience |
|-------------|-----------------|----------|------------|
| 1           | SW Architect    | 30min    | 10+ years  |
| 2           | SW Developer    | 25min    | 1+ years   |
| 3           | SW Architect    | 35min    | 12+ years  |
| 4           | SW Developer    | 25min    | 1+ years   |
| 5           | SW Developer    | 35min    | 1+ years   |
| 6           | SW Developer    | 20min    | 1+ years   |
| 7           | SW Developer    | 25min    | 4+ years   |
| 8           | SW Developer    | 30min    | 3+ years   |
| 9           | SW Developer    | 25min    | 2+ years   |
| 10          | SW Developer    | 25min    | 1+ years   |
| 11          | SW Developer    | 30min    | 4+ years   |
| 12          | SW Developer    | 35min    | 4+ years   |
| 13          | SW Developer    | 30min    | 6+ years   |
| 14          | SW Developer    | 25min    | 5+ years   |
| 15          | SW Developer    | 25min    | 5+ years   |
| 16          | SW Architect    | 35min    | 12+ years  |

Table 5.2: Description of semi-structured interviewees

Repository Mining

With the help of Gerrit API, repository mining was performed, which provided useful data that helped in conducting exploratory analysis. The data includes changes, patches, comments, size, age and reviews. The range of the data extraction considered was the last two years so that the test data would be closer to the current usage data.

Pineapple Feedback Questionnaire

After Pineapple is tested and deployed on a large scale, a feedback questionnaire regarding Pineapple’s usability, reliability, and feasibility (RQ3) is provided to Pineap-
ple users in the tool itself. A total of 24 reviewers have answered the pineapple feedback questionnaire. The structure of the Pineapple feedback questionnaire is presented in Table C.1 in Appendix C.

5.2.3 Data Analysis

The analysis of the data collected from literature review, semi-structured interviews, repository mining and feedback questionnaires is explained in this section.

Literature Review

The factors identified in the literature review were compiled and examined, along with some statistical analysis. The list is then filtered to create an initial list of factors which were used for discussion in the interviews (RQ1). The final list of factors included the factors that were deemed to be important by the interviewees and are accessible and collectable from Gerrit or any other source, and are also relevant to the research.

Semi-Structured Interviews

The following steps have been conducted to analyse the data collected from interviews:

- The initial list of factors were shown to the interviewees to select the factors they feel may impact review process.
- The initial factor selection is complimented with the help of the feedback provided by the interviewees.
- The interviewees were asked to distribute a cumulative of 100 points among the selected factors which they think that impact code prioritisation.
- The interviewees were also asked to provide the relation between the selected factors which helps in building the Bayesian network.
- Later the factors are considered upon the total number of points assigned to them along with their accessibility and relevance.

The process of the literature review and interviews can be seen in Figure 5.1. The factors, therefore, help in selecting an adequate approach as well as build a stable and efficient Bayesian network. The reason for using point system in the interviews rather than traditional ranking system is to provide better difference between ranks of the identified factors.

Repository Mining

After collecting the data from the repositories of two subsystems: Charging and RMCA, analysis has been performed on the data extracted. A total of 110 repositories are used for repository mining. The majority of the analysis done is represented
in the Tables and Figures/graphs. The main aim of this analysis is to find useful patterns from the obtained data, which may help in provided other solutions or add support to the selected approach. The Table 5.3 is an overview of both the subsystems and provides the stats related to the repositories present in both the subsystems.

| Product | Changes | Activity | Developers |
|---------|---------|----------|------------|
| Both    | 92.4K   | 785K     | 445        |
| Charging| 70.3K   | 512.2K   | 337        |
| RMCA    | 22.1K   | 272.8K   | 173        |

Table 5.3: Scope of data considered for training and development

The data represented has the interval from 1\textsuperscript{st} January 2017 to 31\textsuperscript{st} March 2020

In the above table, changes refer to the code review requests which have been created that are either merged or abandoned by the reviewers. The changes help us to collect the data of desired factors from RQ1. Activity is all the patches and comments created in each particular change. The messages generated by bots and other auto-generated scripts are excluded. The data in activity helps us obtain the reviews given by the developers and also the reviews provided by the automated tests.

The developer count in Table [5.3] do not add up to the sum of both the subsystems, this is because some developers have worked in both projects during the analysed time interval. It is the count of developers who have worked on the project at the given interval of time.

**Developer Activity**

Developer activity helps us identify their impact on both changes and activity as they also indirectly impact the prioritisation process. Figure 5.2 helps to provide an overview of the active developers’ in both Charging and RMCA. The developer is considered active if he/she has requested any change request or reviewed any code change in a time span of one month. A time interval of one month is considered to avoid the inconsistencies caused due to vacations and holidays.

Since Charging is a bigger product than RMCA, it can be observed that the number of developers in Charging are more compared to RMCA. We can also summarise from the graph that the trend is stabilised and similar between both the products.

**Changes**

The data provided by the Gerrit changes are highly helpful and used by Pineapple. In Figure 5.3 the number of newly created changes per month is displayed. When comparing Figure 5.2 with Figure 5.3 we can identify that the number of active developers correlates with the amount of created changes. However, the size, age of the changes cannot be related.
5.2. Research Design

Figure 5.2: Active Developers involving in creating new change requests

The graph shows similarities between the curve for both Charging and RMCA which means that the development trend and the way of working may be similar among them. It can also be observed that there are some spikes/dips in the graph, which may be due to more developers working on important period tasks and also because of vacations and holidays. The spiking and fluctuation in the graph may also occur due to sprint periods, and possibly other factors.

Figure 5.3: Newly created change requests per month

Statistical Data Analysis

In this section, a mathematical analysis of data for the factors which we use for training and developing the Bayesian Network is shown. This analysis helps us identify the patterns and also helps in discretising the values. The reason and usefulness of discretisation were explained while describing the Bayesian Network in the Section 6.1.2.
The Tables D.1, D.2, D.3 show the statistical values mean, 25% percentile and 75% percentile for the factors age, size and number of patches of a given change respectively.

The statistical analysis is done for a time interval of 6 months starting 1st January 2017 to 30th March 2020. An interval of six months has been chosen to see if the patterns in the values remain the same, or do they vary upon a time. It is essential for the mathematical statistics to be similar for both old and new data because using the same discretisation bin values stored for old data even for new data with different metrics may not produce effective prediction results.

The age values in Table D.1 are in minutes, the size values in Table D.2 are lines of code, and the number of patches is the versions in each change with different versions and changes. As the tables suggest all the statistical measurements are similar, and a slight difference can be seen in age values of the latest interval of 2020 that maybe because of the smaller interval and fewer changes as the interval consists of only three months.

In Table D.3 as the range of number of patches is small we do not see much difference in the statistical metrics.

**Performance Evaluation**

The predictive performance will also be performed on this data alongside comparing the difference between the predicted merge status to the actual merge status of a particular change.

After the data is mined, data analysis is performed to find the patterns and build a proper Bayesian network or other suitable approaches. To calculate the predictive performance of Pineapple (RQ2), MSE and MAE are used.

The main reason for not using accuracy or f1 score metrics to compare the approaches is that these metrics mainly depend on the classification part of an issue, but in our case, we are trying to predict the probability of a given change instead of just trying to predict whether the change will be merged or abandoned. In this case, using metrics which measure the rate of error and error margin is more suitable; this is the reason for selecting RMSE and MAE over other metrics. The lower the MSE and MAE scores the better is the prediction of the approach, i.e. better performance. Alongside performance evaluation using some python scripts the response time and prioritisation speed of the tool are also calculated to evaluate the tool.

The equations shown below help us calculate the RMSE and MAE metrics.

\[
RMSE_{score} = \sqrt{\frac{\sum_{i=1}^{n} (Predicted_i - Actual_i)^2}{n}} \quad (5.1)
\]
MAE score = \frac{1}{n} \sum_{i=1}^{n} |Actual_i - Predicted_i| \tag{5.2}

The result obtained for both the metrics RMSE and MAE range between 0.0 and 1.0, where 1.0 indicates that the predicted value is in complete contrast compared to the actual value. Similarly, 0.0 indicates the best-case scenario. This makes clear that lower the metric values better is the prediction results.

Pineapple feedback Questionnaire

The data collected from the feedback questionnaire is plotted and analysed to help answer RQ3. As the questions use Likert items, it is suitable to plot pie charts to evaluate the results.

While asking the end-users about the tool, we have also provided a text field to provide any suggestions as a good opportunity to improve Pineapple. The pie chart distribution helps us visualise on how well the developers agree with certain statements; it is available in Figure A.1 in Appendix A, Table C.1 in Appendix C.
Pineapple Overview

Pineapple is a code review prioritisation tool that uses a Bayesian Network to perform recommendations. Pineapple consists of five microservices, which are using docker containers while deployment. Each service runs in an individual Docker container.

The current version of the tool is used through a web-based GUI (see Figure A.1 A.2 in Appendix A). Pineapple can also be used via REST API calls. The services were designed to be semi-independent; each microservice has a specialised area of responsibility.

Figure 6.1: Pineapple Overview

Figure 6.1 shows the overall process flow of the tool, where a reviewer provides his user-id. Data is gathered from Gerrit for that particular user using which Bayesian Network prioritises code review changes. The results are later to the reviewer.
6.1 Architecture

The architecture of Pineapple is presented in Figure 6.2. The lines represent major data channels along with the direction and its response data. The tool consists of 5 microservices, namely Core, ETL, ML, Prioritiser, Front-End. The architecture also provides information on how the Bayesian Network is stored in a pickle and loaded into the prioritiser.

6.1.1 ETL

The ETL service stands for Extraction, Transformation, and Loading of data from Gerrit to a Database. It is the only service with a direct connection to Gerrit. As the name suggests, ETL extracts raw data in Json format from Gerrit using API calls. It is then transformed into tables later loaded into the database. It extracts data once per week, and the newly extracted data is used to retrain the Bayesian Network. When the weekly extraction is done, ETL sends a signal to the core that new data is available in the database; this prompts the core to signal the ML to retrain the Bayesian Network. The ETL service is also responsible for extracting the user meta-data when the user passes his user-id as input to prioritise his review requests from Gerrit.

6.1.2 ML

The ML (Machine Learning) service loads the data from the database, which is extracted and transformed by the ETL. It uses that data and trains the Bayesian Network.
Network, which is used to prioritise the code review requests. The implementation and performance of the developed Bayesian Network is discussed below.

**Bayesian Network**

In this section, the implementation and performance of the Bayesian Network are explained. The factors identified and finalised after answering RQ1 were used to build the Bayesian Network. A Bayesian Network consists of nodes connected to each other upon dependency and consist of conditional probability distribution among the nodes. Each node in the Bayesian Network represents each factor that affects code review prioritisation. The final architecture of the Bayesian Network used after testing many drafts is shown in Figure 6.3.

![Figure 6.3: Bayesian Network Architecture](image)

The nodes in Figure 6.3 represent the factors identified in RQ1, which is explained in the section 7.1.3. The arrows between the factors represent the dependency between the factors. The "Change Status" is the node which represents the merge probability of the change and is the predicting variable of the Bayesian Network.

The age, size and number of patches are the three variables which were discretised from continuous values to discrete values to make predictions more efficient. The variable values were categorised into three different categories, and the binning values are stored in the pickle along with the model. These binning intervals are again used to categorise the metadata which is used to predict merge probability.

The nodes merge-conflict and change-type are independent variables which are not part of the Bayesian Network. They are used to sort the changes upon their importance, i.e. a change with merge conflict will be given lower rank compared...
with its counterpart. Similarly, the type of the change also impacts the importance of the review request.

Once the BN is trained, it is stored in a pickle file in a docker volume which is loaded by Prioritiser service.

When the new BN is trained, a new pickle file replaces the existing one with the newly trained model. The Bayesian Network is retrained once per week along with the ETL extraction. The trained Bayesian Network is capable of calculating the merge probability of a given code review request, more information regarding the Bayesian Network is provided in the Section 6.1.2

6.1.3 Prioritiser

The Prioritiser service loads the trained Bayesian Network from the pickle file (created by the ML service) stored in a docker volume and performs prioritisation on the open code review requests for the tool user.

Once the BN is loaded, it requests the ETL service to extract meta-data (open code review requests) associated with the tool user through the Core. The structure of this data is the same as the structure of the data used to build the Bayesian Network as shown in the Table E.1 in Appendix E. Once the data is provided, predictions are made by the Bayesian Network, and the probability of the open code review request being merged is generated.

The review requests are then sorted upon the probability values, the ones with higher merge probability are given a better rank. Higher the probability, higher is the rank of a code review request in the prioritised list as it was desired by the majority of the developers while conducting interviews. There is an abstraction of this procedure in Figure 6.1

The probability lies between 0 to 1. where 0 indicates that the chances of change request being merged is zero and 1 indicates that the change request has high chances of being merged. Based upon these probability values the change review requests are prioritised.

6.1.4 Front-End

The Front-end service, as the name suggests, is the front-end of Pineapple and allows the users to request Pineapple to prioritise their respective open code review requests. The front-end service also consists of the feedback questionnaire shown in Figure A.1 in Appendix A. The responses for the questionnaire provided by the reviewers act as static validation for the tool and is sent to the core which stores them in the database.
6.2 Development

6.1.5 Core

The Core service acts as the backbone for Pineapple. It acts as a messenger to communicate and binds all the services together. All the data and message transfer between the services is done through the Core. It also handles the scheduler, which triggers the ETL service to automatically perform their tasks weekly. Additional task done by core is to store the responses for feedback questionnaire provided by the developers in the database.

6.2 Development

Pineapple was built using Python and deployed in a dockerised environment consisting of Docker containers on Ericsson’s production servers. After deployment, the developers/reviewers were able to access the Pineapple and provide us with feedback via the tool itself. The feedback was collected through the tools graphical web interface, that can be seen in Figure A.1 in Appendix A. Pineapple is developed using Python (version 3.6) and Flask. A lot of the data handling is done using pandas which makes the ETL and ML services compute-heavy parts of the system and have the possibility of being optimised in the future.
Chapter 7

Results and Analysis

In this chapter, exploratory analysis of the mined data, results obtained for defining the research questions along with analysing the obtained results. The motivation for the selection of factors and the relationship between the factors are also explained along with the comparison of performance between Bayesian Networks and existing studies.

7.1 RQ1 - Review Factors

The results associated with RQ1 are presented in this section. To clearly explain the process of factor selection, a step by step process is used below. It consists of findings and selection of factors, along with motivation of selecting the factors used in Bayesian Network.

7.1.1 Findings

In Table 7.1 the information related to the result found from the literature review, and the interviews are presented. As the data in the table shows, factors that are most prominent and mostly mentioned in both literature and interviews are the size of the change, age of the change, change type (Keywords), automated tests, peer reviews and merge conflicts. The table consists of the number of times the factor has been mentioned in literature as well as interviews.

While conducting interviews, the interviewees have been provided with 100 points to distribute among the selected factors upon their level of importance. The cumulative points obtained by each factor are also represented in Table 7.1. It can be observed that Keywords and automated tests have received the highest number of points which indicates the higher importance given by the developers to these factors.

7.1.2 Factor Selection

To make the factor selection process easier a set of criteria was created. Using these criteria the factors were filtered out and non-viable factors which were to complex/unfit to implement were ignored. The criteria considered are defined below:
Chapter 7. Results and Analysis

| Factor               | Description                                                                 | Int. Freq. | Lit. Freq. | Int. Points | Literature Source |
|----------------------|-----------------------------------------------------------------------------|------------|------------|-------------|-------------------|
| Size of the change   | Number of add, del lines of code                                            | 13         | 4          | 140         | [23, 13, 27, 26]  |
| Files changed        | Number of files added or deleted                                            | 1          | 4          | 8           | [23, 13, 27, 26]  |
| Age                  | Age of the change                                                           | 11         | 2          | 135         | [23, 13]          |
| Contain Keywords     | Keywords such as TR, bug fix and feature in commit messages                 | 16         | 2          | 544         | [23, 13]          |
| Tests                | Automated tests done by bots and CI/CD tools                               | 16         | 3          | 322         | [23, 13, 27]      |
| Developer stats      | Contribution and acceptance rate along with activeness                      | 5          | 4          | 72          | [23, 13, 27]      |
| Repository stats     | Activeness of repository                                                    | 6          | 1          | 83          | [23]              |
| Number of revisions  | Number of patches in a change request                                       | 9          | 3          | 86          | [23, 13, 27]      |
| Peer Review          | Review given by another reviewer for a particular change                    | 10         | -          | 147         | -                 |
| Merge Conflicts      | Code conflicts faced while merging code changes                              | 16         | -          | 160         | -                 |

Table 7.1: Factors identified from Literature Review and Interviews

C1: The factor is discussed in Literature study

C2: The factor is discussed in Interviews

C3: The factor can be obtained using automated extraction

C4: The complexity of factor extraction is bearable

C5: The complexity of factor data analysis is bearable

C6: The obtained data is reliable

In Table 7.2, the factors identified from both literature and interviewed are evaluated using the criteria mentioned above. This helps us to provide motivation for the selection of factors done and also explains why other factors are not suitable for our development.

In the table above the factors with value 'x' for each criteria indicates a positive answer whereas the blank spaces indicate negative response.

### 7.1.3 Implemented Factors

The motivation for selecting the identified factors that have been implemented in the Bayesian Network of Pineapple are described below:
Table 7.2: Factor Selection Criteria

| Factor                      | C1 | C2 | C3 | C4 | C5 |
|-----------------------------|----|----|----|----|----|
| Size of the change          | x  | x  | x  | x  | x  |
| Files changed               | x  | x  | x  | x  | x  |
| Age                         | x  | x  | x  | x  | x  |
| Contain Keywords            | x  |    | x  |    |    |
| Tests                       | x  | x  | x  | x  |    |
| Developer stats             | x  |    |    | x  | x  |
| Repository stats            | x  |    |    | x  | x  |
| Number of revisions         | x  | x  | x  | x  | x  |
| Peer Review                 | x  |    | x  | x  | x  |
| Merge Conflicts             | x  | x  | x  | x  | x  |

**Keywords** - The keywords which are used to label a change or classify a change are given higher priority by the developers. This pattern can be seen by analysing the table 7.1. All the 16 interviewees have mentioned the factor and considered it to be one of the most important factors. This factor is not directly implemented in the Bayesian Network but is used to sort the changes depending upon their own tags/value itself. Existence of keywords has received the highest number of points in the interviews. The keywords are extracted from the commit messages and change messages using regular expressions.

**Tests** - The existence of automated tests has been considered essential because the verdicts provided by these tests have some value, and they indirectly perform the code review process at the primary level. So the automated tests impact the merge status of a change and have also been mentioned by all the interviewees. The automated test verdict also indicates to the reviewer whether they need to focus on a particular change or not. The automated test values of a change are extracted using the Gerrit API.

**Size** - The size of a change is considered important as the exploratory analysis and the domain knowledge of the interviewees indicated that the changes with large sizes are often defect prone and have high chances of failing the automated tests. It also shows that developers like to review smaller changes when compared to bigger ones. The size of the changes is extracted using the Gerrit API.

**Age** - The age of a given change is also an essential factor with respective the number of times it has been mentioned in both Literature and Interviews. The exploratory study shows that the changes with larger age times tend to be abandoned for the majority part or left unaddressed. This indicates that including the age in the Bayesian Network would help find the relation between other factors as the age of a change is related to the number of revisions/patches and merge status. The age of the change is extracted using the Gerrit API where it is the measurement in minutes calculated from the time of creation to the time of prioritisation request.
Peer Review - When a developer is reviewing a change request it is important for him to know whether the change request has already been reviewed by any other developer and has he provided any feedback on the particular change. If change has a verdict of -1 given by another reviewer, it means that the committer has to make some changes already before getting approved, so the reviewer can skip that change at the moment. All these scenarios were considered and discussed with the interviewees which further motivated to select the factor.

Number of Revisions - It is clear that the changes which had issues in their initial draft have tried to be rectified, which results in the creation of a new revision. This means the occurrence of new patches impacts the code review result. As the table 7.1 shows, number of patches has been considered to be important and has been mentioned many times in both Literature and Interviews. This motivated in including the factor – number of patches/revisions to the Bayesian Network.

Merge Conflicts - Inclusion of this factor is straight forward, as the changes having merge conflicts needs to be revised as they cannot be merged. So these changes need to be given lower priority. Similar to keywords, merge conflict is not included in the Bayesian Network but is used to sort the changes depending upon their merge conflict status.

7.1.4 Excluded Factors

The motivation for not selecting the identified factors in Pineapple is explained below:

Number of Files - The reason for not considering the number of files modified for a change is that there might be changes which have many files changes and has small size and some changes where the only one file is changed, but the size of the change is big. This makes the factor not reliable to prioritise the changes depending upon the number of files modified. To get the size of the change, we rather use the sum of insertions and deletions of lines of code for a particular change.

Developer stats - The contribution rate and change the acceptance rate of a particular developer come under developer stats. The main motivation for not including developer stats is that even though the person with better experience has high chances of having the code being able to merge, it doesn’t make the change important to be reviewed first and have a higher prioritisation rank.

Repository stats - The activeness of a repository is considered to be the repository stats. It is the rate at which new changes are created in a repository frequently. The motivation for excluding this factor is that when a change from a rarely used repository is created and needs to be reviewed; if that particular change is given less priority depending upon its activity status and not reviewed quickly, it may serve as a bottleneck for that particular repository. This made not selecting and implementing the repository stats factor in Pineapple.
7.2 RQ2 - Pineapple Performance Evaluation

In this Section, we present the results related to the performance evaluation of Pineapple (RQ2), along with the comparison of the performance metrics of the Bayesian Network and Pineapple with the existing and other approaches.

7.2.1 Model Evaluation

The performance metrics of the Bayesian Network are shown in the Table 7.3 which consists of the metrics accuracy, RMSE, MAE. The values in the table shows that the performance of the Bayesian Network is competent. The metrics in Table 7.3 are a cut above when compared to the metrics provided in existing studies [26]. The comparison of the metrics with the existing and other approaches is discussed in the Section 7.2.1.

| Metric  | Value   |
|---------|---------|
| Accuracy| 88.3%   |
| RMSE    | 0.21    |
| MAE     | 0.15    |

Table 7.3: Bayesian Network Performance Metrics

To evaluate the model performance we use both Root Mean Square Error (RMSE) and Mean Average Error (MAE) metrics. The mathematical equation used to calculate RMSE and MAE are shown in the Equations 5.1, 5.2.

The main purpose of these metrics is to know how much deviation is observed in the predicted values from the actual values on an average. The reason for selecting these metrics is that we are trying to calculate the probability of change merge status, which is not a classification problem by itself. To know the model performance, we need to understand how the predicted results digress when compared to the actual one.

The metrics were calculated by performing k-folds cross-validation of sample size five, which splits the data randomly and performs both training and testing. The values provided in the table 7.4 represent the mean of the values obtained after five k-fold iterations.

The different approaches used to implement the ML model were Bayesian Network, Random forest, XGBoost, Logistic Regression and KNN. The reason for selecting Bayesian Network is that BN accounts for uncertainty while predicting the merge probability of a given change request which is not done by any other existing approaches. The table also gives an overview and comparison between the metrics obtained with different approaches.
Chapter 7. Results and Analysis

Table 7.4: Comparison of Model Evaluation Metrics

| Model                | RMSE | MAE |
|----------------------|------|-----|
| Bayesian Network     | 0.21 | 0.15|
| Random Forest        | 0.30 | 0.25|
| XGBoost              | 0.26 | 0.21|
| Logistic Regression  | 0.29 | 0.23|
| KNN algorithm        | 0.31 | 0.26|

As the values in the table suggest the RMSE and MAE scores for the Bayesian network are lower compared to the other approaches, i.e., that the Bayesian Network performs better than the other approaches. According to existing literature, Random Forest was more predominantly used to prioritise review requests, but the evaluation shows when we consider probability rates and less margin for error Bayesian Network is a better approach.

7.2.2 Response Time Evaluation

In this section, we calculate the response time of the prioritisation request done by the end-user. The response time calculated by using a test script is written in Python that uses the requests package.

The requests package has an inbuilt elapsed time function which provides the time taken to get back data for a given response request. The testing script may create additional overhead to response time, but it can be considered negligible for this purpose. In order to avoid fluctuations in the response time for a particular prioritisation request caused by traffic and server irregularities, the response time was collected over ten prioritisation requests.

| Batch | Response Time |
|-------|---------------|
| 1     | 2.140885      |
| 2     | 1.901708      |
| 3     | 2.359215      |
| 4     | 1.974151      |
| 5     | 1.915349      |
| 6     | 2.094808      |
| 7     | 1.822021      |
| 8     | 2.049487      |
| 9     | 2.036489      |
| 10    | 1.894578      |
| Average | 2.018869   |

Table 7.5: Response time for prioritisation requests
The results of the test are shown in Table 7.5 along with the average and standard deviation of the response times.

The response time results obtained were calculated for a user with around fifty open changes, and the average response time of the prioritisation requests is about 2 seconds which considered to be good. But from the data analysis, it was found that the average number of open changes for a particular developer is between twenty and thirty, which means the response time may be quicker in real-time. The reason for selecting a user with fifty open changes is to push the tool to some extreme scenarios which is more than normal.

7.3 RQ3 - Pineapple Feasibility and User Evaluation

In this section, we report the results related to Pineapple’s feasibility, usefulness and usability of Pineapple using the feedback provided by the developers. The feedback questions are provided in Pineapple, and the users can provide feedback right after they review the prioritised requests. The questions used for evaluation can be found in the Table C.1 in Appendix C.

Figure 7.1 shows the result related to question one of the questionnaire (usefulness), where the developers were asked whether Pineapple is useful to prioritise review requests. The results show that 60.9% of people agree and 21.7% of people strongly agree i.e, 82.6% agree that Pineapple is useful to prioritise review requests. The Pie chart shows that the majority of developers are positive about the usefulness of the tool and think that tool has the potential to help the reviewers.

Figure 7.1: Q1: Pineapple is useful to prioritise review requests.

Figure 7.2 shows the result related to question two of the questionnaire (usability), where the developers were asked whether Pineapple is easy to use and understand. 56.5% of people agree, and 39.1% of people strongly agree i.e, 95.6% agree that Pineapple is easy to use and understand, which is a huge portion of positive re-
responses, and there were 4.3% neutral responses. There were completely zero negative responses to this question.

Figure 7.2: Q2: Pineapple is easy to use and understand.

Figure 7.3 shows the result related to question three of the questionnaire, where the developers were asked whether Pineapple helps in decreasing the overall lead time of a review process. The responses show 52.2% of people agree, 26.1% of people are neutral and 13% people disagree with strongly agree and strongly disagree have 4.3% each. This does explain that majority of the people are positive and some of the developers were neutral with very less percent of people differing.

Figure 7.3: Q3: Pineapple helps in decreasing the lead time of review process.

Figure 7.4 shows the result related to question four of the questionnaire, where the developers answer whether Pineapple helps inexperienced reviewers in performing code review requests. The response obtained for this question was that 34.8% of people agree, 26.1% of people are neutral, and 21.7% of people disagreed. It had an equal distribution of solution compared to other questions, but there has been a total of 47.8% of positive responses using which we can say that it may help inexperienced
7.3. RQ3 - Pineapple Feasibility and User Evaluation

Figure 7.4: Q4: Pineapple helps inexperienced developers in performing code reviews.

Figure 7.5 shows the result related to question five of the questionnaire (reliability), where the developers answer whether the prioritisations done by Pineapple are reliable. The graph shows that 47.8% of people agree along with 13% of people strongly agreeing while there has been 17.4% of neutral and negative response each i.e, 60.8% agree that prioritisations done by Pineapple are reliable. This shows that developers feel Pineapple is accurate enough while prioritising the code review requests.

Figure 7.5: Q5: Prioritisation done by Pineapple are reliable.

All the five graphs that are shown above help us motivate and defend the factors, approach selected to develop Pineapple alongside providing a static evaluation of the tool. After analysing the graphs, it is clear that user evaluation of Pineapple has been positive and shows high feasibility, usability, and usefulness of the tool. Even
though the results are not perfect and some feedback and suggestions were received to improve the tool, the current results are significant for the most part.
Chapter 8

Discussion

The results obtained from this research indicate that Pineapple could be a helpful code review prioritization tool which helps code reviewers in prioritising the open code review requests already assigned to a particular developer, with positive responses from the users. When compared with other existing studies, Pineapple solves many limitations explained in Chapter 4. It is a bit difficult to compare Pineapple with existing literature because none of them was tested in a large and real-time environment. In Pineapple’s case, it was deployed and tested in an industrial environment, and many developers have used the tool. In this research we compared Bayesian network with models using ML approaches used in other studies. Although it was not an exact comparison, it allows for having a good baseline in relation to how good Pineapple performs.

One of the main contributions of this research work alongside factors identification is the development and evaluation of the tool in a real live environment which brings many factors that impact review process and also helps have the possibility of gaining knowledge on new challenges while performing code review requests and prioritising them. This is because while working in a real-time development environment, there are possibilities of finding new ways which we may not consider during development and local testing.

While conducting interviews, it was observed that the majority of the software developers were very much interested in the idea and wanted to use the tool whenever available. The input was obtained from the interviews that they feel the tool to be helpful and ease the task of code reviewing. After interviewing 4-5 developers, it was clear what the developers wanted, and after 9-10 interviews, the data gathered through the interviews have been saturated and was similar to all the different interviewees. Providing the factors found in the literature to the interviewees and gathering their opinion on these factors helped to validate the identified factors which add motivation to the factors. The interviews also helped in finding the relationship between the factors which helped in building the Bayesian Network; Otherwise, it would be difficult to build the network from scratch without having any data evidence or motivation. It was also found through interviews that people often tend to invite people to review their change personally, but these factors can be handled as they bypass the purpose of the research and the tool.
The results indicate that using Bayesian Network for implementing the ML part of Pineapple is more effective and accurate compared to existing approaches of Random Forest and other approaches considering the validation done using the RMSE and MAE metrics. The main reason for not using accuracy or f1 score metrics to compare the approaches is that these metrics mainly depend on the classification part of an issue, but in our case, we are trying to predict the probability of a given change instead of just trying to predict whether the change will be merged or abandoned. In this case, using metrics which measure the rate of error and error margin is more suitable; this is the reason for selecting RMSE and MAE over other metrics. The Bayesian Network developed uses the data provided and calculates the conditional probability distribution (cpd) between the factors and uses the calculated cpd’s to predict the probability of new changes. Initially, the automated tests were considered and related to other factors individually, but it delayed the training process a lot which made us consider the minimum value of all the automated tests as a single factor instead.

Considering the existing literature methods and approaches, there were many limitations such as the code reviews where not prioritised on a reviewer level including multiple repositories instead on a repository level. The predictions were made in such a way that changes with less merge time were prioritised first. The prioritisations made in such a way may not be appropriate as there may be change requests which are bug fixes and require high time/attention and needs to be reviewed first. It is also unknown that the existing approaches work in a real-time environment because they were not evaluated in that manner.

From this research, the way of using probabilistic prioritisation, statistical validation and performing static evaluation for the developed tool provides implication for research. Considering implications for industry, the tool is deployed and being used by Ericsson developers. The static validation showed that Pineapple has the potential for changing their ways of working in relation to how they review code, leading to improvements.

While implementing the GUI for Pineapple, the URL of the Gerrit change is linked to the change id of the change which helps the user to redirect to the Gerrit review page from pineapple instead of manually opening the changes. It would be a better future solution to integrate Pineapple in Gerrit as a plugin which makes it easier for the user to use the functionality. Currently, Pineapple is designed to work with Gerrit, to make it work with other VCS the ETL and the ML services needs to be changed, and the way the data handled to train and predict should be managed. But the fact that developers have responded with 95% positive response for evaluation question where they feel Pineapple is easy to use and understand it feel the GUI implementation efficient. The reason for having a separate GUI was to able to evaluate the tool using the feedback questions and user feedback.
While trying to visualise the responses given by the developers, it was observed that the majority of the responses were positive, and some of them were neutral. The only question which had a significant amount of negative responses was for question 4, where the developers answered whether Pineapple helps inexperience developers to review code review requests. This may be the case because when the tool helps them to know which change to review first, they still need to have knowledge about the change request they are going to review. Apart from that, the responses for the other questions have been majority positive, which indicates that the factor, approach selection was adequate.

The implementation and deployment of Pineapple were done in a docker environment where each of the services were deployed in individual containers. This type of orchestration helps in scaling the services when there is high load on any particular service. This docker deployment is done using docker-compose, and the trained Bayesian Network is stored in the form of pickles in docker volumes. The reason to use docker volumes is to have persistent data and use it for later use when required.
Chapter 9

Threats to Validity

The Validity Threats of the study is Discussed in this Chapter. As Suggested by Runeson and Host [34] the threats are categorized into reliability, internal validity, external validity and, construct validity.

9.1 Reliability

The threat of reliability concerns the repeatability of the study by researchers who are in possession of same or similar results [34]. This threat is handled by detailing and defining the methodology in as much detail as possible. However, qualitative data forms a critical part of the study which has been collected by means of questionnaires and interviews. Therefore the data which forms the basis for making key decisions that shaped the tool, and the feedback obtained could be affected by having a different set of participants.

Interviewing different developers will inevitably generate different ideas and responses but so will actions of the interviewing researcher in the interview. The inputs offered by the interviewees could also impacted by the exact clarifications the researches do when asked. One possible problem is that the questions could have been misinterpreted which was countered by clarifications and pre-interview information about the tool.

The user feedback collected during testing may vary depending on the type of users requested and their usage of the tool. However, due to the larger amount of involved developers (with regard to interviews and questionnaires), the results ought to come out looking balanced.

The tool usage experience of a user invariably influences the responses which would be recorded. Consequently, the answers could vary depending on how much the tool has been used and for what situations. The experience of a developer is also a factor which has been balanced out by asking multiple developers with different experience levels in the study.

In terms of repeatability, firstly an attempt was made to include a broad selection of developers in all aspects based on qualitative data, to prevent too narrowly-scoped groups of select streams. Also all questions were thoroughly inspected to ensure that they were unambiguous and not open-ended.
9.2 Internal validity

Internal validity defines the impact of factors on the other investigated factors that might affect the validation result [34]. The internal validity threats depend upon factor selection that may include biased answers. To mitigate the risk of bias, factor selection was performed with interviewees with different components and different positions. The information gained from the interviews adds more knowledge to the literature findings which indeed provides less bias.

Since all the tool was open to all developers who were working on the affected repositories, the results obtained would be dependent upon which developers opted to use the tool. It is likely that the developers who determine to use the tool have a positive attitude toward it due to the interest shown. The mitigation of the problem can be done by having some managers who take help of their team to try the tool.

Several validation techniques that includes a partially different set of participants Minimizes the impact of bias on consolidated results

9.3 External validity

External validity defines the extent to which the findings can be generalized and how interested they are to people outside of the Study performed [34]. Since the study performed is based on development of code review recommendation tool, it seems to have more interest in the approach used in the study performed As the validation methods (questionnaires) that are used are not seen before in related works and may help in validating further recommendation tools that involve code review system. Even though the study is based on Ericsson and the conclusions are based on MCR, alot of conclusions are relatable to other code review environments. Most likely other Large-scale projects can be built or improved with the similar approach.

9.4 Construct Validity

Construct validity defines the extent of how efficient the operational methods answer the research questions [34]. A viable threat is the solutions to the interviews that rely on how the interviewees interpreted the questions. The threats were discussed by giving an initial information of the tool’s goal and how the information should be used.

The set of respondents were made by the experienced developers, which would have a strong impact on validations and their experience helps them to perform an efficient analysis of the research questions.

As the basis was a set of documents from a Literature Review and few interviews it may have a possible problem in selection of factors. As the problem statement is similar, we may find better factors which had a better analysis, if a more significant LR which includes some bordering areas of study was made. To ensure more Basis
for decision making, more developer knowledge with additional interviews or possibly even workshops could have benefited RQ1. Even if the selection of factor was limited, the experienced developers and the information from the literature review gathered much valuable knowledge.
Chapter 10

Conclusions and Future Work

10.1 Conclusions

In this thesis, An improvement case study was conducted to develop a code review prioritization tool. Issues that were aimed to be addressed were:

- Prioritisation of open code review requests of a given user from multiple repositories.
- Help reduce the workload for a developer while code reviewing.
- Decrease the overall lead time of the code review process.
- Validate and Evaluate the developed tool in real time environments.
- Count for uncertainty while prioritising code review requests.

Initially to answer RQ1, Literature review and semi-structured interviews were conducted to identify the factors that impact the code review prioritisation process. A list of factors were finalised from RQ1 that provided a baseline to find and implement an approach to develop the tool. Statistical data analysis was performed on the data extracted from the repositories of Gerrit to select and finalise the factors from the list of factors identified.

The prioritization tool Pineapple depends upon these finalized factors. The factors used in the development of Pineapple are explained in Section 7.1 along with the motivation for selecting and ignoring each factor.

To answer RQ2 a Bayesian network was developed using the factors and relationships finalised in RQ1. Later performance evaluation of the tool was conducting using both RMSE and MAE metrics. The metrics in comparison with other existing approaches showed that Bayesian Network is a better implementation for the prioritization problem. The performance evaluation was also done by considering the response time of the prioritization requests. The average response time for a developer with around 50 open review requests showed to be about 2 seconds, which can be considered normal in real-time use.

To evaluate the feasibility, usefulness of Pineapple (RQ3), a static validation of the tool was conducted by using a feedback questionnaire provided in Pineapple. The
developers who use the tool answer the questions and their responses are visualized to analyse the feasibility, usability and usefulness of Pineapple. The visualisations of the responses are shown in Section 7.3. It was observed that around 83% of the developers consider Pineapple to be useful for prioritising code review requests; 95.7% of the developers believe Pineapple to be easy to use and understand; 56.5% of the developers consider that Pineapple helps in decreasing the lead time of a review process while 26% of the developers being neutral, which is one of the main objectives of the research; 60.8% of the developers believe that the prioritization done by Pineapple are reliable. All these responses help us to sum up the static validation of Pineapple and shows that the developers are satisfied with the outcome of the tool.

10.2 Future Work

Even though the performance evaluation and user evaluation of Pineapple were on the positive side, there are some areas that can be improved and have the whole tool much better and easy to use. Due to time constraint of only four months available for development, all the features were not implemented in the tool.

As future work and improvements, we can add many configuration settings to the tool having an administration GUI instead of manually configuring the data. Currently the repository names, blacklist etc. are manually inserted by the administrator while deploying the tool. This does not impact the quality of the prioritisations but helps in the maintenance and evolution of the tool. Currently, there is no option to extract data from a new single repo, when a new repository is added, we need to extract all the repositories in order to get the new data. So implementing a thread solution can be useful to decrease computational consumption while performing ETL.

One of the future tasks can be the implementation of Pineapple functionality in the form of Gerrit plugin or any automated pipeline task. This helps the user to use less number of tools and can complete the work at a single place.

Some of the other tasks that can be worth improving are to add additional information to the tool and try creating new Bayesian Network structures with new dependencies because as the data changes and the structure of the data used to train the network changes, the Bayesian Network may not perform as expected. The Bayesian Network has to be updated from time to time as there may be some degradation in the performance of Pineapple as the ways of working and data may change.

As a part of improvement a long term evaluation of feasibility, usability and usefulness of the tool can be conducted which helps in betterment of Pineapple.
References

[1] Shade Ruangwan, Patanamon Thongtanunam, Akinori Ihara, and Kenichi Matsumoto. The impact of human factors on the participation decision of reviewers in modern code review. *Empirical Software Engineering*, 24(2):973–1016, Apr 2019.

[2] Introduction to bayesian networks - towards data science. https://towardsdatascience.com/introduction-to-bayesian-networks.

[3] Junji Shimagaki, Yasutaka Kamei, Shane McIntosh, Ahmed E Hassan, and Naoyasu UBayashi. A study of the quality-impacting practices of modern code review at sony mobile. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 212–221, 2016.

[4] A Frank Ackerman, Priscilla J Fowler, and Robert G Ebenau. Software inspections and the industrial production of software. In *Proc. of a symposium on Software validation: inspection-testing-verification-alternatives*, pages 13–40, 1984.

[5] Aybuke Aurum, Håkan Petersson, and Claes Wohlin. State-of-the-art: software inspections after 25 years. *Software Testing, Verification and Reliability*, 12(3):133–154, 2002.

[6] Michael Fagan. Design and code inspections to reduce errors in program development. In *Software pioneers*, pages 575–607. Springer, 2002.

[7] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201, 2014.

[8] Ali Ouni, Raula Gaikovina Kula, and Katsuro Inoue. Search-based peer reviewers recommendation in modern code review. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 367–377. IEEE, 2016.

[9] Forrest Shull and Carolyn Seaman. Inspecting the history of inspections: An example of evidence-based technology diffusion. *IEEE software*, 25(1):88–90, 2008.

[10] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 712–721. IEEE, 2013.
[11] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. Modern code reviews in open-source projects: Which problems do they fix? In Proceedings of the 11th working conference on mining software repositories, pages 202–211, 2014.

[12] Achyudh Ram, Anand Ashok Sawant, Marco Castelluccio, and Alberto Bacchelli. What makes a code change easier to review: an empirical investigation on code change reviewability. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 201–212, 2018.

[13] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development: the integrator’s perspective. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, volume 1, pages 358–368. IEEE, 2015.

[14] Tarik Ozkul. Assessment of intelligence of user-computer interface. In 2016 7th International Conference on Information and Communication Systems (ICICS), pages 1–6. IEEE, 2016.

[15] Peter C Rigby, Daniel M German, Laura Cowen, and Margaret-Anne Storey. Peer review on open-source software projects: Parameters, statistical models, and theory. ACM Transactions on Software Engineering and Methodology (TOSEM), 23(4):1–33, 2014.

[16] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. Modern code review: a case study at google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, pages 181–190, 2018.

[17] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. Review participation in modern code review. Empirical Software Engineering, 22(2):768–817, 2017.

[18] Murtuza Mukadam, Christian Bird, and Peter C Rigby. Gerrit software code review data from android. In 2013 10th Working Conference on Mining Software Repositories (MSR), pages 45–48. IEEE, 2013.

[19] Jesus M Gonzalez-Barahona, Daniel Izquierdo-Cortazar, Gregorio Robles, and Alvaro del Castillo. Analyzing gerrit code review parameters with bicho. Electronic Communications of the EASST, 2014.

[20] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. The influence of non-technical factors on code review. In 2013 20th Working Conference on Reverse Engineering (WCRE), pages 122–131. IEEE, 2013.

[21] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA, USA, pages 15–17, 1985.
[22] Richard E Neapolitan. *Probabilistic reasoning in expert systems: theory and algorithms*. CreateSpace Independent Publishing Platform, 2012.

[23] Patanamon Thongtanunam, Raula Gaikovina Kula, Ana Erika Camargo Cruz, Norihiro Yoshida, and Hajimu Iida. Improving code review effectiveness through reviewer recommendations. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 119–122, 2014.

[24] Gaeul Jeong, Sung Hun Kim, Thomas Zimmermann, and Kwangkeun Yi. Improving code review by predicting reviewers and acceptance of patches. *Research on software analysis for error-free computing center Tech-Memo (ROSAEC MEMO 2009-006)*, pages 1–18, 2009.

[25] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355, 2014.

[26] Yuanrui Fan, Xin Xia, David Lo, and Shanping Li. Early prediction of merged code changes to prioritize reviewing tasks. *Empirical Software Engineering, 23*(6):3346–3393, 2018.

[27] Guoliang Zhao, Daniel Alencar da Costa, and Ying Zou. Improving the pull requests review process using learning-to-rank algorithms. *Empirical Software Engineering, 24*(4):2140–2170, 2019.

[28] ES Van der Veen. Prioritizing pull requests. 2015.

[29] Shin Yoo, Mark Harman, Paolo Tonella, and Angelo Susi. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 201–212, 2009.

[30] Paolo Bussetta, Fitsum Mesheha Kifetew, Denisse Munante, Anna Perini, Alberto Siena, and Angelo Susi. Tool-supported collaborative requirements prioritisation. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 180–189. IEEE, 2017.

[31] Jason But, Thuy Nguyen, Lawrence Stewart, Nigel Williams, and Grenville Armitage. Performance analysis of the angel system for automated control of game traffic prioritisation. In *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 123–128, 2007.

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[33] Teade Punter, Marcus Ciolkowski, Bernd Freimut, and Isabel John. Conducting on-line surveys in software engineering. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, pages 80–88. IEEE, 2003.
[34] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131, 2009.

[35] Martin Ivarsson and Tony Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3):365–395, 2011.

[36] Jennifer Rowley and Frances Slack. Conducting a literature review. *Management research news*, 2004.

[37] Chaim Noy. Sampling knowledge: The hermeneutics of snowball sampling in qualitative research. *International Journal of social research methodology*, 11(4):327–344, 2008.

[38] Deepika Badampudi, Claes Wohlin, and Kai Petersen. Experiences from using snowballing and database searches in systematic literature studies. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, pages 1–10, 2015.
Appendix A

Supplemental Information

Figure A.1: Pineapple web interface with feedback questionnaire
Figure A.2: Pineapple web interface with prioritised review requests
Appendix B

Interview Guide

| Nr | Question |
|----|----------|
| 1  | Please provide your background - your name, role and work experience |
| 2  | The need for having/implementing the tool? |
| 3  | Which factors do you consider for prioritisation of the review requests?  
    Present the factors identified from literature review |
| 4  | Given 100 points distribute them among the selected factors accordingly given its importance (both literature and new factors) |
| 5  | What are the possible relations between the selected factors? |
| 6  | Any other additional feedback or suggestions? |

Table B.1: Interview structure for the factor identification process.
## Appendix C

### Feedback Questionnaire

| Nr | Type | Question |
|----|------|----------|
| 1  | SAD  | Pineapple is useful to prioritise review requests. |
| 2  | SAD  | Pineapple is easy to use and understand. |
| 3  | SAD  | Pineapple helps in decreasing the lead time of review process. |
| 4  | SAD  | Pineapple helps inexperienced developers in performing code reviews. |
| 5  | SAD  | Prioritisations done by Pineapple are reliable. |

Table C.1: Questionnaire used to do the static validation of Pineapple
# Appendix D

## Statistical Analysis Results

| Interval                  | Mean | 25th% | 75th% |
|---------------------------|------|-------|-------|
| (2017-01-01) to (2017-06-30) | 12,973 | 38    | 6,753 |
| (2017-07-01) to (2017-12-31) | 13,689 | 29    | 6,254 |
| (2018-01-01) to (2018-06-30) | 13,049 | 51    | 6990  |
| (2018-07-01) to (2018-12-31) | 12,637 | 54    | 6024  |
| (2019-01-01) to (2019-06-30) | 10,435 | 51    | 5942  |
| (2019-07-01) to (2019-12-31) | 7,562  | 42    | 4404  |
| (2020-01-01) to (2020-06-30) | 4,710  | 43    | 3998  |

Table D.1: Statistical Analysis for Age of the changes

| Interval                  | Mean | 25th% | 75th% |
|---------------------------|------|-------|-------|
| (2017-01-01) to (2017-06-30) | 476   | 6     | 122   |
| (2017-07-01) to (2017-12-31) | 520   | 5     | 155   |
| (2018-01-01) to (2018-06-30) | 523   | 6     | 166   |
| (2018-07-01) to (2018-12-31) | 668   | 4     | 161   |
| (2019-01-01) to (2019-06-30) | 972   | 4     | 151   |
| (2019-07-01) to (2019-12-31) | 808   | 4     | 122   |
| (2020-01-01) to (2020-06-30) | 789   | 4     | 133   |

Table D.2: Statistical Analysis for Size of the changes

| Interval                  | Mean | 25th% | 75th% |
|---------------------------|------|-------|-------|
| (2017-01-01) to (2017-06-30) | 3.77 | 1     | 4     |
| (2017-07-01) to (2017-12-31) | 3.99 | 1     | 4     |
| (2018-01-01) to (2018-06-30) | 4.72 | 1     | 5     |
| (2018-07-01) to (2018-12-31) | 4.60 | 1     | 5     |
| (2019-01-01) to (2019-06-30) | 4.68 | 1     | 5     |
| (2019-07-01) to (2019-12-31) | 4.12 | 1     | 5     |
| (2020-01-01) to (2020-06-30) | 3.92 | 1     | 5     |

Table D.3: Statistical Analysis for Number of Patches of the changes
Appendix E

Bayesian Network Data Structure

| Variable         | Type                   |
|------------------|------------------------|
| Age              | New, Medium, Old       |
| Size             | Low, Medium, High      |
| No. of Patches   | Low, Medium, High      |
| Change Status    | Merged, Abandoned      |
| Peer Review      | -2, -1, 0, 1, 2        |
| Automated Tests  | -1, 0, 1               |
| Merge Conflict   | Yes, No                |
| Change Type      | Label Names            |

Table E.1: Data Structure used in Bayesian Network
