Developers’ performance analysis based on code review data:
How to perform comparisons of different groups of developers

Mikalai Barysau

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author:
Mikalai Barysau
E-mail: mibd15@student.bth.se
mikalai.barysau@gmail.com

University advisor:
Dr. Darja Šmite, Professor
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
Abstract

Context. Nowadays more and more IT companies switch to the distributed development model. This trend has a number of advantages and disadvantages, which are studied by researchers through different aspects of the modern code development. One of such aspects is code review, which is used by many companies and produces a big amount of data. A number of studies describe different data mining and data analysis approaches, which are based on a link between code review data and performance. According to these studies analysis of the code review data can give a good insight to the development performance and help software companies to detect a number of performance issues and improve the quality of their code.

Objectives. The main goal of this Thesis was to collect reported knowledge about the code review data analysis and implement a solution, which will help to perform such analysis in a real industrial setting.

Methods. During the performance of the research the author used multiple research techniques, such as Snowballing literature review, Case study and Semi-structured interviews.

Results. The results of the research contain a list of code review data metrics, extracted from the literature and a software tool for collecting and visualizing data.

Conclusions. The performed literature review showed that among the literature sources, related to the code review, relatively small amount of sources are related to the topic of the Thesis, which exposes a field for a future research. Application of the found metrics showed that most of the found metrics are possible to use in the context of the studied environment. Presentation of the results and interviews with company's representatives showed that the graphic plots are useful for observing trends and correlations in development of company's development sites and help the company to improve its performance and decision making process.

Keywords: code review, data metrics, performance measurement
## Contents

### Abstract

1 Introduction

1.1 Introduction to the topic .......................... 1

1.2 Background ........................................ 3

1.2.1 Empirical background and motivation ......... 3

1.2.2 Previous attempts to address the problem .... 4

1.2.3 Related research ............................... 5

1.2.4 Related personal experience .................... 6

1.3 Aim, objectives and research questions .......... 7

1.3.1 Aim ............................................. 7

1.3.2 Objectives ..................................... 7

1.3.3 Research questions ............................ 7

1.4 Contribution ..................................... 8

2 Research methodology ............................... 9

2.1 Method selection .................................. 9

2.2 Case study design ................................ 11

2.3 Literature review ................................ 13

2.3.1 Literature review approach selection and its brief overview 14

2.3.2 Exclusion criteria ............................ 15

2.3.3 Analysis of the sources, included to the study 15

2.4 Semi-structured interview ....................... 16

2.5 Research overview and traceability .............. 17

3 Results ............................................. 18

3.1 Literature review results ......................... 18

3.1.1 The list of performance-related issues, visible through the selected set of metrics 24

3.2 Case description ................................ 26

3.2.1 Information about the studied project .... 26

3.2.2 Life cycle of a Gerrit change ................. 28

3.2.3 Description of the code review process and structure of code review data 30
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Research overview and traceability</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>Code review data metrics, discovered during systematic literature study</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>List of keywords, associated with vulnerabilities from [P2], [P3], and [P4]</td>
<td>24</td>
</tr>
<tr>
<td>3.3</td>
<td>Detailed information about studied SPAT database</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>The revised list of data metrics</td>
<td>87</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Introduction to the topic

Nowadays more and more IT companies switch to distributed development model and share their development function with offshore software development companies located in different parts of the world. This policy has a number of attractive features, such as [1]:

1. lower price of working hour in receiving companies;

2. opportunity to develop software 24 hour a day by using difference in time zones;

3. localization of the company’s sites in the same area as company’s customers gives an opportunity to understand the needs and expectations of the customers better and perform a more effective customization of the developed product;

However, practice shows that besides the above advantages of distributed software development, it has a number of issues, which can badly influence the development performance, or even cause the project to fail [2]. The literature, dedicated to the distributed development, sheds a light on the problems of this approach, which were noticed by software companies’ in their practice [1]:

1. physical separation of company’s employees;

2. difference in cultural background;

3. difference in experience background;

4. network and hardware issues.

Some sites of the same company might be located in India, Bangladesh or China, others – in Canada, Australia, Russia or Turkey. Quite often difference in culture, knowledge and experience level between sites representatives leads to a big diversity in the quality of the code produced in different locations [2]. One
Chapter 1. Introduction

of the methods used by software companies to eliminate this drawback and unify code quality is code review [3].

This method was proposed by Michael Fagan [4] and is also known as Fagan’s inspection. Originally code review was described by Fagan as a part of development process, executed in a strict formal way. During the formal code review process participants perform line-by-line analysis of the code, using defined set of rules. The results of inspection are presented in a form of a report, which contains information about what was examined and how many issues were exposed [4, 5]. Scientific studies showed that comparing to a testing process, code review effectively improves quality of the code on the one hand, and quality of the development process, on the other. However, the main problem of the formal code review, which was faced by practitioners, is quite big amount of time, which is required to perform the complete formal procedure [6].

Mainly because of this reason many companies switched to a more lightweight process, which is also known as Modern Code Review [6, 7]. In common practice it is an informal process, which is performed with the usage of special purpose code review tools, such as Gerrit [1], Review Board [2], GitLab [3] and many others.

As it was mentioned above, switching to distributed development causes numerous development issues, which can affect the quality of the software product [8–10]. A well known study, performed in Microsoft [11], was targeted to compare the quality of the software products, produced by distributed and non-distributed development sites. The results of the comparison showed, that the number of the post-release defects, as a main measure of the code quality, in both cases is almost the same. In other words, the authors proved, that software product, produced by a single site, has the same quality as similar software product, produced by multiple sites within distributed development model. On the other hand, before the code is being released, it goes through the various quality assurance steps [12]. These steps are mainly targeted to ensure the functional and non-functional requirements. However, none of the used techniques consider the effort, payed by the company’s employees to keep the quality of the code on same level in the case, when the performance of the development sites is noticeably different. This research was performed in the software company, which faced the problem, when after the switching to the distributed development model the quality of the code remained the same, but the software architects faced a strong work overload, caused by a huge difference between the code, that was produced in a single Site And the code, that was produced within the distributed development model. At the same time the existing quality assurance mechanism did not allow to identify any specific issues and improve the quality and the results of the development process.

1 https://www.gerritcodereview.com/
2 https://www.reviewboard.org/
3 https://about.gitlab.com/
In the context of this research, developers’ performance is understood as a structural term, which describes particular qualities of code, produced by developers, and development process in general. These qualities can be evaluated based on artifacts from development process flow (version control system logs, build environment logs), from automated code quality checking systems (test environment logs, static analysis of the source files, dynamic analysis of binary files), from human discussions during code inspection process (code review comments) and bug fixing process (bug trackers comments), etc. The number of indicators of performance quality is variable and strongly depends on specifics of the studied environment and purpose of the performance analysis. In this project developers’ performance will be evaluated with usage of a set of metrics, which are described with more details in the Chapter 1 subsection 1.2.3.

1.2 Background

1.2.1 Empirical background and motivation

Distributed development model is used by many well known companies across the world. One of these companies is Swedish multi-national corporation Ericsson. As a Master’s degree studies student, looking for a Master’s Thesis topic, in the end of year 2015 the author of this Thesis had several discussions with Ericsson’s managers and software architects about the company’s distributed development experience. According to the Ericsson’s representatives, the company’s experts faced a strong code review work overload, appeared after the switching to the multi-site development approach. In particular, as code reviewers, they noticed quite significant difference in the quality of the code, produced in freshly joined sites. Technically, the gap in the sites’ developers competence and experience is being filled with the additional effort, paid by experienced and valuable employees. On the one hand code review indeed helps to unify code quality and share the experience and knowledge across the company’s sites. On the other hand, migration to the distributed software development model strongly influenced performance of the company’s experts. Ericsson’s representatives underlined that to avoid deterioration of the quality of the produced code, caused by lower competence of the development sites, they are forced to spend much more time on the code review process, than previously. This problem caused a list of company’s needs: 1) a need of a method of identification the root cause of the development performance issues; 2) a need of an examination of the site’s learning progress; 3) a need of finding potential directions for developers’ performance improvements. Company’s practice showed, that the existing code quality evaluation mechanisms, such as testing environment and static code analysis tools, were not able to identify the root cause of the above problems. At the same time the code review process itself produces a big amount of data, which contains information
about the quality of the written code, such as code review comments, \textit{diff} files\footnote{\textit{diff} files show difference in source code between revisions} information about the amount of commented code, time, needed to perform a review, and other development process related data. Therefore, based on the above needs, the company’s experts came up with an idea of performing a research in this field to find out, what is already known about this problem in the industry and scientific community, how the code review data can be processed and what kind of information can be extracted from it. The subsequent discussions with the Ericsson representatives helped to identify a list of objectives of the research, which formed a basis for this Thesis:

- extract data, which will allow to evaluate the effort, paid by company’s experts on code review for sites’ developers;
- create an opportunity to examine trends in developers’ performance evolution during the code review process;
- create an opportunity to measure developers’ performance and examine their learning progress;

\subsection*{1.2.2 Previous attempts to address the problem}

Modern Code Review undoubtedly is much less time-consuming process and still quite effective from the perspective of improving the developers’ performance \cite{6}. At the same time migration to the tool-based code review also resulted in huge datasets, which contain crucial information about development process and code quality. Application of the data-mining approach allows to extract this information and perform it’s further analysis \cite{13, 14}. During the discussion about possible starting points for the project, the author of this Thesis received an information about previous attempts of the company to solve their needs for data analysis, made by two students from BTH \cite{15}. During the study the students manually analyzed a set of code review comments and collected statistical information about a number of functional and non-functional defects, type of comments and role of code reviewers in projects. For this purpose they created a data mining tool, named \textit{SPAT}. This tool allows to download code review data from the Gerrit server – code review platform, used in the company. The results of the manual classification and analysis of the code review data contained examples of such metrics, as a number of exposed defects, defects classification, team members and architects participation in the code review process. In addition, the students performed experiments with automated data extraction and collected statistical information, such as number of revisions, connected to specific feature and the number of commits reviewed by specific persons. After the evaluation of the results of the research, performed by Grisšins and Sekáč, the company was
interested in a deeper research about possible ways of analysis of code review data and types of information, which can be extracted from it in an automated way. The results of this research were not published, but the report was created and the content of the report was verified and confirmed by the company’s employees as trustworthy.

This research was performed in cooperation with Andrej Sekáč, but his part was mainly focused on the computer science aspect of the data analysis. The author of this Thesis was more concentrated on different kinds of metrics and examination of specifics of applying these metrics on the code review data, produced by the different sites of the company. Thus the contributions in both cases are unique.

1.2.3 Related research

To formulate the aim and objectives of the research and define research questions it was needed to know, if any systematic literature study, related to the code review data analysis, was performed previously and what is already known about this topic. Grišins and Sekáč performed a literature overview for their research, but it was not systematic and the results mainly contained information about the code review process itself and its influence on the software development process. Because of this reason the author performed a string search for the reported systematic literature studies about code review data. Description of the performed procedure can be found in the Appendix C. During the search a corresponding literature review was not found. Thus, it was decided to perform a snowballing literature study within this Thesis. Process of selection of the literature review method and process description can be found in the section 2.3. The following parts of this subsection presents results of the literature study.

Results of the found sources analysis

Based on the extensive literature search 6 sources were found to be relevant out of 266.

Quite a big part of the reviewed sources was related to the code review process organization, its role in the code quality assurance, different approaches to the code review process, existing code review tools, e.t.c. Part of the excluded sources contained information, which was useful for general overview of the code review process and was used during the work on this research, but was not included to the further literature study due to being not directly related to the topic of the research.

Significantly small part of the processed sources was directly related to the topic of this research. This fact proves relative innovation of the topic and research contribution of this Thesis. On the other hand, the found sources give not so much precise information about the analysis of the code review data.
Chapter 1. Introduction

Summary of the findings, discovered during the analysis of the found sources:

1. Several sources contained a number of metrics for code review data, which can be used for the extraction of the code quality-related data \[P6\], \[P5\].

2. The results given by the proposed metrics are highly dependent on the specific team, product, code fragment, type of change, e.t.c. In other words, mentioned code review data metrics should be used for observing trends and results should be considered in the context of specific product, team, e.t.c. \[P6\].

3. Code review data can be analyzed using linguistic analysis tools for identification of type of issues, exposed during code review and pointed out by code review comments \[P2\], \[P3\], \[P4\].

4. One paper contained an example of a visual representation of metrics, extracted from code review data \[P1\].

The metrics, which were discovered in the literature sources \[P6\] and \[P5\], are presented in the Table 3.1. Comments of the author regarding specific metrics and results of application can be found in the subsection 3.4, Chapter 3.

Three sources \[P2\], \[P3\], \[P4\] which discuss security vulnerabilities, present most frequently exposed vulnerabilities and suggest a list of keywords, which are characteristic for the code review comments, related to the corresponding vulnerability. The list of these vulnerabilities and keywords is presented in the table 3.2. The metric, based on usage of these keywords for counting code review comments, is described in the subsection 3.4.

The performed literature review is described in details in the appendix D.

Section 3.2 from the Chapter 3 contains results of applying these metrics on company’s code review data. Section 3.6 presents feedback from company’s experts about extracted results and indicated problems. The ways of presentation of extracted results, described in \[P1\], were used during analysis of the data and discussions with company’s experts.

1.2.4 Related personal experience

At the time of writing this Thesis the author has around 3 years of experience in code reviewing (as a reviewer and as a developer). For one year the author worked in a multinational software company. Therefore the author’s opinion, which comes from his practice and experience, is also expressed in this Thesis.
1.3 Aim, objectives and research questions

1.3.1 Aim

The aim of this research is to help the case company to measure and improve developers’ performance. In particular, to find data metrics, associated with code reviews, understand, if they are applicable in the studied environment and find a way how to compare developers’ performance in different sites over time using found metrics.

1.3.2 Objectives

OBJ 1: Collect and analyze reported knowledge about code review data analysis and its usage in developers’ performance evaluation;

OBJ 2: Define a set of code review data metrics for developers’ performance analysis in the studied environment;

OBJ 3: Analyze code review process in the studied environment from the developers’ performance evaluation perspective;

OBJ 4: Validate the suitability of the identified metrics in the studied environment through sample code review data analysis;

OBJ 5: Obtain company’s experts opinion about the results of the research;

1.3.3 Research questions

Based on the information found during related research and analysis of the problem performed in cooperation with company’s representatives, the following research questions were defined:

RQ 1: How the code review data can facilitate developers’ performance analysis?

RQ 1.1: What performance-related metrics can be computed based on the data available from code reviews?

RQ 1.2: How shall these metrics be analyzed in order to evaluate developers’ performance?

RQ 2: Which performance-related issues can be identified with the help of selected set of metrics, based on a sample code review data?

RQ 3: How the results of the research can help the company in solving developers’ performance-related issues?

Relation between research questions, objectives and research methods is presented in the table 2.1 in the Chapter 2 section 2.5.
1.4 Contribution

The main contribution of this work consists of improved practice at the case company regarding the identification of developers’ performance issues. This Master’s Thesis, as a result of the performed research, contains the following contributions:

- Results of a systematic literature study – a collection of literature sources, closely related to the code review data analysis from the developers performance analysis perspective (see subsection 1.2.3, Chapter 1).

- A list of metrics, based on the found literature and proposed during the code review data analysis and discussions with the company’s experts (see table 3.1, subsection 1.2.3, Chapter 1).

- Results of applying these metrics in the company’s context and feedback, gained from the company’s experts (see section 3.6, Chapter 3).
Chapter 2

Research methodology

2.1 Method selection

To answer the research questions and complete the objectives listed in the section 1.3, the author of this Thesis conducted case study, which consist of a systematic literature study to prepare a set of metrics for code review data, application of the performed metrics on the archival data and a set of semi-structured interviews to validate the results of the research. Results of the literature study contain a set of code review data metrics, which were applied in the studied environment within a case study. Finally, the results of the case study were discussed with the company’s experts in form of semi-structured interview.

There are several reasons behind choosing case study as main research methodology for this research. First of all, it is caused by the aim of the research. Guidelines for conducting and reporting case study research in software engineering presents 4 types of research purposes [16]:

- Exploratory – finding out what is happening, seeking new insights and generating ideas and hypotheses for new research.
- Descriptive – portraying a situation or phenomenon.
- Explanatory – seeking an explanation of a situation or a problem, mostly but not necessary in the form of a causal relationship.
- Improving – trying to improve a certain aspect of the studied phenomenon.

The main purpose of this research is to propose a set of metrics that would help to evaluate performance on a developers or development unit level by exploring the ways to link code review data with performance indicators. According to the below source, case study in software engineering is mainly used for exploratory purpose [16].

The same source emphasizes that "case studies are by definition conducted in real world settings, and thus have a high degree of realism, mostly at the expense of the level of control". The same idea is mentioned in [17]: a case study "investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and context are not clearly evident".
Chapter 2. Research methodology

This research was conducted to analyze specific case — code development in a specific project, performed by several development sites. The analysis of the case is based on analysis of data, produced by different company’s sites while developing the same project.

Literature review was performed to gather existing knowledge about the topic of the research and systematize it for further applying on the code review data. More details about chosen literature review approach can be found in the section 2.3 of this chapter.

Semi-structured interview The main purpose of performing interviews in terms of this research was to validate the results of the literature review and case study. It was decided to use this method instead of Survey to perform feedback collection in a more flexible way. In more details this method is presented in the section 2.4.

Section 2.2 presents the design of case study, performed in this research.

Other research methods, such as controlled experiment, ethnography or action research were found inappropriate in this case for the reasons described below.

According to the experiment methodology description [18], experiment is conducted to investigate "a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables". In other words, an experiment is conducted in cases, when the same data with is used as input several times with different parameters and the purpose of experiment is analyze the output, received using different combinations of experiment parameters. In case of this research, the main purpose was not to analyze influence of different parameters on same code review data, but to apply different metrics to this data and try to find trends in performance of company’s sites.

Ethnography was also found inappropriate, since it focuses on studying a specific social group [19].

Action research was firstly considered as a candidate for the research method for this research because of it’s iterative nature. However, after studying the guidelines for this method [20], it was refused, since its main purpose is to solve problem and also study the experience of solving the problem, which is also different to the purpose of this research.

Figure 2.1 overall scenario of this research and depicts connections between research questions, objectives, used research methods and achieved results of the research.
2.2 Case study design

According to the case study guidelines [21], case study design consists of the following elements:

1. Rationale – Why is the study being done?
2. Purpose – What is expected to be achieved with the study?
3. The case – Overall, what is being studied?
4. Units of analysis – In more detail, what is being studied?
5. Theory – What is the theoretical frame of reference?
Chapter 2. Research methodology

6. Research questions – What knowledge will be sought or expected to be discovered?

7. Propositions – What particular (causal) relationships are to be investigated?

8. Concepts and measures – How are entities and attribute being defined and measured?

9. Methods of data collection – How will data be collected?

10. Methods of data analysis – How will data be analyzed?

11. Case selection strategy – How will cases (and units of analyses) be identified and selected?

12. Data selection strategy – How will data be identified and selected? Who will be interviewed?

13. Replication strategy – Is the study intended to literally replicate a previous study, or theoretically replicate a previous study; or is there no intention to replicate?

14. Quality assurance, validity and reliability – How will the data collected be checked for quality? How will the analysis be checked for quality?

Elements 1, 2 and 6 are covered in the introduction to this Thesis (Chapter 1). The rest of the above elements and case study protocol are described below.

The case

The case in this research is software studied project, considered within a context of distributed development. According to the guidelines [21], this case can be classified as Embedded single-case study.

Units of analysis

Units of analysis in terms of this case study are the performance of developers in different sites, presented in form of particular indicators.

Theory and Measures

Theoretical part of the case study is covered by Systematic Literature study, which resulted in a set of code review data metrics, which will be applied during the case study.
Propositions and Hypotheses

According to the [22], "propositions are predictions about the world that may be deduced logically from theory". The proposition, driving this research is that code review data can be linked with developers’ performance using metrics.

Methods of Data Collection

This case study is based on archival data, collected from company systems. During the previous research in the company [15], researchers created a database and data mining tool called SPAT, which fills the database with requested data. During this case study the author is using SPAT tool and the database, which were extended and modified to enable calculation and storage of defined set of metrics. For the validation purposes company’s experts opinions are collected through semi-structure interviews.

Methods of Data Analysis

The code review data, extracted from the system, was analyzed according to the metrics description from the literature sources. Details of the performed semi-structured interviews can be found in the section 2.4 and results can be found in the Chapter 3 section 3.6.

Case Selection and Data Selection Strategy

Before selecting the data the author contacted the architects from the studied project, which were leading the development process of specific features. For the study purpose, the architects were asked to provide a list of changes and features, which were developed in different sites during different periods of time.

Quality assurance

To ensure that archival and metrics data are accurate, a set of automated test cases was added to the SPAT tool to check the results of each metric, using prepared data and expected result values. Besides that the results of the research were validated with the company’s experts during set of semi-structured interviews.

2.3 Literature review

Before starting the research it was needed to understand, what is already known about the topic of the Thesis and examine existing work, related to code review data analysis and it’s potential usage in code quality analysis. This activity covers [OBJ 1].
Collect and analyze reported knowledge about code review data analysis and its usage in developers’ performance evaluation.

The following 2 subsections describe the process of literature approach selection (subsection 2.3.1) and used exclusion criteria. The results of literature review execution can be found in the Chapter 1, subsection 1.2.3.

2.3.1 Literature review approach selection and its brief overview

In order to complete OBJ 1 in a rigorous and replicable way, the author has decided to conduct a systematic literature study. During the review of the actual approaches of systematic literature studies in software engineering ([23], [24], [25] and [26]) two different approaches were discovered:

1. **Systematic Literature Review (SLR)** – an approach, based on searching literature sources in literature database, using search strings [23];

2. **Snowballing** – an approach, based on iterative examination of references and citations [24];

To decide, which approach shall be used to complete OBJ 1 the author has examined a comparison, made by Jalali and Wohlin [27]. According to the results of this paper, both methods have their pros and cons. On the one hand, SLR can lead to a higher level of noise (irrelevant sources) in the results. On the other hand, Snowballing brings a risk of incomplete results, since some relevant sources might be rarely cited or not cited at all. To mitigate the impact of disadvantages of both literature study approaches and achieve more rigorous results, it was decided to use Snowballing as a main search strategy and database search, performed with search strings, to create a start set of sources (initial step of Snowballing procedure).

The Snowballing literature study was performed according to the guidelines, prepared by Claes Wohlin [24]. According to the guidelines, Snowballing procedure starts with defining a start set of sources. Next step of Snowballing lies in performing a set of iterations. Each iteration is associated with the corresponding input set of sources. The first iteration is associated with the start set of sources. During each iteration each source from the input set is processed using backwards (examination of source’s references list) and forward (examination of source’s citations) snowballing. Found sources are analyzed and filtered using defined exclusion criteria (see subsection 2.3.2). The sources, which left after filtering, became an input set for the next iteration and procedure runs until no new sources will left after filtering. For more detailed description of the Snowballing procedure see [24].
2.3.2 Exclusion criteria

To filter found sources the following exclusion criteria were used during the literature review:

1. The context of the literature source citation is not relevant;
2. Title of the source did not contain relevant information;
3. Abstract or text of the literature source did not contain relevant information;
4. Literature source is written in language, other than English;

The following types of information, contained in the source, were identified as relevant:

- information related to performance evaluation based on code review data;
- information about relation between code review data and developers’ performance;
- information about code review data measuring, visualization or analysis, related to performance evaluation;

Since the data mining tool was already created during the previous research [15], information about data mining and data extraction from code review data database server was bypassed.

During the database search only criteria #3 and #4 were applied. Firstly, the author was trying to determine, if the candidate contains relevant information by reading the abstract. If it still was unclear, the full text of the source was analyzed.

During the Snowballing review the exclusion criteria were applied one after another.

If considered source met at least one of the used criteria, it was instantly removed from the study and not analyzed anymore, otherwise it was included to the study.

If considered source met criteria #2 or #3 it was classified as out of scope.

2.3.3 Analysis of the sources, included to the study

After the snowballing procedure was finished, the found sources were analyzed and the information, relevant to the topic of this research, was systematized for further application on code review data in studies environment. The author read the full text of the sources and extracted the information about data metrics, described in the found literature. The table 3.1 contains all the metrics from
the found sources with description of each metric. It was decided first to collect all found metrics and then verify, which of them can be applied in the studied environment.

2.4 Semi-structured interview

Semi-structured interviews were used during this research in order to obtain company’s experts opinion about the results of performed case study and literature review.

The main reason behind selecting this research method over structured interviews is that in structured interview the researcher follows defined interview questions, which are standardized for all respondents [28]. This kind of interviews is applicable, when all questions are known and the main purpose of the interview is to collect the answers from respondents. The set of possible in some cases also may be defined (yes, no, low, high, medium, e.t.c.) [29].

The opposite version of the structured interview in unstructured interview, where the interviewee is the source of both questions and answers [29]. However, analysis of data, received from unstructured interview may be quite complicated due to possibly big variety of different answers and questions. Thus, a purely unstructured are not recommended to be used extensively due to a high costs [29].

Semi-structured interview is a mixed version of structured and unstructured interview, where the list of questions is defined before the interview, but the interviewer also can ask additional questions and extend the interview. This type of interview is used in situations, when the some unexpected answers are possible [29]. To make discussions with company’s experts flexible and keep an opportunity to extend the interview in case, when additional questions might improve quality of the interview, the author decided to choose semi-structured interviews to complete objective [OBJ 5]

- **OBJ 5** Obtain company’s experts opinion about the results of the research;

After finishing of the literature review and case study, the author presented the results of the research to the company’s experts. After the presentation three interviews 30-60 minutes length were performed with 3 company’s employees. The interviews were performed with the key persons, who were engaged in production of the studied sample of code and code review process. These persons are line manager and release manager, who were managing distributed development process during the studied period, and a software architect, who was the main reviewer in the studied sample of Gerrit changes. During the interviews the author of this Thesis was asking questions from the below list:

1. Are the results of the research useful for the company in terms of developers’ performance analysis?
Chapter 2. Research methodology

- if yes, how can it help in developers’ performance analysis?
- if no, why?

2. Which of the results are less useful or completely irrelevant? Why?
3. Which results were unexpected before the research was started?
4. What was missing in the results of the research in your opinion?
5. Is there a need to continue or expand the current research?
  - if yes, what should be included in the next research?
  - if no, why?

Answers to the interview questions, given by company’s experts, are presented in the Chapter 3 section 3.6.

2.5 Research overview and traceability

Relation between objectives, research questions and research methods is presented in the Table 2.1

Table 2.1: Research overview and traceability

<table>
<thead>
<tr>
<th>Objective</th>
<th>Research question</th>
<th>Research method</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJ 1</td>
<td>RQ 1.1, RQ 1.2</td>
<td>Snowballing literature review</td>
</tr>
<tr>
<td>OBJ 2</td>
<td>RQ 1.1</td>
<td>–</td>
</tr>
<tr>
<td>OBJ 3</td>
<td>RQ 1.2</td>
<td>–</td>
</tr>
<tr>
<td>OBJ 4</td>
<td>RQ 2</td>
<td>Case study: data collection</td>
</tr>
<tr>
<td>OBJ 5</td>
<td>RQ 3</td>
<td>Semi-structured interview</td>
</tr>
</tbody>
</table>
3.1 Literature review results

During the literature review the author of this Thesis collected a set of metrics that were presented in the reviewed literature. The below table contains all the found metrics with a short description and reference to the original source.
Table 3.1: Code review data metrics, discovered during systematic literature study

<table>
<thead>
<tr>
<th>Metric</th>
<th>Measurement approach</th>
<th>Definition from the source</th>
<th>Values and interpretation from the source</th>
<th>Literature sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>Number of modified lines of code.</td>
<td>Size of the review.</td>
<td>-</td>
<td>[P6]</td>
</tr>
<tr>
<td>sLOC</td>
<td>Number of lines of source code without whitespace and comments.</td>
<td>Size of executable code.</td>
<td>-</td>
<td>[P6]</td>
</tr>
<tr>
<td>Inspection time</td>
<td>Amount of time, which is needed to perform the review.</td>
<td>Duration of the review.</td>
<td>Should not be long, otherwise it indicates that reviewer is &quot;forgetting or neglecting the review&quot;</td>
<td>[P5],[P6]</td>
</tr>
<tr>
<td>or Review time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[P5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Pre-review time

<table>
<thead>
<tr>
<th>Description</th>
<th>Explanation</th>
<th>Optimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time interval between the moment, when the author uploaded the change and the moment, when the test build was performed, initial tests were executed and the author marked the change as &quot;ready for the review&quot;.</td>
<td>This metric indicates how much rework is needed after the change was uploaded for the first time and build and tests were executed.</td>
<td>Optimal value should be less than 24 hours. Long pre-review time indicates that additional work was required after uploading change for the first time.</td>
</tr>
</tbody>
</table>

### Pre-integration time

<table>
<thead>
<tr>
<th>Description</th>
<th>Explanation</th>
<th>Optimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time interval between the moment, when reviewer finished the review and accepted the change, and the moment, when the integration of the change started.</td>
<td>This metric indicates how long it takes for the integrator to start the integration process.</td>
<td>Optimal value should be not more than 24 hours excluding weekends.</td>
</tr>
</tbody>
</table>

### Integration time

<table>
<thead>
<tr>
<th>Description</th>
<th>Explanation</th>
<th>Optimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time interval between the moment, when reviewer finished the review and accepted the change, and the moment, when the change was merged into the main branch.</td>
<td>This metric indicates, how long it takes for the integrator to perform an integration process.</td>
<td>Optimal value is less, than 24 hours, excluding weekends.</td>
</tr>
<tr>
<td><strong>Inspection rate</strong></td>
<td>Speed, at which certain amount of code is reviewed. Measured in ( LOC ) divided by inspection hours.</td>
<td>Quality of the review.</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><strong>Defect count</strong></td>
<td>Amount of defects, which were found during the review.</td>
<td>Effectiveness of the review. According to the source, defect here is something a reviewer wants changed in the code. It could be as serious as a bug in an algorithm or as trivial as some reformatting or typo in a comment.</td>
</tr>
<tr>
<td><strong>Defect rate</strong></td>
<td>Speed, at which defects are uncovered by reviewers. Measured as defect count divided by inspection hours.</td>
<td>Speed of the review performance.</td>
</tr>
<tr>
<td><strong>Defect density</strong></td>
<td>Number of defects found in a given amount of source code. Measured as defects count divided by LOC.</td>
<td>Effectiveness of the review.</td>
</tr>
</tbody>
</table>
### Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Quality of the change and &quot;smoothness&quot; of the development process. Low number of patch-sets indicates good quality of the code and development process.</th>
<th>Optimal value if 2-5 patch-sets per change.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of patch-sets(^2)</td>
<td>Number of patch-sets in a single change.</td>
<td></td>
<td>[P5]</td>
</tr>
<tr>
<td>Abandoned changes count</td>
<td>Amount of Gerrit changes that are abandoned and not merged.</td>
<td>Effectiveness of the development process.</td>
<td>- [P5]</td>
</tr>
<tr>
<td>Number of positive and negative review labels(^3)</td>
<td>-</td>
<td>This metrics indicates how many negative and positive feedbacks were given.</td>
<td>- [P5]</td>
</tr>
</tbody>
</table>

1. *Change*, according to the Gerrit terminology [30], is a single instance of code review.
2. *Patch-set*, according to the Gerrit terminology [30], is a single modification of a change.
3. *Review labels*, according to the Gerrit terminology [30], are shortcuts for reviewer’s feedback [30]. The range of values is \([-2, -1, 0, +1, +2]\).
Chapter 3. Results

According to [P6] analysis of results, extracted using given metrics, should be performed together with the analysis of specifics of the reviewed code. For example, reviewing a core component of some system most probably will take much time due to a high reliability and performance requirements to such component. Thus, in case, when the reviewed code has good quality (low defect count and defect density) and the amount of changed code is not big (low LOC), the code will be reviewed longer (high inspection time, low inspection rate and defect rate). On the other hand we can apply these metrics to the same amount of code, written for a component, which has non-critical functionality and currently is implemented as a proof-of-concept [31]. In other words, this code is not critical term of quality. In this case all the metrics might give completely different results just because of a different context of the reviewed component. For instance, reviewers might skip performance and code-style issues and concentrate directly on the functionality part of the code, which will also result in a low defect count and defect density. Inspection time may vary, since on the one hand reviewers are expected to spend less time on reviewing proof-of-concept code. On the other hand, low quality code might require more time to examine it.

In the same way other additional details, which may influence the code development and reviewing processes, should be also analyzed together with the code review data. More practical examples will be presented in the Chapter 3 sections 3.2 and 3.6.
Table 3.2: List of keywords, associated with vulnerabilities from [P2], [P3] and [P4]

<table>
<thead>
<tr>
<th>Vulnerability type</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Race Condition</td>
<td>race, racy</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>buffer, overflow, stack</td>
</tr>
<tr>
<td>Integer Overflow</td>
<td>integer, overflow, signedness, widthness, underflow</td>
</tr>
<tr>
<td>Improper Access</td>
<td>improper, unauthenticated, gain access, permission, access</td>
</tr>
<tr>
<td>Command Injection</td>
<td>command, injection</td>
</tr>
<tr>
<td>Cross Site Scripting</td>
<td>cross site, CSS, XSS, htmlspecialchars (PHP only)</td>
</tr>
<tr>
<td>Denial of Service (DoS)</td>
<td>denial service, DOS, crash</td>
</tr>
<tr>
<td>Deadlock</td>
<td>deadlock</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>SQL, SQLI, injection</td>
</tr>
<tr>
<td>Format String</td>
<td>format, string, printf, scanf</td>
</tr>
<tr>
<td>Cross Site Request</td>
<td>cross site, request forgery, CSRF, XSRF, forged</td>
</tr>
<tr>
<td>Forgery</td>
<td></td>
</tr>
<tr>
<td>Common keywords</td>
<td>security, vulnerability, vulnerable, hole, exploit, attack, bypass, backdoor, threat, expose, breach, violate, fatal, blacklist, overrun, insecure, crash</td>
</tr>
</tbody>
</table>

3.1.1 The list of performance-related issues, visible through the selected set of metrics

The following list presents list of metrics and performance-related issues, linked to it, that can be captured by applying the selected metrics on the code review data.

1. **LOC**: High value indicates long code reviews, which might indicate both bad task specification and planning or improper way of working and committing many modifications in one commit.

2. **sLOC**: Same as previous.

3. **Inspection time or Review time**: According to the sources [P5] and [P6] long inspections indicate, that reviewers forget or neglect the review [P5]. According to the author of this Thesis it also might mean, that the review required much effort to put in it due too poor quality of the reviewed code or unfamiliarity of the reviewer with the reviewed component or the technology, used in the reviewed code.
4. **Pre-review time:** High value indicates the high amount or rework, needed to perform before the review was started.

5. **Pre-integration time:** High value indicates that the integrator postpones the integration of the change to the the main code. This might mean high effort, work overload of the integrator or bad time management.

6. **Integration time:** High value indicates that the integrator needs more time to perform the integration of the change to the the main code. This also might mean high work overload of the integrator or high number of integration problems, such as merging conflict. The latter indicates, that the change was not well prepared by its author.

7. **Inspection rate:** High value indicates, that big amount of code was reviewed during very short time, which might mean low quality of the code review.

8. **Defect count:** According to the source [P6] it indicated effectiveness of the review. According to the author of this Thesis, it also indicates quality of the reviewed code from the perspective of the number of present defects.

9. **Vulnerability-related comments count:** Quality of the reviewed code from the perspective of the number of present security-related defects.

10. **Number of comment conversations:** According to the company’s experts high value usually indicates poor quality of the code and big amount of needed rework.

11. **Defect rate:** According to the source [P6] the metrics reflects the speed, at which the code is being reviewed. According to the author of this Thesis, it also indicates the quality of the reviewed code. For instance, high value means high number of defects, present in the code.

12. **Defect density:** According to the source [P6] the metrics reflects effectiveness of the review. According to the author of this Thesis, it also indicates the quality of the reviewed code. For instance, high value means high number of defects, present in the code.

13. **Number of patch-sets:** Quality of the development process and the code. For instance, low value of the metric indicates, that the code need small amount of rework to get approved for merging.

14. **Abandoned changes count:** Effectiveness of the development process. High number of the abandoned changes might mean, that a lot of effort was wasted due to abandoning or reimplementing particular features.
Chapter 3. Results

15. **Number of positive and negative review labels**: Number of positive and negative feedbacks, that were given to the reviewed code. High number of positive feedbacks means good quality of the code. High number of negative feedbacks means bad quality of the code correspondingly.

### 3.2 Case description

#### 3.2.1 Information about the studied project

**General information**

The studied project is developed for approximately 15 years and current number of software developers, participating in it is around 150 persons. During the previous research in the company the researchers identified 4 groups of employees:

- Developers – responsible for code development and reviewing code, produced by other developers;
- Design leads – experienced developers, whose responsibility is performance of code review and controlling quality of produced code;
- Architects – highly experienced developers, whose responsibility is building the architecture of the project and reviewing important parts of the code;
- Release managers – responsible for approving merge requests and managing git branches;

Development teams of studied project are located in different countries around the world [15]. During the discussions with company’s experts two main development site’s were identified, which, in term of this Thesis, will be marked as Site A and Site B. Comparison and analysis, made in this Thesis, will be done using code review data from these two sites.

The studied project was initially created and developed by Site A. Thus, developers from Site A have the most knowledge about the project, established code style rules and traditions and general view on how the project will be developed. Team of main architects is also residing in Site A.

Site B was introduced to the project in autumn 2014. At the same time a group of developers from Site B arrived to Site A for knowledge handovers and set of hands-on trainings, organized by local experts. At the beginning of 2015 They returned to Site B and started to work independently. Also, during several month at the beginning of 2015 a small group of experts from Site A were supporting developers from Site A during performance of their tasks. After this moment Site A handovered part of development responsibilities to Site B. As a results, experts from Site A concentrated on the requirements specification for Site’s B tasks, supporting Site B developers in complicated tasks and reviewing
Chapter 3. Results

the produced code. Internal organization of the work and tasks management were performed by Site B employees independently.

Features, implemented in the project, also can be divided into several groups:

- **FC** features. Before 2015 were developed by Site A. Since beginning of 2015 are mostly developed by Site B;

- **FR** features. Are mostly developed by Site A;

During the initial discussions with company’s experts, which preceded the research performance, it was discovered that development of Site B is noticeably lower and employees from Site A have to pay a lot of effort to support them. Comparing to Site A the main problems of Site B, which were mentioned by company’s experts, are following:

- lower quality of the code;

- longer reviews with bigger number of code to review and higher complexity of the review process;

- significantly slow learning progress of the Site B;

The studied sample of code reviews was produced between May 2014 and December 2015. During this period the development of the features FC and some FRs was transferred from Site A to Site B. Thanks to this, the selected sample contains code review data, that can be segregated into 2 groups: one group with the changes, produced by Site A and another with the changes, produced by Site B. According to the information from the software architect, who was involved in the development and reviews of these features, from the technical point of view both groups contain equal number of challenging tasks with high complexity and also simpler tasks, which required less effort to develop and review them.

Technical information

Development of the studied project is performed in Java language, but also partially is performed in C++, since previously C++ was the main development language. Git is used as a version control system and Gerrit is a review tool. Software builds and tests are automatically triggered by Jenkins continuous integration tool. Static analysis of the code is performed using SonarQube tool.

1 https://git-scm.com/
3.2.2 Life cycle of a Gerrit change

Whenever a developer wants to initiate a code review, he or she creates a unique Gerrit change and the review process starts. Gerrit changes consist of patch-sets. Each change consists of at least 1 patch-set. Patch-set is a modification of specific change.

When change or new patch-set of specific change are uploaded to Gerrit, Jenkins triggers a build process of the committed code and run tests, if build was successful. At the same time automated tools environment starts the code static analysis. Scheme of the life cycle of a Gerrit change in the studied environment is presented on picture \[\text{3.1}\]

General steps of the change’s life cycle are the following:

1. Developer creates a change;
2. Developer makes code modification;
3. Developer uploads change for review;
4. Next step consists of several substeps:
   - Change is being reviewed by other developers, architects and design leads;
   - Automated tools environment builds code and performs tests, if build was successful.
   - Automated tools environment triggers static code analysis;
5. If change receives +2 (change is approved), developer starts merging process, otherwise go to step 2.

Next subsection provides information about code review data, which is produced during code review process.
Figure 3.1: Life cycle of a change in the studied environment
3.2.3 Description of the code review process and structure of code review data

During the research the author was analyzing mainly data from Gerrit database, produced during studied project development process.

As it is mentioned in previous subsection, single instances of code review is called change. Each Gerrit change is associated with a specific commit message, which contains change's description. If change is a part of some developed feature, developers usually add feature's acronym to this message. Example of the commit message, associated with specific change:

[2015-08-04 12:33:21]: FEATURE_12412: Modular input dialog for client-side GUI.

When patch-sets are uploaded, a special Gerrit message, associated with current change, is created, which contains information about sequence number of uploaded change. Here is an example of such message:


During the review process reviewers can leave code review comments about specific blocks of code. In most comments reviewer points out to some fragment of code, which is not clear for him/her or some fix or modification of the code is required according to his or her opinion. Quite often the author of the code replies to the comment and thus a single comment transforms to a mini-discussion. Here is an example of a code review comment discussion, related to a specific fragment of code:

[2014-01-22 16:43:46] {reviewer}: This method duplicates functionality, which is already implemented in FEATURE_56417.

[2014-01-22 17:13:32] {developer}: FEATURE_56417 implementation does not work properly with little-endian memory layout, which is required in this case. Therefore I implemented a new method. Moreover it uses hash map instead of vector, which is much faster.

[2014-01-23 10:35:18] {reviewer}: Ok, please, mention it in the docs with a reference to FEATURE_56417 API and change method's name according to the coding guidelines: https://internal-resources.com/guidelines/1643as32

[2014-01-23 14:01:39] {developer}: done

When review is completed, reviewer creates a review message with his/her feedback and a review label. Label can have one of the following values: -2, -1, 0,
+1, +2. Normal developers are allowed to give the following labels: -2, -1 and +1. Architects and design leads are allowed to give the same labels, as developers and also +2. Receiving +2 means that change is approved and can be merged. Otherwise, if design lead or architect hasn’t approved the change, it cannot be merged and should be modified by the change’s author. Example of the review message:


There is also another type of review messages, created by automated tools environment before and after performing build and test or static code analysis. Here are 2 examples of messages from build system:

[2016-02-02 09:19:01] {Automated Build Tool}:
Patch Set 8: Build Started
https://internal-build-system.com/21893/

[2016-02-02 09:34:54] {Automated Build Tool}:
Patch Set 8: Build Failed
https://internal-build-system.com/21893/ : FAILURE

After finishing static code analysis automated tools environment also creates a Gerrit message, if any issues were exposed. Such message looks like the following:

Patch Set 10: Code violations detected on changed java lines: https://internal-code-checker-system.com/254231/
Top 3 occurring of 1 violations:
5 checkstyle: 2:coding.EqualsAvoidNullCheck

When the review process is completed, message with +2 was received, the author of the change starts merging process. In case of successful merge Gerrit system created a last message, which informs about successful merge:

[2014-01-09 12:31:51] {Gerrit Code Review}: Change has been successfully merged into the git repository.

However, sometimes merge process may require additional work due to merge conflicts. In this case Gerrit creates corresponding message and the author of the change has to resolve all conflicts and try to merge the change again. Here is an example of message, which informs about unsuccessful merge:
Chapter 3. Results

Change could not be merged because of a missing dependency.
The following changes must also be submitted:
I4145a6d45c6e456a4d56e4b0055ac22686fab613ad Please rebase
the change and upload a replacement commit.

There are also other cases, when changes are getting abandoned. In this case
the author of the change creates a corresponding review message:


In many examined cases this message, when some time already passed. Some-
times this message is created after a day or two, sometime after few months.
In addition to the data from Gerrit, information about teams setups was
extracted from team setup files, provided by management stuff of the studied
project.

3.2.4 Tools, used during the research

For the study purpose, SPAT database was filled with code review data, pro-
duced from May 2011, when the studied project switched to Gerrit Code Review
Tool, till the end of April 2016. Detailed information about studied database is
presented in the table 3.3.

Table 3.3: Detailed information about studied SPAT database

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall number of commits</td>
<td>39 724</td>
</tr>
<tr>
<td>Overall number of changes</td>
<td>12 283</td>
</tr>
<tr>
<td>Overall number of review comments</td>
<td>109 008</td>
</tr>
<tr>
<td>Overall number of changed files</td>
<td>473 270</td>
</tr>
</tbody>
</table>

Visualization of the metrics collection results was performed using Matlab
environment. The data that was used to generate plots, presented in this Thesis,
was produced from February 2014 till April 2016 by 2 company’s development
sites, mentioned here as Site A and Site B (see subsec 3.2.1). Four types of plots
were created in Matlab to present results, gained from the set of data metrics:

1. Normal plot (see fig. 3.4). X axis represents timeline and Y axis represents
metric value. Each datapoint on the plot is marked with a text label, which
contain metrics value (the value without brackets) and a list of changes,
identified by change number (the value in squared brackets). Datapoints are
connected with a normal line. In addition, a linear regression line (dotted
line) is drawn on plot, presenting data from specific site. In the middle of the linear regression line there is a colored text label, which contains mean value for current dataset. This plot can contain data from multiple datasets.

2. Plot with average metric values (see fig. 3.5). Is similar to the Normal one, except for the lack of number of specific changes. Each datapoint presents an average value for 1 month, week or day.

3. Histogram-1 (see fig. 3.6), which represents distribution of metric values. Datapoints (value without brackets) are also marked with a text label with a list of changes (the value in squared brackets). X axis represents value of the metric and Y axis represents number of changes with corresponding metric result. This plot contains data only from one site. Datasets from different sites are presented with the same scale. To make the plot more informative, text labels change their color from maximum value in all datasets (red color) to minimum value (yellow color).

4. Histogram-2 (see fig. 3.7) is similar to Histogram-1. The only difference is that data in Histogram-2 is grouped into bars. Each bar is also marked with colored text label, which contains unique values, containing in specific bar.

It is important to note, that the presented below figures are just screen shots of the interactive plots, created in Matlab. Some of the plots might have to much overlapping details due to high density of datapoints (for example 3.8). Such a look of the fires remained untouched to provide an overview of the bigger picture. Since this study is not an analytical study, the below figures should be considered only just as illustrations of the extracted data. The detailed analysis is available using the interactive Matlab plots with the possibility of the zooming and moving through the plot. For example as it is done on the figure 3.2.
Figure 3.2: Zoomed plot of the Inspection time metric
Title of each plot contains the name of the metric and list of features, which contained the analyzed changes. Average metric value plots contain also duration of a time frame, used to build a plot. Histograms contain the name of the development site, where the presented results were produced. The following subsections present examples of the plots, build from the analyzed data.

3.3 Research results in the context of the company

Figure 3.3 presents the existing product quality mechanism, employed by the studied company, and contribution of the results of this Thesis in its context. Before this research the only quality evaluation approach, which was used in the studied project, was post-release evaluation, based on the number of defects (Trouble reports), exposed during 'Testing and deployment' phase of the product life cycle. This approach allows to evaluate quality of the final product, but does not expose the developers’ performance issues. In particular, it does not allow to compare quality of the performance of the particular sites. During this research the author concentrated on the ways of extracting performance issues from the code review database using a set of data metrics.
Figure 3.3: Two quality evaluation mechanisms: test-based and code review data-based
3.4 Metrics application experience

This subsection presents the experience, gained by the author during the application of the proposed set of metrics. The results of activities, described below, were used to answer the RQ 2. Each metrics is discussed separately. In addition, some metrics contain the authors reflection on the metric, based on his practical experience. Some of the found metrics were not implemented. In such cases the reason of skipping the metric is also provided.

3.4.1 Vulnerability-related comments count

Before application vulnerability-related comments count metric to the analyzed set of changes the author applied it to the whole database to identify the overall rate between false positives and comments, which are really related to software vulnerabilities. The results of execution SQL query contained 1565 review comments, which contained one or more keywords from the table 3.2. Brief analysis showed that found comments can be roughly divided into two groups:

- Comments, which might be vulnerability-related, but it is hard classify such comments without knowledge about the comment context;
- Comments, which contained such keywords, as "integer", "SQL", "access", "format", "string", e.t.c. Most of these comments were not discussing any security issue;

This short experiment showed that identification of vulnerability-related comments, based keywords from the table 3.2, is quite noisy and the results of this metric should be carefully checked and filtered. Complete analysis of 1565 found review comments was not the main goal of this part of case study. After short discussion with software architects it was decided to skip this metrics due to big amount of required manual analysis and low frequency of security-related issues, exposed during code review process.

3.4.2 LOC and sLOC

Sources: [P6]

Proposed measurement approach: Intentionally, this metric should present the number of lines, which were reviewed during the code review. In other words, the metric presents number of inserted, removed or modified lines of code.

Customization of the measurement approach: Since Gerrit API presents all changes to the source file as inserted or removed lines, in this research LOC metric was calculated as a sum of deleted and inserted lined during each review. sLOC metric was not calculated, since measuring the executable code was not needed during this research.
**Authors experience:** Practice shows [32] that smaller size of code review indicates high quality of the development process due to limited effectiveness of reviewer during long reviews.

**Experience from the case:** Figure 3.4 presents the average number of LOC per review, calculated from code review data from *Site A* and *Site B*. According to the company’s experts, comparing to *Site A*, changes from *Site B* were often quite big and took long time to perform the review. This problem can be perfectly seen on the plot. On the other hand, *Site B* improved their results and started to submit smaller changes. It is hard to see it on the datasets line, but linear regression line shows that the mean value is decreasing with time. It can be seen on the figure 3.5 which present average metric value per month.

This tendency can also be seen on the histograms. In particular, most of the *Site A* metric values are yellow-labeled and values from *Site B* are significantly closer to the red zone (the region of the plot with red labeled data points).
Chapter 3. Results

Figure 3.4: LOC per review
Figure 3.5: Average LOC per review
Figure 3.6: LOC per review Histogram-1, Site A and Site B
Figure 3.7: LOC per review Histogram-2, Site A and Site B
3.4.3 Pre-review time

Sources: [P5]

Proposed measurement approach: Time interval between the moment, when the author uploaded the change and the moment, when the test build was performed, initial tests were executed and the author marked the change as "ready for the review".

Experience from the case: During analysis of the code review data it was discovered that in the studied environment pre-review time of the change cannot be detected due to lack of such rule. Some changes received +1 from the author, after being uploaded and build and initial tests were successfully executed, but overview of the code review process in the studied environment showed that not all changes' authors are doing it in the same way. Therefore, collection of this metrics was impossible in the studied environment.

3.4.4 Inspection time or Review time

Sources: [P5] and [P6]

Proposed measurement approach: The amount of time, which is needed to perform the review. The source [P6] suggests to measure it in hours, with 6 minute accuracy. Inspection time starts when author gives his/her change +1, which means that the change is ready for the review, and ends, when the change receives +1 from reviewer, who has to approve the code.

Customization of the measurement approach: Calculation of this metric in the studied environment was performed in a way, different to the description from [P5]. This was caused by the following facts, discovered during analysis of code review data in the studied environment:

1. It is not always possible to identify pre-review time (see description of the previous metric).

2. In the studied environment review finishes, when change receives +2.

3. Some changes receive +2 more than once. It happens when the change receives first +2 and is allowed to be merged. However, due to merge conflicts the author has to resolve the conflicts and upload modified version of the change as a new patch-set. The new also patch-set must be review by the reviewers and cannot be merged, until it will receive a new +2 label. It means that review continues until the last +2 label was received.

4. Not all changes were merged. Some of them were abandoned, another left open for unknown reason. As it is mentioned in the subsection 3.2.3, actual review ends before the moment, when the message about abandoning the change was created.
Therefore, in this research *inspection time* metrics was calculated as a difference between the moment, when the first patch-set of the change was uploaded to Gerrit, and the moment, when the change received the last +2 label or moment, when last message, not related to change abandoning, was created. Weekends were subtracted from review time duration.

**Authors experience:** In addition to the metric’s description in [P5] and [P6] this metric may also be influenced by level of experience of the reviewer. If the reviewer is not familiar with the reviewed component, it might take much more time to review the code. Also long duration may indicate high complexity of the code itself or its integration, due to a big number of potential problems, which reviewer has to analyze. Moreover, it takes a long time to perform a review of the code, which has bad quality, since reviewer has to point out and comment each defect he/she found.

**Experience from the case:** According to the company’s representatives, at the beginning Site B had longer reviews, than Site A, but after some time passed and the developers from Site B gained some experience, the reviews started to takes less time. It is hard to see it on the plot with average values (fig. 3.9). The histograms (fig. 3.10 and 3.11) confirm it, but the difference between both sites is not that big. The trend is visible on the fig. 3.8. The mean value of the Site B is approximately 1.5 time bigger, than the mean value of Site A. It also can be seen that during the first months reviews for the Site B were quite long. Improvement trend can hardly be seen from the data points, but linear regression line shows quite intensive decrease of the metric value for Site B.
Chapter 3. Results

Figure 3.8: Inspection time
Figure 3.9: Average Inspection time
Figure 3.10: Inspection time Histogram-1, Site A and Site B
Figure 3.11: Inspection time Histogram-2, Site A and Site B
3.4.5 *Inspection rate*

**Sources:** [P6]

**Proposed measurement approach:** Speed, at which certain amount of code is reviewed. Measured in \( \text{LOC divided by inspection hours} \). High values mean that big amount of code was reviewed during short time.

**Experience from the case:** According to the original description of this metric, it shows how fast the review was performed. On the below figures (fig. 3.12, 3.13, 3.14, 3.15, 3.16) it can be seen that for *Site A* there were conducted more reviews, when big amount of code was reviewed quickly. The big difference can be seen both on the plots and on the histograms.
Chapter 3. Results

Figure 3.12: Inspection rate
Figure 3.13: Average Inspection rate
Figure 3.14: Inspection rate Histogram-1, Site A and Site B
Figure 3.15: Inspection rate Histogram-2, Site A
Figure 3.16: Inspection rate Histogram-2, Site B
3.4.6 Pre-integration time

**Sources:** [P5]

**Proposed measurement approach:** Time interval between the moment, when reviewer finished the review and accepted the change, and the moment, when the integration of the change started.

**Authors experience:** As it is mentioned in the [P5], this metric may indicate completely different things in different projects. For instance, if integration took long time in a stable mature project, which does not constantly it’s architecture and key elements, most probably, it means that development process of the change was poorly planned and led to integration of too big amount of code. Also it may indicate that the code itself was not properly reviewed from the integration perspective.

On the other hand, in case of dynamically changing project long integration time maybe caused by the nature of such development approach and might be typical within such project.

**Experience from the case:** Due to lack of such markers in the studied environment, it was not possible to collect this metric.

3.4.7 Integration time

**Sources:** [P6]

**Proposed measurement approach:** Starts, when the integration process begins and ends, when the change is being merged.

**Experience from the case:** Company’s experts have not noticed any significant difference between integration process in both sites. However the figures below (fig. 3.17, 3.18, 3.19, 3.20) show, that integration process for the code, produced in Site B takes more time. The mean value of the metrics for Site B on the figures 3.17 and 3.18 is approximately 2 time bigger.
Figure 3.17: Integration time
Figure 3.18: Average Integration time
Figure 3.19: Integration time Histogram-1, Site A and Site B
Figure 3.20: Integration time Histogram-2, Site A and Site B
3.4.8 Defect count, Defect rate and Defect density

Sources: [P6]

Proposed measurement approach: Amount of defects, which were found during the review.

Customization of the measurement approach: During this research defects are counted as number of violations, detected by automated tools environment;

Authors experience: Low defect count does not necessarily mean low effectiveness of the review. For instance, it might mean a low number of defects in the reviewed code. Thus, within this research this metric was used for indicating number of actually discovered defects.

Experience from the case: Despite the fact that according to the company’s experts developers from Site A are more skilled and write code of better quality, than developers from Site B, the results captured by this metric does not show that big difference between the sites. In fact, on the plot with the data points from all changes (fig. 3.21) Site B has higher mean value and higher pick value. On the other hand, on the plot with average values (fig. 3.22) Site A has the higher mean value. Histograms (fig. 3.25 and 3.26) also do not show a big difference between the sites, so the company’s experts conclusions in this case seems to be not entirely correct.
Figure 3.21: Number of violations
Figure 3.22: Average Number of violations
Chapter 3. Results

Figure 3.23: Violations density
Figure 3.24: Violations rate
Figure 3.25: Number of violations Histogram-1, Site A and Site B
Figure 3.26: Number of violations Histogram-2, Site A and Site B
3.4.9 **Number of comment conversations**

**Sources:** this metric was proposed by the company’s experts during the discussions about the studied sites performance differences.

**Proposed measurement approach:** Number of conversations about specific fragment of code that happens within code reviews.

**Experience from the case:** The companies experts emphasized that code, produced by *Site B*, has much more defects and needs more time to be reviewed, than code, produced by *Site A*.

The interesting thing is that *Number of violations* plots (fig. 3.21, 3.22, 3.25, 3.26) doesn’t show that big difference, but from the below plots (3.27, 3.28, 3.29, 3.30, 3.31 and 3.32) it can be clearly seen that much bigger number of comments was created during code reviews for *Site B*. In this case both the plots and the histograms expose a noticeable difference.
Figure 3.27: Number of comment conversations
Figure 3.28: Average Number of comment conversations
Figure 3.29: *Number of comment conversations* Histogram-1, *Site A*
Figure 3.30: Number of comment conversations  Histogram-1, Site B
Figure 3.31: *Number of comment conversations* Histogram-2, *Site A*
Figure 3.32: Number of comment conversations Histogram-2, Site B
3.4.10 **Number of patch-sets**

**Sources:** [P5]

**Proposed measurement approach:** Number of patch-sets in a single change.

**Experience from the case:** The company’s experts noticed that code from Site B often needs more rework and has more patch-sets. It can be seen on the below figures that the number of patch-sets is noticeably higher for Site B. In this case again the plots (fig. 3.33 and 3.34) and the histograms (fig. 3.35 and 3.36) show, that during the studied period Site A produced less patch-sets per change, than Site B.
Figure 3.33: Number of patch-sets
Figure 3.34: Average Number of patch-sets
Figure 3.35: Number of patch-sets Histogram-1, Site A and Site B
Figure 3.36: Number of patch-sets Histogram-2, Site A and Site B
3.4.11 *Abandoned changes count*

**Sources:** [P5]

**Proposed measurement approach:** Amount of Gerrit changes that are abandoned and not merged.

**Experience from the case:** The company’s experts haven’t noticed any trends in changes abandoning in *Site A* or *Site B*. However, the plot 3.37 shows that during the studied period developers from *Site A* have abandoned 2 times more changes, than developers from *Site B*, which is an interesting observation.
Figure 3.37: Number of abandoned changes
3.4.12 Number of negative review labels

Sources: [P5]

Proposed measurement approach: Count of -1 and -2 review labels.

Authors experience: This metric could give quite spurious results. For instance, big number of +1 labels might be considered as a good results, but actually will be caused also by big number of patch-sets, which is, rather, a bad sign. Thus, in this research the this metrics will be used as rate of positive and negative labels.

Experience from the case: Similarly to the previous plots, this set of plots (fig. and ) and histograms (fig. and ) shows that the number of negative labels from code review for Site B is significantly higher, which also correlates with high number of patch-sets. The same observation was done by the company’s experts.
Figure 3.38: Number of negative labels
Figure 3.39: Average Number of negative labels

Chapter 3: Results
Figure 3.40: Number of negative labels Histogram-1, Site A and Site B
Figure 3.41: Number of negative labels Histogram-2, Site A and Site B
3.5 The list of metrics revised from the perspective of application in the studied environment

The following table presents the revised list of metrics, that were found during the conducted literature study. Is also contains the explanation, why a particular metric currently cannot be used in the studied environment, if it is so.
### Table 3.4: The revised list of data metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Is applicable in the studied environment</th>
<th>If not applicable, why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>sLOC</td>
<td>No</td>
<td>Measuring the executable code was not needed during this research</td>
</tr>
<tr>
<td>Inspection time or Review time</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Pre-review time</td>
<td>No</td>
<td>In the studied environment the metric cannot be currently applied due to lack of required steps in the employed code reviews process.</td>
</tr>
<tr>
<td>Pre-integration time</td>
<td>No</td>
<td>Same as previous.</td>
</tr>
<tr>
<td>Integration time</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Inspection rate</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Defect count</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Vulnerability-related comments count</td>
<td>Yes, but was skipped</td>
<td>The metrics, however, was skipped, due to a big noise in the results, produced by the proposed keywords and low number of vulnerability-related comments in the code reviews, according to the company’s experts.</td>
</tr>
<tr>
<td>Number of comment conversations</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Defect rate</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Defect density</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of patch-sets</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Abandoned changes count</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of positive and negative review labels</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### 3.6 Semi-structured interview results

This sections presents the results of validation of the research results with the company’s experts. The collect the feedback from the company’s representatives
the author performed with them a series of semi-structured interviews. The answers of all interviewees were grouped by questions and combined together into one compound answer to each question that contains answers from all interviewees.

1. **Are the results of the research useful for the company in terms of developers’ performance analysis?**
   - if yes, how can it help in developers’ performance analysis?
   - if no, why?

   All the interviewees responded that results of the performed research are useful for the company. Most of the interviewees emphasized that possibility to analyze performance of different sites and teams separately gives an opportunity to perform a complex analysis of the developers’ performance. In particular, it enables observation of learning progress of new teams and compare it to more mature teams. Possibility to observe trends in specific site performance is also very helpful. Company’s experts underlined the particular use of some data metrics, such as patch-sets number per change, modified lines number per change or conversations number. Submitting bigger changes for code review was pointed as a common problem in some teams, especially new ones. Applying these metrics and visualizing the results allows to have more insight in teams performance evolution. Abandoned changes number or inspection time and inspection rate metrics help to expose facts of too long and too short reviews, which were not considered that much previously. Therefore it gives an opportunity to observe developers’ performance issues that company’s experts were not aware of previously.

2. **Which results of the research were unexpected?**

   Some metrics plots, such as Inspection time, Code violations number or Number of abandoned changes showed quite surprising trends, since initially company’s experts were persuaded that performance of the Site A is much better, than performance of Site B due to its maturity and it does not have any negative trends in it. However, the data, collected using the above metrics, exposed the issues in the performance of the Site A that were unexpected and unseen due to lack of a appropriate tool for collecting and visualizing data.

3. **Which of the results are less useful or completely irrelevant? Why?**

   Since the research was performed in a strong cooperation with company’s experts, all activities within this research were consulted with company’s
representatives and all questionable steps were discussed in details to prevent any redundant activity. Also, company's representatives emphasized that even in cases, where no any trends or correlations were visible from the plots, it is still a useful result from both research in analytical perspectives. Therefore none of the results of the research was considered as useless.

4. **What was missing in the results of the research in your opinion?**

All interviewees are satisfied with the results of the research. Several persons emphasized that presented results itself don’t provide a performance analysis, but since it was not a main goal of the research, it was not pointed as a drawback of the research. The objectives, which were discussed with the company’s representatives from the beginning of the research, were changing slightly during the research performance due to discovered differences between theory and practice, but the general idea of the research remained unchanged. In addition, the continuation of the research was performed during the summer job after the Thesis was finished, where the author worked on extension of the list of metrics and ways of the representation of the metrics output.

5. **Is there a need to continue or expand the current research?**

   - if yes, what should be included in the next research?
   - if no, why?

During the interviews the company’s representatives shed a light on ongoing research, performed by doctoral student that extensively uses proposed metrics and visualization tool for collecting performance-related data and performing its statistical analysis.

6. **What contribution was made by the performed research?** For the company use perspective the performed research developed a tool for decision support on how to distribute work and optimize performance of multi-site software projects. In addition, the collected data metrics and created plots allow company’s experts to monitor the learning and knowledge sharing processes between more and less mature teams and sites.

7. **Who will use the results?** The created data collecting and analytics tool inside of the company mainly will be used by managers and software architects. Additional group of people, who will also extensively mine and analyze code review data using the created tool are other researchers, who are taking part in ongoing research and the subsequent researchers, related to analysis of application of the distribution development model in the company and sharing knowledge between development teams and sites.
During this research the author performed several activities, such as literature study, studying archival data and development environment and performing discussions with company’s employees in form of semi-structured interviews. In this chapter the performed activities will be discussed and threats to validity will be reported.

4.1 Discussing the findings of the research

Collection and analysis of the literature sources, related to the topic of the Thesis, gave an insight to the state-of-art and allowed to base current research on the previous work, performed by other researchers. The main purpose of the literature review was to understand, how the code review data shall be approached in order to use it for analysis of the quality of software developers’ work. As it was mentioned in the section [1] the quality assurance and evaluation mechanisms, which were used in the studied project, were mainly focused on the number of trouble reports, created during the testing phase. On the other hand, this mechanisms did not allow, to identify the issues, related to the quality of the development process itself. For instance, software testing does not allow to verify, how much effort was payed by the code reviewers to keep the quality of the code on appropriate level in the cases, when some of the development sites perform worse, than others. To address this problem the author performed a systematic literature study, which was intended to collect reported knowledge about the code review data measurement and development issues identification, based on it.

The approach described in the found literature sources is based on the mining code review data and extracting performance-related metrics. From the 266 literature sources only 6 were closely related to the topic of this research and contained relevant information about the mining of code review data in order to obtain quality indicators. Despite the fact, that quite broad search string was used to formulate the start set for the literature study, quite significant part of the found sources were rejected after analysis of their content. Most of the found sources were focused on the code review process itself and very few of the found sources were related to the data mining of the code review. At the same time the
example of the studied company showed, that existing quality assurance mechanisms do not identify the performance issues, which cause a big work overload for the company's employees. This observation points to quite broad field for further studies in this field.

During the literature review, performed as a part of this research, 15 different development performance metrics were found. Each metric allows to analyze quality of the development process from certain perspective, such as average length of the review or amount of required rework during the code review phase. The author studied the way of collecting data samples as it was proposed in the reviewed sources, then examined the way of working in the given environment and adapted the found techniques to the project under the research. On the one hand the work performed during this research is based on a compilation of previously performed researches. On the other hand during the application of the existing techniques the author earned new experience from the case, which complements the previously gained knowledge. The performed research demonstrated, for instance, the importance of complex analysis of the issues indicated by the collected metrics and its strong dependence on the actual context of the development process.

The results, presented in this paper, cannot be considered as completely generic. The results of the literature study, such as list of relevant and irrelevant sources and list of data metrics, can be reused by other researchers in a context of different software product. However, the revised list of metrics and the collected experience is very case specific, since the modifications, done to the original metrics, are caused by the features of the studied environment. This part of the results cannot be generalized, but for sure would be useful for the future researchers and software companies. The future researchers can refer to the results of this study as a related experience. The key employees of software companies might be interested in the extracted quality indicators and also the reasons, why some metrics were modified and how the code review process in their environment can be improved in order to make data mining of the code review process more efficient.

Obtained opinion of the company's key employees helped to verify the results of the performed research and identify potential directions for the future work. For instance, the performed interviews helped to understand, which of the found metrics are considered as the most valuable for the managers and software architects, what is overall use of the performed work and how it will be used by the company to solve the performance issues.

4.2 Threats to validity

According to the guidelines for the case study [33] the author evaluated the validity of the performed study based on the following criteria: construct validity, internal validity, external validity and reliability.
Chapter 4. Discussion

4.2.1 Construct validity

"This aspect of validity reflects to what extent the operational measures that are studied really represent what the researcher have in mind and what is investigated according to the research questions" [33]. Within this research the author identified the following threats to construct validity: 1) the search strings, used during the literature study, might not cover a part of the literature sources, related to the topic of the research; 2) the author might miss some databases, which might contain relevant literature sources. To mitigate the first issue the author selected Snowballing Literature Study method. This literature review method which considers not only the sources, that contain the selected search strings, but also recursively examines the references, used in the found sources. In order to extend the number of the found sources the author used Google Scholar search engine, which performs search in various databases of scientific papers and helps to find papers, published by different publishers.

4.2.2 Internal validity

This aspect of validity reflects the situations, when the studied factor is influenced by a different factor, which might be unknown for the researcher or the effect might bias the results [33]. Since the found metrics were formulated in a different environment, not all of them turned out to be applicable in the studied project. This fact was discovered during the analysis of the description of the found metrics and the features development process in the studied environment. To create a tool set, applicable in the studied environment, the author had to modify or eliminate some of the metrics and create a revised list of metrics. Most of the modifications were connected to the lack of specific "markers" in the code review data or significant differences in code development procedures, employed in the studied projects and the original project, where the metrics came from. Since the significant part of the analyzed code review data was produced by human, it is very hard to analyze and interpret such data properly without knowing the context. For instance, meaning of some of code review comments or lack of some of the "markers", such as +1 right after the moment when the reviewer finished the preparation of the code and considers it as ready for the review (Pre-review time metric, see table 3.1). To mitigate this risk the author performed the metrics revision process in cooperation with the company’s employees in order to learn, why specific metrics cannot be applied and find a way, how it can be adapted to the studied environment.

4.2.3 External validity

This aspect of validity reflects, to what extent the results of the study can be generalized. The potential threat to validity here is the fact, that the whole study
was performed within one project and was focused on a data sample, produced mainly by 2 development sites. On the other hand the main purpose of the research was to find a way of extracting performance-related metrics and not to perform the complete analysis of the sites' performance. To perform the data measurements the author adapted found metrics to the studied environment. In a same way the same set of metrics, used within different project or different company, might lead to different results. Therefore the results of this study cannot be completely generalized. The results of the literature review (the list of the reviewed papers and list of found metrics) can be generalized. The results of metrics adaptation and application are case specific.

4.2.4 Reliability

This aspect of validity reflects to what extent the results of the research are dependent on the specific researcher. The author identified several potential issues here. First, the formulation of the search strings for the Snowballing literature review was done based on the author's experience and knowledge. Another researcher might formulate different search strings and create a different start set for the literature review, which can influence its results. Second, the results of the performed interviews were analyzed and summarized by the author in an informal way and the answers to the questions were combined in one complete feedback. The same results might be interpreted and summarized differently by a different researcher. To mitigate this issue the author sent the summarized answers to the interviewed company's employees in order to verify, if their feedback is summarized correctly and was not misinterpreted by the author of the Thesis.
Chapter 5
Conclusions

After performing this research and presenting the results to the company’s experts the author came to a number of conclusions and answers to the research questions, formulated at the beginning of the study. In addition this chapter contains the overview of the activities, which can be performed in future as a continuation of this research.

5.1 General conclusions

1. During the literature review it was discovered that not that many literature sources, found by used keywords, cover the chosen topic.

2. Collected set of data metrics gives a good insight to the developer’s performance and can be used for the complex statistical analysis.

3. In most cases, metrics results correlate with company’s experts opinion about studied development sites, which underlines the reliability of the chosen metrics.

4. It is important to visualize the results from different perspectives using different kinds of plots. I.e. some issues were not seen in the normal plot, but were clearly seen on the plot with average values or histograms.

In order to complete the aim of this research, the author found the following answers on the research questions for this research:

[RQ 1.1] What developers’ performance-related metrics can be computed based on the data, available from code reviews? During this research 14 metrics were found during the literature review and 1 metrics was proposed by the company’s experts. 3 metrics from the reviewed literature sources were found currently not applicable in the studied environment and 1 metric was skipped due to a big noise, produced by it in the results. The revised list of metrics is presented in the table 3.4.

[RQ 1.2] How shall these metrics be analyzed in order to evaluate developers’ performance? The examples, presented in this Thesis, show that selected metrics
can be used in statistical analysis. However, without knowing the context such analysis might lead to wrong results. Therefore it is important to discuss the results with company's representatives.

**RQ 1** How code review data can facilitate performance analysis? Code review data can be a big help in performance analysis, since it contains a big number of performance quality indicators, such as number of issues, amount of rework, e.t.c.

**RQ 2** What developers’ performance-related issues are visible through the selected set of metrics, computed based on code review data from different sites of the company? Proposed set of metrics indicates a number of developers’ performance issues. The list of the captured issues is presented in the subsection 3.1.1.

**RQ 3** How the results of the research can help the company in solving developers’ performance-related issues? Semi-structured interviews that were conducted after the research was complete, showed that the results of experimental application of selected metrics on company’s data strongly correlate with company’s experts observations of developers’ performance issues. The results of this research are extensively used in the ongoing research, performed by a doctoral student.

### 5.2 Future work

During this research the author discovered various potential directions for future studies, related to the results of this thesis. In this section the author will 3 main ideas for future researches.

The first potential direction for the future research is identification, which issues are still not covered with the existing list of metrics and which metrics should be created. For example, such research as well can be performed in one or several software projects in cooperation with the company’s experts. As an input data the researcher can obtain a list of code review issues, noticed by the company’s experts and as an output – the list of performance indicators, extracted using the created set of metrics. If some of the issues, reported by the company’s representatives, will not be identified by the existing metrics, the list of the missing indicators with the specification of the required metrics can be proposed.

The second potential direction is a research based on developers’ performance analysis and evaluation. During this research the author created a tool set for such study and its application in an analytical research will allow to verify this tool set and perform complete analysis of performance of multiple development...
teams in the industry. As it was mentioned before, such research was started simultaneously with this research in the same company. However, such analytical research might be performed in different companies with different development context.

Another potential way of the continuation of the work, performed during this study, is comparison of the created analytical tool set with existing code quality evaluation mechanism. For example, comparison with evaluation mechanism, based on code quality ISO standards [34] and [35]. For instance, the set of metrics, collected during this research allows to identify the problems, which are "unseen" for the traditional quality evaluation mechanisms, based on the source code analysis and number of post-release defects. The new study might be based on the comparison of the quality of the code, based on the above ISO standards and the tool set, created during this research.
Appendices
Appendix A

Start set creation: the list of rejected candidates


[C8] McIntosh, Shane, et al. "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and
Appendix A. Start set creation: the list of rejected candidates


Appendix A. Start set creation: the list of rejected candidates


[C21] MSRG Mobile. "Bug Triaging". **Reason of exclusion: the result is a citation.**


Appendix A. Start set creation: the list of rejected candidates


Appendix A.  Start set creation: the list of rejected candidates


[C56] S. Datta. "Perspectives on task ownership in mobile operating system development (invited talk)" 2014. **Reason of exclusion: out of scope (abstract analysis).**

[C57] S. Kaspar. "PHOENIX". **Reason of exclusion: out of scope (full text analysis).**


[C63] M. Mäntylä. "Software evolvability-empirically discovered evolvability issues and human evaluations" 2009. **Reason of exclusion: one of already processed papers is a part of this source.**


[C65] M. Mäntylä. "Static code analysis and code". **Reason of exclusion: the result is a citation.**


Appendix A. Start set creation: the list of rejected candidates


[C74] M. Verstraeten. "Tool support for security code review on smart cards". **Reason of exclusion: out of scope (full text analysis).**

[C75] EAC AdvAnTAgE "TSP SYMPOSIUM 2009" 2009. **Reason of exclusion: one of already processed papers is a part of this source.**


[C78] A. Bosu, J.C. Carver, M. Hafiz, P. Hilley. "When are OSS developers more likely to introduce vulnerable code changes? A case study" 2014. **Reason of exclusion: out of scope (abstract analysis).**

[C79] V.J. Hellendoorn, P.T. Devanbu, A. Bacchelli. "Will they like this?: evaluating code contributions with language models" 2015. **Reason of exclusion: out of scope (full text analysis).**


Appendix A. Start set creation: the list of rejected candidates


[C83] Bird, Christian, Trevor Carnahan, and Michaela Greiler. "Lessons learned from building and deploying a code review analytics platform." 2015. **Reason of exclusion: out of scope (full text analysis).**

Appendix B

The list of studies included in the literature review


Appendix C
Pilot study: description and results

To evaluate amount of reported sources about the topic the author performed a string search for a systematic literature study about code review data analysis. The search was performed using Google Scholar search engine and the following search strings:

1. "code review data" AND "literature study";
2. "code review data" AND "literature review"

To identify relevant sources the title of the source was analyzed firstly. If the title was relevant, the abstract of the paper was analyzed to decide, if the source is relevant, or not.

The first search string gave 2 results:


The second search string gave 6 results:


1 https://scholar.google.se/
Appendix C. Pilot study: description and results


In both cases any systematic literature study about code review data analysis was not found. Therefore, the author decided to perform a systematic literature study as a first part of my research.
 Appendix D

Description of the performed Snowballing literature study

Start set

Start set was prepared using database search. As it is advised in [24], to avoid a bias, caused by using only one database, the author of this Thesis used Google Scholar engine to perform the literature study. To find sources related to analysis of code review data the following search string was used: "code review data". Usage of this search string implies wider spectrum of sources and, as it can be seen in ratio of found to included sources, it lead to a big amount of irrelevant sources in the search results. On the other hand, usage of more complicated search strings doesn’t help to achieve more accurate results. For instance, adding such keywords as "analysis", "performance", "code quality" or "metrics" did not improved ratio of relevant to irrelevant sources, but, on the other hand, excluded some sources, which did not match these keywords, but still contained relevant information. Thus, it was decided to use more broad search string and improve the lack of search precision by applying the exclusion criteria.

Search results contained 88 candidates. The below list presents only the sources, which were included to the further study, numerated as [P1] [P2] and so on:


Appendix D. Description of the performed Snowballing literature study


The candidates, which were excluded from the search, are listed in Appendix A as [C1], [C2], etc. The reason of exclusion is provided individually for each candidate.

Also, [C2] was not included to the study, but during the source analysis one citation of this source was found relevant and included to the further study:


Iteration 1: backward Snowballing

[P1] has 13 references, none of which was found relevant.

[P2] has 20 references. 13 sources were excluded based on title or place and context of the reference. 2 sources were already processed. 5 sources were found not relevant.

[P3] has 8 references. 3 sources were already processed. 5 sources were found not relevant.

[P4] has 53 references. 19 sources were already processed. 33 sources were found not relevant.

After full text analysis 1 source was found relevant and added to the further study:


[P5] has 18 references. 6 sources were already processed. 12 sources were found not relevant.

Iteration 1: forward Snowballing

[P1] was cited by 8 sources. All 8 sources were found not relevant.

[P2] was cited by 4 sources. 1 source was processed previously and 3 sources were found not relevant.

[P3] was cited by 4 sources. 3 sources were processed previously and 1 source was found not relevant.
Appendix D. Description of the performed Snowballing literature study

[P4] was cited by 5 sources. 1 source was processed previously and 4 sources were found not relevant.

[P5] was not cited.

Iteration 2: backward Snowballing

[P6] has no references.

Iteration 2: forward Snowballing

[P6] was cited by 45 sources. 11 sources were already processed. 2 source were written not in English. 32 sources were found not relevant.

Literature review summary

During the review 266 sources were analyzed, 6 sources were found relevant:

- 88 sources were analyzed during start set creation. 5 sources were included to the study.

- 133 sources were analyzed during I iteration. 1 source was included to the study.

- 45 sources were analyzed during II iteration. No new sources were include to the study.

Quite big part of the reviewed sources was related to code review process organization, its role in code quality assurance, different approaches to code review process, existing code review tools, e.t.c. Part of the excluded sources contained information, which was useful for general overview of the code review process and was used during the work on this research, but was not included to the further literature study due to being not directly related to the topic of the research.

Significantly small part of the processed sources was directly related to the topic of this research. This fact proves relative innovation of the topic and research contribution of this Thesis. On the other hand, the found sources give not so much precise information about analysis of code review data.
References


References


