Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

LEONARD KEIDEL
MASTER THESIS

EXPLORATORY DATA ANALYSIS ON CODE REVIEW DATA

Submitted on [DATE]

Supervisor: Michael Dorner
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

_____

[CITY], [DATE]

License

_____

[CITY], [DATE]

Abstract

Modern code review enables developers to conduct light weighted inspections using tools and is a well-established part of the software development process. However, it is not clearly defined if current processes and methods are done efficiently. In this paper, we examined a dataset from a tool supported code review process provided by a multinational software development company to determine and test indicators for code review quality. Within an exploratory approach, inspecting data from over 250 000 entries in 5 years of reviewing practice, we checked dataset characteristics, reviewer workload, reviewer selection and social network metrics. We found evidence in all revised categories that the company lacks in an efficient code reviewing process. Our results prove that current code review practice needs better defined standards and with usage of our indicators and numbers, future studies can compare their observed code inspection performance.

Table of contents

Table of figures

List of tables

1 Research Chapter

1.1 Introduction

Code review is a common practice in developing industrial and open source software. The generally known objective of code reviews is to find errors in source code files. But the process is beneficial for many other factors, such as knowledge sharing or community building as well. Fagan in 1976 introduced an inspection process conducted within a physical team meeting including four different roles and five phases to complete (Fagan, 1976). Since software development became more decentralized, like many open source projects, the previous elaborate code inspections changed to lighter weighted peer reviews. Replacing physical meetings, a developer could send his file via E-Mail to various developers. One of the peer needed to review the file and depending on the guidelines accept or let other developers continue the review process (P. C. Rigby, 2011). Nowadays modern code review is tool supported. These tools can already be automated (Balachandran, 2013), but usually still contain human interaction. Gerrit is an example for a platform specialized on code reviews used by open source projects like Qt, ITK, VTK or Android (Bosu & Carver, 2013; Hamasaki et al., 2013). An author can submit a patch to a repository predefining a set of reviewers, who then revise, comment and finally agree or disagree with the change. The patch in the end gets submitted depending on a specified number of positive reviews (Bosu & Carver, 2013). In industrial practice some companies develop review software based on their own wishes, such as *CodeFlow* from Microsoft (Bacchelli & Bird, 2013) and firms that use external developed tools, such as Cisco using *CodeCollaborator* (Cohen & Brown, 2012).

The importance of code review practice has been proven by several studies. According to an exploratory research on Open Source Software projects, unreviewed files have a two times higher chance to induce bugs to the software system. The same study could also verify that reviews positively affect the readability of source code (Bavota & Russo, 2015). McIntosh could find evidence that low code review participation leads up to 5 additional defects and shares a significant link with software quality (McIntosh, Kamei, Adams, & Hassan, 2014; Shimagaki, Kamei, McIntosh, Hassan, & Ubayashi, 2016). And beside finding bugs a qualitative research at Microsoft has shown that software engineers benefit from knowledge sharing and community building while reviewing code (Bosu, Carver, Bird, Orbeck, & Chockley, 2017).

Although the high significance, some companies still claim problems in their review processes. Microsoft revealed in a conference, that current workflows are not performed efficient bearing in mind the high costs. Accordingly code reviews needs a better understanding and improved best practices (Czerwonka, Greiler, & Tilford, 2015). In this paper we will investigate on indicators to determine review quality. To achieve this, we are doing a data analysis on code review data, which we collected from a multinational software development company. The dataset contains around 250 000 entries collected in over 5 years of industrial software development practice. The results stand out from others, as our company is in its sector obligated and regulated to do code review. Hence, we have a complete dataset, where no code can be submitted without an inspection.

The contributions of this paper are the following.

- Insight into the quality of a regulated industrial code review process
- Code review quality measures derived from literature and data analysis
- Author – reviewer network dynamics and metrics from industrial practice

The paper is structured as it follows. First, we will examine related literature and give an overview about quantitative and qualitative studies focused on code review performance as well as researches about social network analysis on code inspection dynamics. After defining our research question, we will describe all methods we performed and sum up the characteristics of our dataset. Building up on this we will present, conclude and clarify implications of all findings from our research.

1.2 Related Work

The following section gives an overview about related and relevant studies. We categorized the work in two sections. First, we present quantitative and qualitative research about metrics found defining code review performance. And in the second subchapter we will introduce studies focusing on social network analysis on code review data.

1.2.1 Code Review Quality

Qualitative Research

Several previous studies approached code review quality by surveying, interviewing or observing both reviewers and developers. An early investigation on technical reviews detects the reviewers expertise as the most relevant quality measurement (Sauer, Ross Jeffery, Land, & Yetton, 2000). Two researches surveying developers from NASA and the Mozilla project equally attributes that thoroughness and clarity define code review quality (Kononenko, Baysal, & Godfrey, 2016; Nelson & Schumann, 2003). The latest study focusing on code review at Microsoft strengthens earlier findings from Bacchelli and Bird that understanding the code changes are an essential challenge for reviewers. Furthermore reviewing in a timely manner, review size and managing time constraints are ranked the highest challenge of developers at Microsoft (Bacchelli & Bird, 2013; MacLeod, Greiler, Storey, Bird, & Czerwonka, 2017). All qualitative studies give valuable insights in problems and concerns of the practicing developers. In contrast, we will focus within our research paper on quantitative measurements building up on these findings.

Quantitative Research

Since the beginning of modern code review, huge amounts of data are accessible for quantitative researches. In 2008 Rigby et al. revised both E-Mail discussions as well as version control repositories of the Apache project. After reviewing over 4000 contributions, they derived best practices considering this project as example of efficient and state of the art peer review technique. The reviews should be conducted early and frequent, in small but complete contributions, working in small time with self-selected experts (P. C. Rigby, German, & Storey, 2008). In a follow-up study in 2012 as well as in 2014, examining over 100 000 contributions from six OSS projects, they could confirm that asynchronous, frequent and incremental reviews are the most efficient. They also stated out that experience is important and these experts need to be empowered by choosing their own files to inspect (P. C. Rigby, German, Cowen, & Storey, 2014; P. Rigby, Cleary, Painchaud, Storey, & German, 2012). Other quantitative research on Open Source projects found out that review participation and intensity shares a link with the quality of a completed inspection (Kononenko, Baysal, Guerrouj, Cao, & Godfrey, 2015; Thongtanunam, McIntosh, Hassan, & Iida, 2015). The closest approach to our research conducted by Izquierdo-Cortazar et al. used quantitative metrics such as activity, time-to-merge and patch-complexity to track review performance. This paper gives an automated methodology to inspect Linux style code review derived from their studies at the Xen and Linux Netdev project (Izquierdo-Cortazar, Sekitoleko, Gonzalez-Barahona, & Kurth, 2017)

In an industrial setting, Kemerer & Paul as well as Ferreira et al. evidence that the review rate is a crucial factor and by lowering them they could improve their process performance (Ferreira et al., 2010; Kemerer & Paulk, 2009). Microsoft investigated in a mixed qualitative and quantitative approach five of their own software development projects focusing on the usefulness of comments. The results implicate several recommendations for improving code review processes. First, agreeing with findings from OSS projects, they recommend choosing reviewers carefully by experience. Still unexperienced people should be included to gain experience. Second the results show that the effectiveness in patches with a high number of files in the changeset decreases. Hence developers need to commit small changes to facilitate the review process. Their last recommendation is to identify the weak areas of reviews using the framework they developed (Bosu, Greiler, & Bird, 2015). The most recent and relevant study from dos Santos and Nunes explored both technical and non-technical factors which affect code review effectiveness in Distributed Software systems. The outcome was that all inspected factors influence code review performance. Increasing file sizes, more involved teams or locations are all decreasing the effectiveness of reviews. As well in contrast to findings from other studies they point out that more participation has a positive effect on effectiveness as the duration is shorter and more precise. Two reviewers are the perfect size for inspecting code which is corroborated by a previous study from Rigby and Bird (dos Santos & Nunes, 2017; P. C. Rigby & Bird, 2013). In our study we will build up on these factors that define review quality and find indicators to track them in our dataset.

1.2.2 Social Network Analysis on Code Review Data

Preliminary work from Lopez-Fernandez in 2004 used Social network analysis on version systems of the Apache, GNOME and KDE project to learn more about structure and internal processes of the developers (Lopez-Fernandez, 2004). Ohira et al. probe cross project collaboration on data from over 90 0000 projects and introduced a tool for visualizing Social Networks among them (Ohira, Ohsugi, Ohoka, & Matsumoto, 2005).

Hamasaki applied Social network analysis to identify different roles of a code review process extracted from Gerrit (Hamasaki et al., 2013). Meneely et al. developed a failure prediction model using Social network analysis on code review data (Meneely, Williams, Snipes, & Osborne, 2008). A growing body of literature has divided the reviewers in a core and a periphery group using Social network analysis (Bosu & Carver, 2014; Huang & Liu, 2005). A similar approach from Yang demonstrate a correlation between activity and importance using Social Network Metrics (Yang, 2014b). Another research from Yang describes finding a link between network position and code review quality (Yang, 2014a). The most relevant study as well questioned if network centrality is an indicator for inspection quality. After examining precisely four big Open Source projects, they could validate that high network centrality correlates to a better bug report outcome. Based on this results they designed a bug report classification mechanism using nine measurements including different centrality metrics (Zanetti, Scholtes, Tessone, & Schweitzer, 2013). Our research will in contrast track the overall inspection performance and provide results to benchmark future research.

1.3 Research Question

In this paper the general research question is:

„What are quantitative indicators to measure code review quality?”

As we want to specify our approach we breakdown this general question in several subcategories. These will then be examined in an exploratory manner. We will derive our assump-

tions from recent literature and in a best case compare our findings directly with results from other studies.

RQ1: What are quantitative indicators to measure code review quality?

RQ 1.1: Which general dataset characteristics determine code review quality?

RQ 1.2: What are indicators for effective workload distribution of code reviewers?

RQ 1.3: How can we measure efficient reviewer selection?

RQ 1.4: Which social network metrics describe code review quality?

To answer the last Research Question RQ1.4. we evaluate the dataset within a Social network analysis and compare the results with the relevant work mentioned in chapter 2.2.

1.4 Research Approach

1.4.1 General Methodology and Approach

The research approach of this paper is divided into two methods, Exploratory Data Analysis and Social network analysis. To implement the exploratory data analysis, we used the methodology described by Tukey (Tukey, 1977). The data collection from our examined software company stores over 250 000 review entries.

To start we needed to deserialize and merge the given datasets. Next, to have a dataset with only completed reviews, we eliminated all entries that haven't been reviewed and accepted yet. Furthermore, we just considered data from 2013-2016, as 2012 and 2017 are not complete. Afterwards we started exploring our data for saliences based on the points we defined by RQ1-3. We used mostly descriptive and statistical methods, on which we interpreted and compared our results.

For the Social network analysis we took general methodology and metrics from both Freeman and Lopez-Fernandez (Freeman, 1978; Lopez-Fernandez, 2004). By applying those methods, we wanted to have a close look at the structure of the code reviewing dynamics at the software company and compare them with similar studies. As a first step we needed to define the actors and their corresponding relationships to build up a network of nodes and edges. In our created structure the network contains two different actors:

- **Reviewer**: Solely reviewer of code patches
- **Author**: Solely author or author and reviewer of a code patch

Coming with the regulations of the company's code review process, an author in a usual case reviews his own code before it gets committed. In our social network we didn't consider these self-reviews to focus on the social dynamics to other actors.

The relationship between these actors in our approach is defined as times an author requested a code inspection from a reviewer. Therefore, the resulting edges are weighted and directed. While the direction empowers us to differentiate between requesting and conducting a review, the weight illustrates the total intensity of both relationships. As a second step we ultimately built up a directed and weighted graph showing the number of interactions from author to reviewer. The resulting network was a first visual description and the foundation of the further metric analyzation. We additionally created another undirected graph, as some metrics are better interpretable in a bidirectional context.

Third we used several metrics to look at individual and average performances inside the network. And last, we compared the numbers and structure with other projects using literature re-

view. In conclusion we aimed to understand the social dynamics of industrial code review processes and find more indicators to track down code review success or failure.

The metrics we used were *in-* and *out-degree*, *degree centrality*, *closeness centrality*, *betweenness centrality* and *graph density*. We will explain the significance and meaning of the social network measurements in the following subchapter. We added some additional metrics (*page rank, eigenvector centrality*) for further references, without focusing and interpreting them inside our study.

1.4.2 Social Media Metrics

In the following chapter we will describe metrics based on a graph with **n** nodes calculated from $\mathbf{p_k}$ an undefined vertices in the graph.

Degree and Degree Centrality

The *degree* $\mathbf{C_D}$ of a network node $\mathbf{p_k}$ is the number of different connections to other nodes. The calculation is done by simply counting the edges at a given node, see formula below (Freeman, 1978).

$$C_D(p_k) = \sum_{i=1}^{n} a(p_i, p_k)$$

$$where\ a(p_i, p_k) = 1\ if\ p_i\ and\ p_k\ are\ connected\ by\ a\ line; else = 0$$

Within a directed graph we need to distinguish between *in-degree* and *out-degree*. In-degree is defined as the number of edges pointing to the node, while out-degree determines the number of edges going from the node to other nodes (Newman, 2014). In our example the *in-degree* counts from the reviewer angle the number of different authors who were requesting a code inspection. The *out-degree* in contrast is from the authors perspective the number of different reviewers he had. Therefore, it enables distinguishing between reviewer and author importance.

The *mean degree* or *degree centrality* $\mathbf{C'_D}$ puts the degree into a proportion to the network by showing how many percent of the network a node is connected to. The calculation for a node is done by dividing the degree with the total number of actors in the network (Freeman, 1978; Newman, 2014).

$$C'_D(p_k) = \frac{\sum_{i=1}^{n} a(p_i, p_k)}{n-1}$$

$$where\ a(p_i, p_k) = 1\ if\ p_i\ and\ p_k\ are\ connected\ by\ a\ line - else = 0$$

The *degree centrality* has high importance to track down individual performances and compare them with other networks of different sizes and conditions.

Closeness Centrality

The *closeness centrality* $\mathbf{C'_C}$ of a node $\mathbf{p_k}$ measures the mean distance to reach any other node in the network. It is calculated by first summing all shortest path distances to all reachable actors and is then normalized by the number of nodes **n** in the network (Freeman, 1978).

$$C'_C(p_k) = \frac{n-1}{\sum_{v=1}^{n} d(p_i, p_k)}$$

$$with\ d(p_i, p_k) = shortest\ path\ distance\ between\ p_i\ and\ p_k\ in\ the\ graph$$

The higher the resulting value is, the more central is his position inside the network. An actor with a high *closeness* in our network indicates that he can reach people of different expertise

easily and fast. In many cases the *closeness centrality* is shown as the actual average distance without the normalization from the formula above. This makes the value easier to interpret, but also more difficult to compare. To have both advantages, we will work with both values.

Betweenness Centrality

Betweenness centrality C_B implies how many times a node $\mathbf{p_k}$ is a broker between two networks and is defined as the fraction of all shortest paths that pass node $\mathbf{p_k}$. The value is usually normalized by the formula $\frac{2}{(n-1)(n-2)}$ with **n** as number of nodes in the graph (Brandes, 2001; Freeman, 1977).

$$C_B(p_k) = \frac{\sum_{s,t \in V} \sigma(s,t|p_k)}{\sigma(s,t)}$$

$$where\ V\ is\ set\ of\ nodes; \sigma(s,t) is\ number\ of\ shortest\ (s,t)\ paths$$

$$and\ \sigma(s,t,p_k)\ is\ number\ of\ those\ paths\ passing\ through\ some\ node\ p_k\ other\ than\ s,t$$

The *betweenness* is a useful complement to *degree* and *closeness* centrality, as it shows another angle evaluating how often an actor connects certain subnetworks or persons with other components. In our reviewer network, the *betweenness* shows actors who might connect between departments and different knowledge areas, hence have a high importance even without having a high *degree* or *closeness*.

1.4.3 Used Tools and Libraries

To perform both Exploratory Data Analysis and Social network analysis on the code review data we used *Jupyter Notebooks* as primary development environment. The *Jupyter Notebook App* is an open source server-client program to edit and execute live code, equations, visualizations or text elements via a web browser. Taking advantage of these specifications, we could structure, execute, visualize and comment every step in one document to share and discuss it easily[1].

The programming language we chose to write and execute our code inside the *Jupyter notebook* is *python 3.6.* The widely usage, easy notation and powerful libraries made it possible to implement an easy readable code to perform all our desired methods.

We imported several libraries to undertake our operations. First, we used *pandas*[2] for data structure and data analysis. This open source library supports high-performance tools for e.g. deserialization, grouping or selecting attributes. The *statistics*, *numpy*[3] and *collections* libraries empower the execution of mathematical and statistical operations. To visualize the descriptive methods, we used *matplotlib*[4] a 2-D plotting library. Ultimately the *networkx*[5] library enables our Social network analysis with built in methods to visualize our graph as well as calculate metrics.

[1] http://jupyter.org/
[2] https://pandas.pydata.org/
[3] http://www.numpy.org/
[4] https://matplotlib.org/
[5] https://networkx.github.io/

1.5 Used Data Sources

1.5.1 General Characteristics of the Dataset

In our study we got the code review data of a multinational software development company. They supplied us with their complete recordings from 5 years of industrial code review practice. As the data is confidential all included personal data, such as names, are anonymized.

The dataset we used for our study is divided in three different files: *Reviews*, *review_entries* and *files*. To have a better understanding of the files and attributes, we shortly summarize the inspection process. We need to state out that every single code commit needs to be reviewed. This must be proven and is regulated by as central entity.

Before the reviewing starts an author submits a code patch. With the submission the corresponding information gets saved in the *file* table. Afterwards the file gets reviewed by assigned reviewers until the author got all electronic signatures including sometimes his own. When the file got all needed signatures, it changes the status from Unreviewed to Accepted in *reviews* with the current timestamp. In a special case scenario, it can already have skipped the previous steps when no review was needed. The exact numbers of the dataset are the following.

- *reviews*: 118166 entries with 10 attributes
- *review_entries*: 254212 entries with 3 attributes
- *files*: 98761 entries with 8 attributes

After merging all files to review_entries and applying all filters described in 2.4 we have a resulting table of 210697 entries with all 21 attributes. A more precise explanation of the different attributes and the ones we focused on will be topic in the following subchapter.

1.5.2 Attributes of the Dataset

Figure 2-1 and Table 2-1 give an overview about all attributes and the referencing tables. As told in the previous chapter variable first_name and last_name from class Person in Figure 2-1 got deleted and anonymized by identifiers.

To have a better understanding of our research results, we will explain all attributes we focused on during our research. All not mentioned attributes we either haven't considered because they were to company specific, for example branch or bundle. Or we didn't need them answering our research question. Some additional statistics and analysis will be part of 3. Elaboration chapter, e.g. *is_safety_relevant*.

The main attributes we focused on in our research were *reviewer*, *author*, *date*, *state*, and *result*. The person's IDs with counting workloads and interactions played the mayor role in our study represented by *reviewer* and *author*. The *date* of the reviews helped us to evaluate certain results and put it into a better context by having an exact timestamp. To organize our data, *state* and *result* were essential attributes to separate the data we wanted to focus on. Further statements about the quality of the dataset and attributes is topic of RQ1.1 and will be explained in the following chapter.

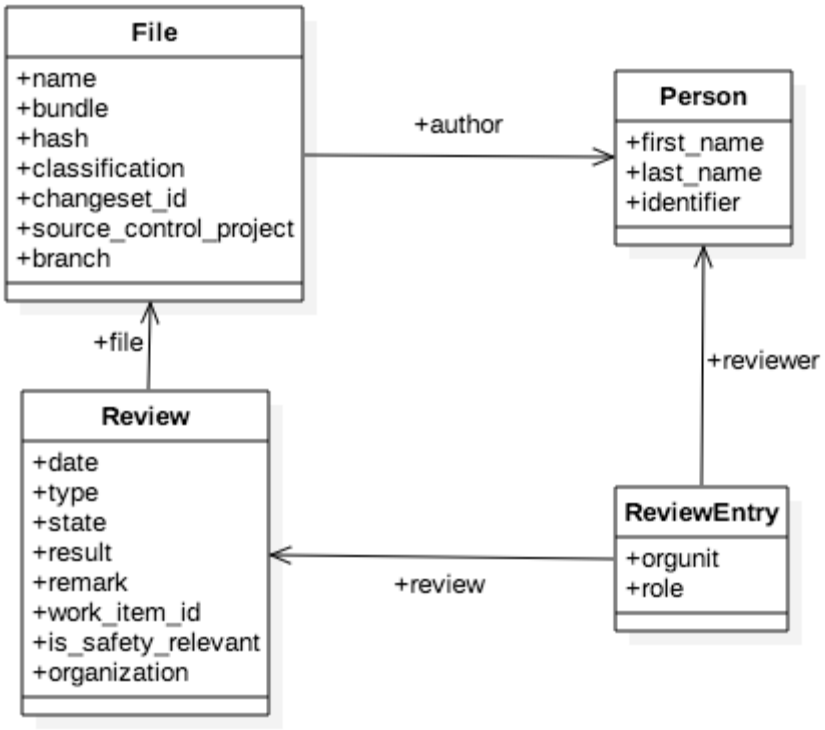

Figure 2-1: Files and attributes - class scheme

Attribute	Type	Values	From table	Study-relevant
Reviewer	float	anonymized person ID	reviews	Yes
role	string	reviewer role (e.g. *DEV*)	review_entries	No
orgunit	string	organization of reviewer	review_entries	No
date	timestamp	time & date at acceptance	reviews	Yes
file	integer	file ID	files, reviews	Yes
type	string	FourEyes or Walkthrough	reviews	No
state	string	Reviewed, Unreviewed, NoReviewNeeded	reviews	Yes
result	string	accepted or empty	reviews	Yes
remark	string	remark (e.g. *none*)	reviews	Yes
work_item_id	integer		files	No
is_safety_relevant	boolean	true or False	reviews	Yes
is_changed_in_range	boolean	true or False	reviews	No
organization	string	organization of review	reviews	No
hash	string	unique hash value	file	No
name	string	filename	file	No
bundle	string	bundle name	file	No
classification	string	file classification (e.g. *Source code*)	file	No
author	integer	anonymized person ID	file	Yes
changeset_id	integer	ID of changeset	file	No
source_control_project	string	name of module	file	No
branch	string	file branch	file	No

Table 2-1: Attributes with types and values

1.6 Research Results

1.6.1 Dataset Characteristics

Which general dataset characteristics determine code review quality?

In our first research question we want to clarify if we can find dataset characteristics, which expose code review quality. As a guideline we adhered to the six dimensions of dataset quality defined by Askham et al. These are *completeness*, *uniqueness*, *timeliness*, *validity*, *accuracy* and *consistency* (Askham et al., 2013). We applied every dimension to our dataset and in the end concluded which are characteristics to determine code review quality.

We started with checking the *completeness,* defined as the absence of empty values. The first simple step was checking for all missing values in our dataset. As a result, we could spot empty values for the two attributes *orgunit* and *organization*. In numbers, *orgunit* is missing **95.4%** and *organization* **96,6%** of its values. Consequently, due to this high number of percentage we can't consider these attributes for further research and they don't give visibility for the companies' reviewing process. To be more precise we also checked all strings for emptiness and numbers for zero values. Within that approach we found out that the *remark* attribute has **192** times an empty string. Even though it makes only a very small percentage (**0.17%**), it is an indicator that the remarks need further investigation.

Looking at the *uniqueness* we could verify that all entries in the *reviews* and *files* table are unique. Although *review_entries* has **5** entries that are duplicated, it just means that a person reviewed a file twice and shouldn't be concerned.

The *timeliness* dimension describes if the data reflects the reality for the point of time. In our dataset we are only able to see the date and time, when the review gets finally accepted. Accordingly, we have mayor limitations in observing the time and we can't compare our data with studies that investigated about time performing a review as a quality indicator (Izquierdo-Cortazar et al., 2017; Jiang, Adams, & German, 2013; Kononenko et al., 2016; P. C. Rigby et al., 2008).

As a fourth dimension a dataset meets *validity*, when it conforms to the syntax of its definition. Not considering the empty values all attributes are valid to format, type and range.

The *accuracy* of a dataset is the degree the values describe the actual event being captured. Looking at the dataset, we could already find out that the timeliness has missing accuracy. Other attributes that our dataset doesn't contain compared to other studies are change sizes (dos Santos & Nunes, 2017; Kononenko et al., 2016; Laitenberger, Leszak, Stoll, & Emam, 1999; P. C. Rigby & Bird, 2013) or a possibility to leave comments after every review entry (Bacchelli & Bird, 2013; Bosu et al., 2015; Rahman, Roy, & Kula, 2017). While elaborating the completeness we could already assume that the *remarks* aren't accurate. Further examination of the *remark* structure showed that the median length of a remark is only **5** characters long and is therefore both incomplete and inaccurate.

Last, we checked our data on *consistency*, the absence of differences. An interesting finding showed us that one file has occurred **1** time marked as *safety relevant* and in contrary **765** cases it hasn't. Since the characteristics of the file shouldn't change for different reviews, we consequently found inconsistency in the important attribute of safety relevance. Apart from that issue haven't found further inconsistent values in our dataset.

Based on our findings we want to determine which dimensions are indicators for code review quality. The inspection for missing values was a first indicator about the quality of the code reviews. It showed that two attributes are not usable for review retrospectives. Transferring the observation to practice, we can't ensure that missing fields in *organization* and *orgunit* are due to bad review practice, as it might also result from missing information from the system. In contrast the missing fields for *remarks* showed us that it was surely indicating to lax code review practice in general. Building up on this we could see that the missing accuracy of the data was also affected by code review practice. First the already mentioned inaccurate usage of the remarks. And second the missing timeliness concluding from insufficient recording of

valuable information. Especially when important factors like review-entry comments, review-entry timestamps or file size are not covered in the dataset, the company can't ensure adequate code review quality. As the *timeliness* in code reviews is always connected to recorded timestamps, we don't separate it as observation and rather include it in *accuracy* as a characteristic. Moreover, we observed that our dataset has inconsistencies. And accordingly, *consistency* is a useful characteristic to measure code review quality. The reason is that if a dataset contains inconsistencies it is highly probable it is cause by an imprecise review.

In comparison we conclude from our findings that neither *uniqueness* nor *validation* are good indicators for code review quality. Both are very important dimensions for data quality but aren't strongly connected to the inspections. Ensuring these are more related to the quality of the review tool than the actual reviewing process.

1.6.2 Reviewer Workload

What are indicators for effective workload distribution of code reviewers?

For our next research question, we wanted to examine how reviewer workload affects code review quality. Several studies have covered the topic about reviewing time and rate. The results conclude that code review rate highly influence their performance (Ferreira et al., 2010; Kemerer & Paulk, 2009). Kemerer & Paul additionally suggest that a code review should be prepared and not exceed **2** hours of inspection time. Bosu revealed in a study that developers at OSS systems spend in average **6** hours per week on reviews and only **12%** more than **10** hours (Bosu & Carver, 2013). A subsequent recent research compared OSS with Microsoft. The results were that the average workload for developers were **4** hours at OSS projects and **5** hours at Microsoft. That makes **10-12%** in an average 40-hour working week.

Building up on these results, we want to discuss whether our data indicates an effective workload distribution of the reviewers. The fact that every timestamp included several review entries, makes it unprecise to say when the review entry was performed exactly. Thus, we revised in a first place the general workload distribution over the years 2013-2016. Figure 2-2 and Figure 2-3 are boxplots showing the number of entries from all participated reviewers. Both illustrate the median, lower and upper quartile and Figure 2-3 additionally shows the outliers. In numbers, **50%** of all reviewers conducted between **18** and **284** reviews with an average of **61** entries in this **4** years. The widespread values and the closeness from median to lower quartile indicate that the workload is focused on a small set of reviewers with high workload. Especially by seeing the outliers from Figure 2-3 we can infer to high workload concentration. According to our results three reviewers recorded more than **5000** entries.

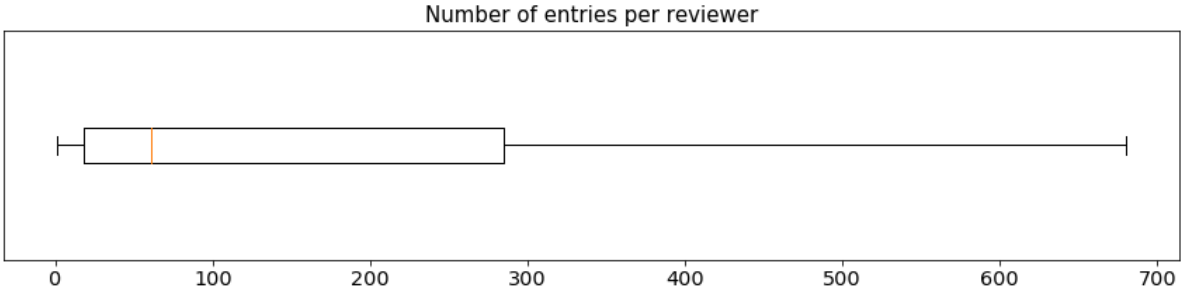

Figure 2-2: Number of review entries from 2013-2016

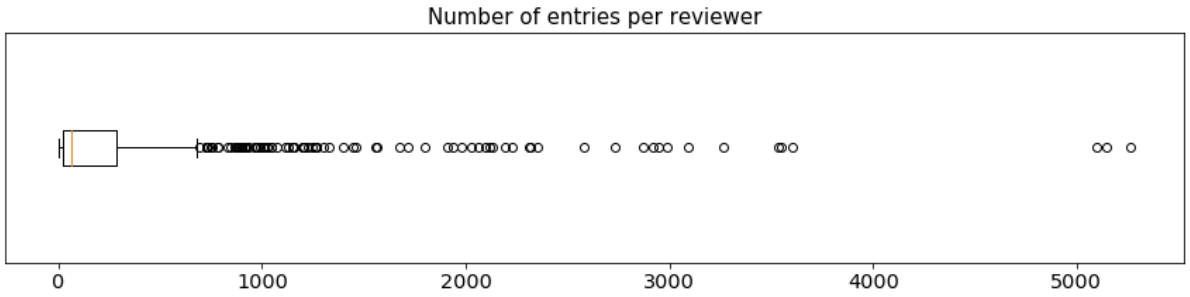

Figure 2-3: Number of review entries from 2013-2016 (with fliers)

In order to get our results into the important time or review rate context and compare it with former research we broke down the numbers in all **48** months of our recordings. We didn't choose a weekly overview, concerning the inaccurate timestamps. While Figure 2-4 describes how the median and quartiles change from month to month, Figure 2-5 shows the widespread outliers for certain reviewers. The median number shows that developers conduct around **6** reviews per month without a wide spread. Assuming that an average review lasts about **60** minutes, they would spend around **1.5** hours on reviews each week. Based on our assumptions and compared to the results from Bosu & Carver, where at Microsoft developers spend **5** hours on average, we can observe significant differences. More noticeable are the extreme outliers in Figure 2-5. Keeping in mind that the inaccuracy of the time might affect some values, we can still see obvious patterns that prove too high workloads for many reviewers. The numbers indicate for example that **217** times a reviewer conducted **160** reviews in just one month. In a from our company 35-hour week this would be a more than **1** review per hour rate. Our highest values show review rates from **7** up to **12** reviews per hour, which would be out of any realistic effort. We would need further investigation to find the causes for this widespread numbers. But it is highly probable that it is in the consequence of lax code review practice.

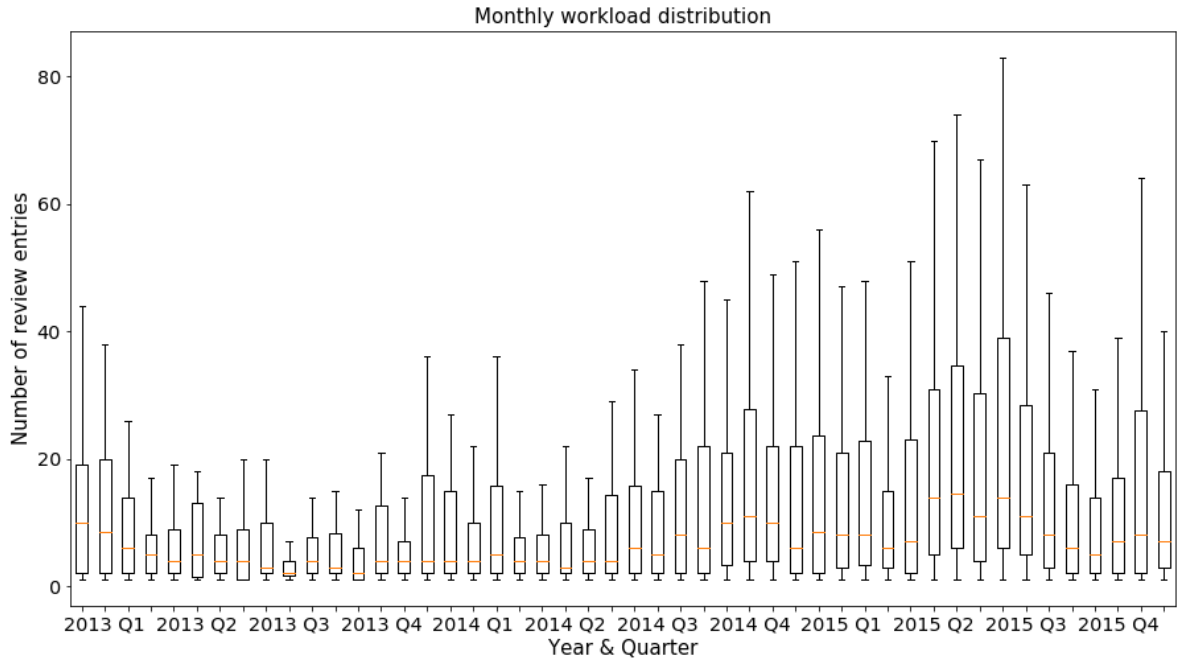

Figure 2-4: Monthly reviewer workload distribution

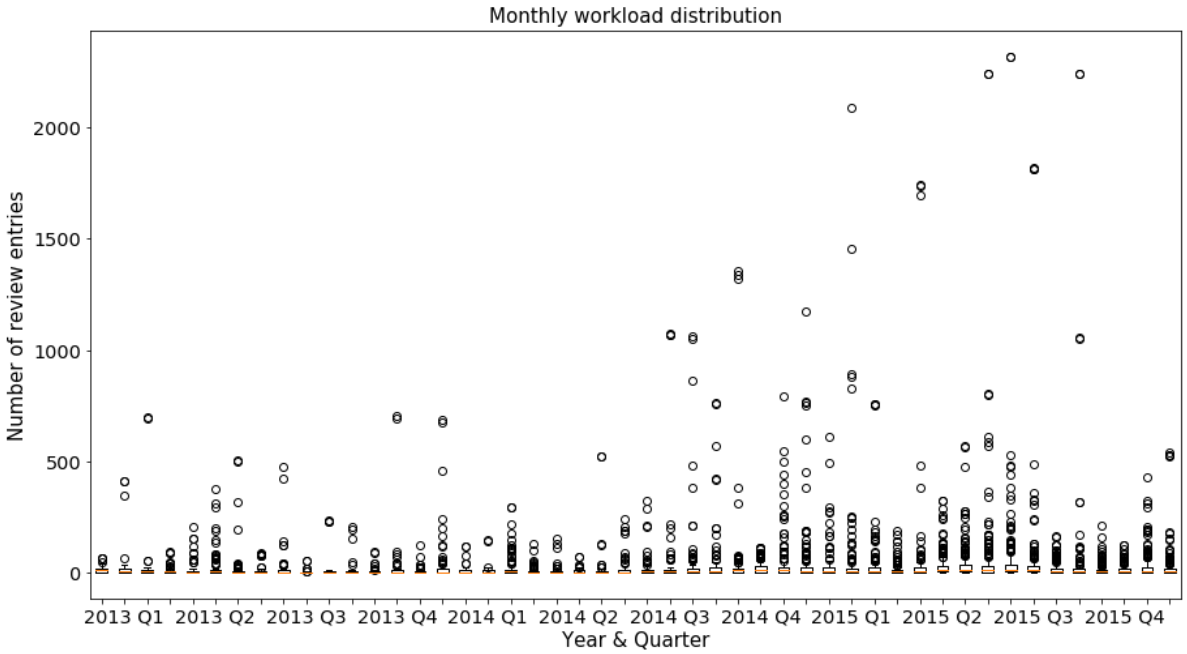

Figure 2-5: Monthly reviewer workload distribution (with fliers)

As a conclusion, we could find various indicators for ineffective workload distribution at our company's dataset. For the scrutiny of an effective workload we recommend using boxplots, as it shows various measurements to analyze your data. Looking at our results we suggest that the median value needs to be well centered between the two quartiles. This would mean the workload is equally distributed. A good median review rate usually depends on the company's structure but should be not more than around **15-20** reviews per month. Moreover, the distance of the two quartiles show the spread of the workload and is another indicator for code review quality. In an efficient process the quartiles should have a low distance which would

mean very good distributed reviewing. Last the outliers or fliers might signalize lax code review or even fraud, e.g. a person just signs code reviews to accelerate submissions. In any case a low number of closely spread outliers would indicate an effective workload distribution of code reviewers.

1.6.3 Reviewer selection

How can we measure efficient reviewer selection?

After having the close look on the reviewer workload, we now want to inspect the author reviewer relationship. The question we want to clarify is how to measure an efficient reviewer selection process. Two recent studies addressed the problem of how to select a reviewer. Both developed an automated tool and chose reviewers based on their experience and expertise (Balachandran, 2013; Ouni, Kula, & Inoue, 2017). Other studies support the hypothesis to rather chose experienced developers to guarantee higher code review quality (Kononenko et al., 2015; P. Rigby et al., 2012; Sauer et al., 2000). Taking these results into consideration we needed to find quantitative measurements if authors choose experienced reviewers for inspecting their files.

Since our data doesn't reveal information neither about fields of expertise nor experience, we needed to work on several assumptions. We assumed first that an author usually works on projects that need similar fields of expertise and second that a reviewer is building up expertise and experience every time reviewing a code. Our resulting hypothesis is the following. If a reviewer has already been revised a file of an author, the author should tend to select this reviewer again, as he already has familiarity and expertise with his code. Accordingly, we want to examine the distribution of the reviewer count for each author depending on the general number of reviews the author requested. Based on our hypothesis an author should have a variation but small pool of experts revising his code.

To measure this, we checked the proportion between number of requested reviews and number of different reviewers an author had. We used a scatterplot to evaluate the distribution and calculated a regression line for a general observation (Figure 2-6). It is important to mention that proportion in our figure is 1:10 between *number of different* reviewer*s* and *number of requested reviews*. Table 2-2 shows in numbers how the relationship is distributed.

To check our hypothesis, we first want to examine the general distribution of the reviewer number per author, represented by the linear regression. It denotes a line with the minimal distance to all data points, hence shows how the *number of different* reviewer*s* grows in relation to the *number of requested reviews*. The resulting lines' slope of 1,23% indicates a small growth and we interpreted the result as an efficient and good ratio for author-reviewer distribution. Our results can be used for future studies in this subject. Another observation is that with increasing number of requested reviews the number of different reviewers is close and mostly below the regression line. Consequently, we can suggest that the very active authors stick to our quality premise and choose only experienced reviewers. Nevertheless, it is to consider that several examples have a concerningly very low number of different reviewers while having requested many inspections. We think these very low numbers of different authors can result to a very unilateral perspective and be negatively for your inspection quality and as well might be an indicator for possible fraud. Generally, the dispersion should be low in a perfect setting. But looking at the numbers from Table 2-2 we can see that variation in our review data is high. First, the high difference of the arithmetic-means and medians demonstrate that we have high outliers, since the median is less affected the extreme values. Second the standard deviation depicts the dispersion from the mean. According to our value of **23.9** we can see that the values are widespread and especially influenced by the outliers. Our last observa-

tion is focusing on the density of the distribution. We can see that most authors seem to only request very few reviews, since the density is the highest in the left bottom corner.

Summing up our observations the companies' authors generally tends to choose reviewers on experience following our premise. The relation is generally good but wide spread. Again, the extreme outliers need to be further investigated.

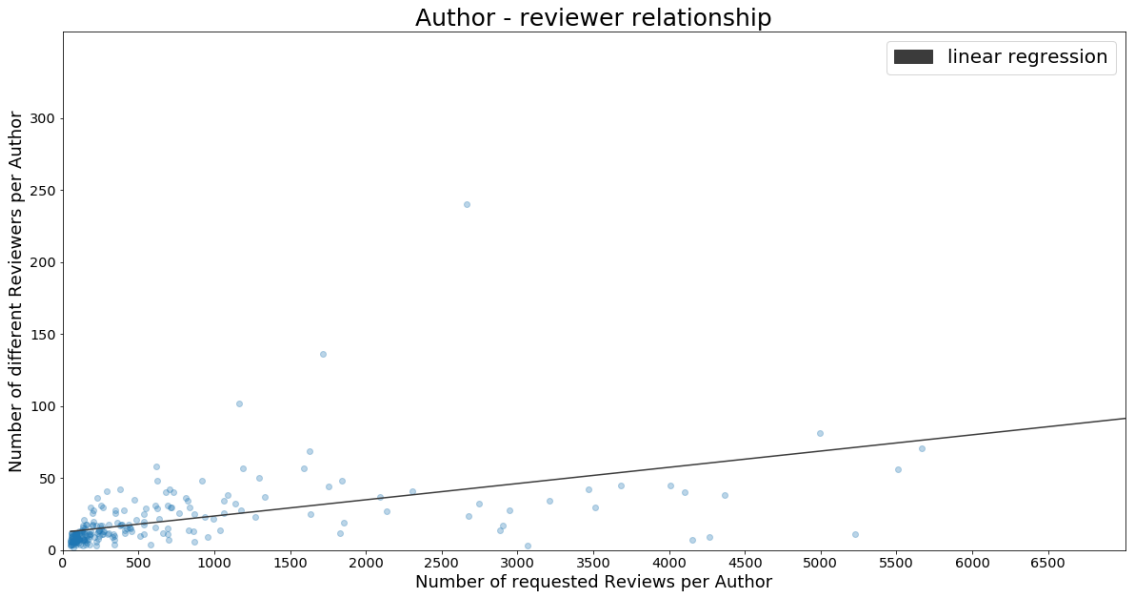

Figure 2-6: Author - reviewer relationship

	Different reviewer per author	Requested reviews per author
Median	8	61
Arithmetic mean	14.4	507.7
Standard deviation	23.9	1 356.1
Outlier	240	142 206
	Other information	
General distribution	682 r*eviewers* - 415 *authors*	
Linear regression slope	1,226 %	

Table 2-2: Author - reviewer relationship in numbers

In a second a approach motivated by the density observation from Figure 2-6, we wanted to look from the angle of the reviewer. Rigby et al. defined within their study on the Apache server that an effective reviewing environment consists of a small group of self selected experts (P. C. Rigby et al., 2008). To investiagte this we sorted all **682** reviewers by the number of reviews they conducted in descending order. Afterwards we separated them into **14** groups from the **50** most active to the **50** least active reviewers. Figure 2-7 shows our result represented by a *pareto chart*. It first shows by barchart the totalized number of the group conducted reviews and by linechart the percentage of share from all reviewers. We can see that the first **50** have a share of over **50%** of all reviews and the first **150** over **80%** of all reviews. Accordingly our data shows that **80%** of all work is done by only **20%** of the reviewers. Therefore this is by the definition of having a small group of expertised and experienced reviewers an efficient reviewing process.

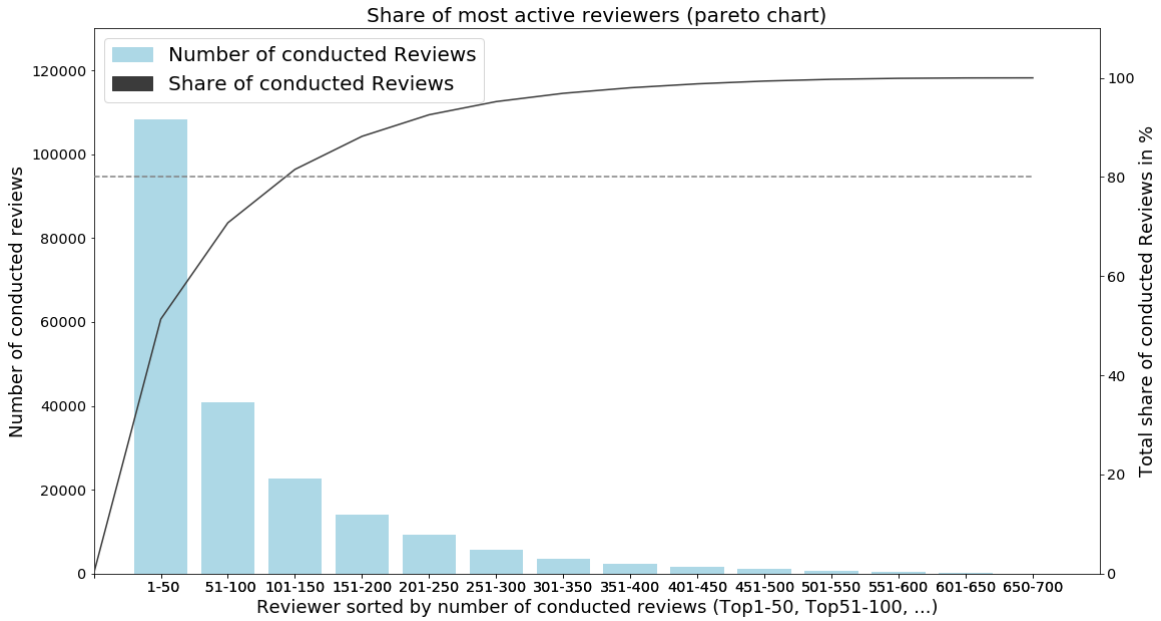

Figure 2- 7: Share of most active reviewers (pareto chart)

Comparing our results with the reviewer workload (RQ2), the criterious seem contrary. First we critized the intense workload for various reviewers and next we looked for a small group of experts conducting reviews. Hence it is important to explain that we don't look at the time axis within our observations for the reviewer selection. Consequently we separated this two approaches. Nevertheless, the two criterious are not contrary as we can both have a small pool of experts having a managable monthly review rate.

In conclusion we can measure efficient reviewer selection by putting number of requested reviews and number of different reviewers per author into relation. The resulting numbers of different reviewers should slowly rise by increasing requested reviews and be low spread along an regression line with a small slope. Our pareto chart indicates the share of how many reviews gets conducted by the most active reviewers. According to former research a small group of experts conducted more efficient reviews.

1.6.4 Social Network Metrics

Which social network metrics describe code review quality?

In this section we analyze our dataset using a social network analysis. In our network we use the metrics described in chapter Research Approachto analyze which describe code review quality. First, we have a look at the resulting network from Figure 2-8. It shows all authors and reviewers connected by weighted undirected edges containing all data we have. The graph is hardly interpretable, because of its' high density and big number of nodes and edges. But the formation shows that many authors are strongly connected requesting and reviewing code while many reviewers in contrast just have just one-time connections mostly reviewing code.

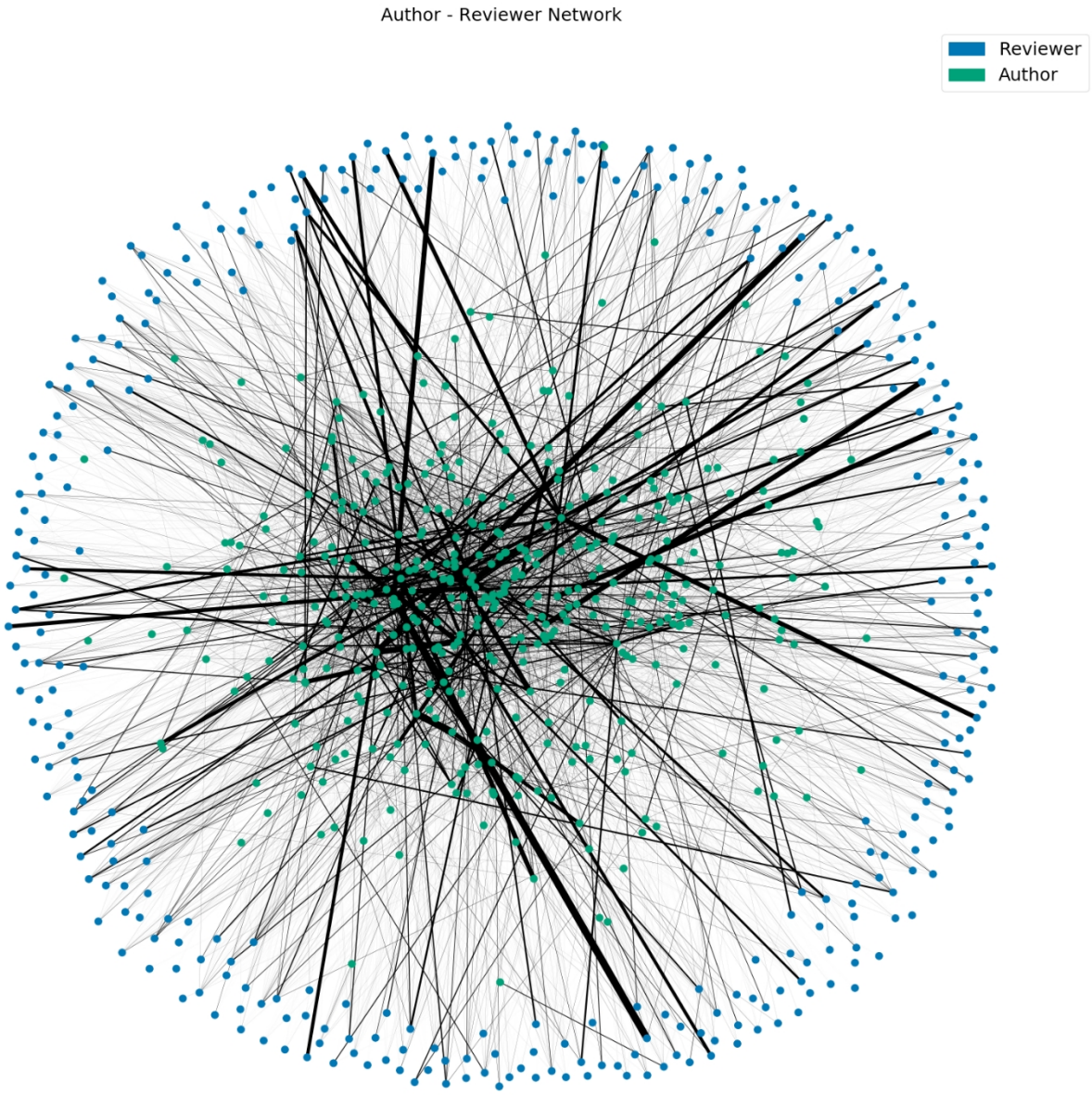

Figure 2-8: Social network (reviewer - author)

To compare our results, we will use the study from Meneely et al. as he revealed several metrics we also inspected. Their research focus on failure prediction using a Social network analysis on developers from a network product (Meneely et al., 2008). Still we will need to interpret several numbers without references, as current research is missing comparable measurements in this field. Our numbers will be useful benchmarks for further studies on code review networks.

For the calculation of our metrics we differentiated between two types of graphs, *undirected* and *directed*. The *directed* graph is meant to distinguish between authors and reviewer importance. But looking at centrality and some other measurements we interpreted every review request as bidirectional interaction. The reason is that in a *directed* network an actor who is only reviewer wouldn't have influence at all, since he has no outgoing edges. All metrics we measured within the *directed* graph we marked with "**directed***"* in the Tables below.

Table 2-3 shows general characteristics from the network of Figure 2-8. The companies' author-reviewer Network consists of in total **777** different actors with **5477** edges. The density, or proportion of all edges to all possible edges, is **1,81%**. Compared to the dataset from Me-

neely et al., which consists of **161** developers, our network is comparable big. The fact that our density is around 1-2%, indicates that we have a widespread network with a low level of exchange between the actors.

Table 2-4 shows centrality measures applied on our network. In contrast to the results from Meneely, where the training data is from a development of one network product, we could expect lower average centrality values by the higher diversity of our company's products. Usually most metrics are normalized to compare networks easier. Since our paper from Meneely hasn't normalized its values, we appended them in brackets if necessary.

To analyze the general code review performance, we first want to focus on the average values. The average *degree* is **14** for our author-reviewer network. In comparison the developer graph from Meneely has an average *degree* of **19**, even while having fewer nodes. In contrast the average not-normalized *closeness* of our network is comparably strong with an average distance of **2.90** in contrast to the smaller network from Meneely with a value of **2.77**. Looking at our last centrality measurement, the *betweenness* centrality, we don't have comparable data. But we can see that the *betweenness* centrality is with an average of **736** comparably high considering all the outlying actors from Figure 2-8 with very low *betweenness*.

As a second step we compare our outlying values with the average values to evaluate single performances. When having a closer look at the individuals with the best values from our table, we found out that all metrics beside out-degree is dominated by the same person. With its extremely high out degree it controls around one third of the whole network with direct connections. As well, the person is very fast connected to all other actors and connects many sub networks and single persons to the main component. Especially by comparing the average with these values, we can see an impressive influence.

Characteristics	Value
Number of nodes	777
Number of edges	5477
Density	1.81%

Table 2-3: Social network general characteristics

Metric	Average value	Best value
Degree	14.09	265
In-degree (directed)	7.38	55
Out-degree (directed)	7.38	239
Degree centrality	1.90%	34,14%
In-degree centrality (directed)	0,95%	7,09%
Out-degree centrality (directed)	0,95%	30,79%
Closeness centrality	0.35 (2.90)	0.56 (1.79)
Betweenness centrality	0.0025 (736)	0.2288 (68792)
Eigenvector centrality	0.0223	0.0283
Pagerank	0.0012	0.0214

Table 2-4: Social network metrics

We concluded that a social network analysis enables to identify important actors inside the companies' reviewer network. By degree it is easily possible to spot important authors and re-

viewers and the other centrality measures indicate hubs and generally important actors. To determine the review quality, we used average values and compared them with another study. Even though it was a good indication about the quality of the structure, we would need more comparable data to make more precise statements.

1.7 Discussion

Discussing our research result we want to have a first look at our mayor limitations. As described in chapter Dataset Characteristicswe found several limitations throughout inaccuracy and missing fields in our dataset. This made several approaches impossible or led to restrictions. Another limitation in our study was the missing of comparable data for a precise evaluation. Even though the strong research on the field of code reviews, relevant studies doesn't reveal equivalent precise numbers in order to validate our results.

These limitations decrease the general internal validity of our study. The missing time accuracy leads to biasing our results for the reviewer workload. Moreover, the missing information about patch sizes minimizes the precision in the workload observations. This makes it as well more difficult for external transferability. Nevertheless, the fact that the code review data is regulated and complete raises the internal validity, as we can ensure that all code review activities are documented. This fact makes it as well externally transferable to industrial practice that can confidentially ensure a fully documented inspecting process. Our results within the social network analysis can be due to its normalized values, easily be transferred to further reviewer author dynamics. The general methodology about analyzing the dataset, reviewer selection and reviewer workload is applicable for further studies in this field. As most projects document date as well as author and reviewer with IDs, it is external valid when considering the possible differences in patch sizes throughout the reviews.

1.8 Conclusion

In this paper, we have proven that the current code review standards in industrial practice needs improvements and more precise standards. On the investigation of a multinational company conducting governmental controlled code review process, we could find several indicators to measure code review quality. We examined dataset characteristics, reviewer workloads, reviewer selection and an author-reviewer network in an exploratory approach. Our results have high significance as we could find outliers in all categories we inspected. Based on this findings companies can compare their code review practice using our approaches and results. As an implication companies conducting code review, should be aware that current practice still needs careful observation and reconsideration.

1.9 Future Work

While we had a quantitative approach for our research paper, as a part of future work we plan a qualitative method to investigate further on the dataset we have. It would be interesting to interview several persons who are involved in the center of the reviewing network to investigate further on reasons for our outlying values. Another approach for future research is to compare our results to a similar sized open source project using the same methods. This would empower us to have a face to face evaluation of our findings.

1.10 Acknowledgments

Finishing this study, I want to acknowledge all responsible persons of the software company who shared their confidential data with us and gave us a unique insight in recent code review standards.

I want to acknowledge my supervisor, Michael Dorner, who contacted the company, processed all data and supported me with feedback and ideas in the quickest and most supportive possible manner.

Last, I want to acknowledge all the personnel of the Open-Source-Software chair from Prof. Dr. Dirk Riehle, who established a well-organized thesis process with helpful guidelines and high standards.

2 Elaboration chapter

2.1 Remarks

In a relevant study at Microsoft, Bosu et al. measured code review quality building up on interviewing developers on code review comment usefulness. Regarding to their outcome, comments perceived as useful are identification of functional issues, validation issues, API suggestions, software design to follow or comments about team coding conventions. As a result, they created a classifier evaluating the usefulness of code review comments (Bosu et al., 2015).

Based on this study, we checked if we can find similar observations inside the code review *remarks*. In chapter Dataset Characteristicswe already could shortly prove that the *remarks* are very unprecise due to its shortness and incompleteness. In this chapter we want to have a closer look at the length and content of the *remarks*.

Figure 3-1 shows a boxplot representing median, upper- and lower quartile of the *remarks* in character length. It shows that 50% of the *remarks* are between **4** and **18** characters long with a median of **5**. The antennas show that comments with **0** strings and comments upon the length of **35** appear in the dataset as well. We can conclude that within that range of character length the reviewers hardly give useful comments, such as defined from Bosu et al. above.

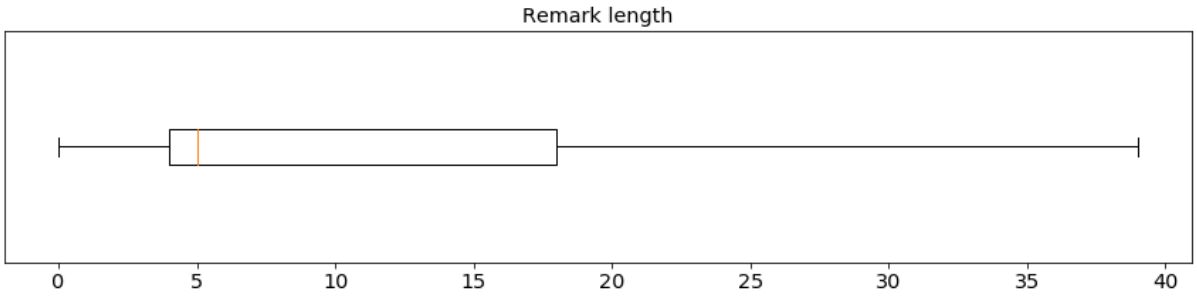

Figure 3-1: Remark character length

To complement our findings, we want to inspect the content of the *remarks*. With only looking at the length of the comments, we can't ensure if the company isn't using special codes or abbreviations. Therefore, we created a wordcloud, showed below in Figure 3-2. This graphic groups all terms or phrases and represents the number of appearances as size of the word inside the graphic. The resulting figure states out that we can't find evidence for any form of codes or abbreviation that justify the average short length of the comments. Most of the *remarks* don't have any content and use either default terms, very few additional information, small hints or some state out the review type.

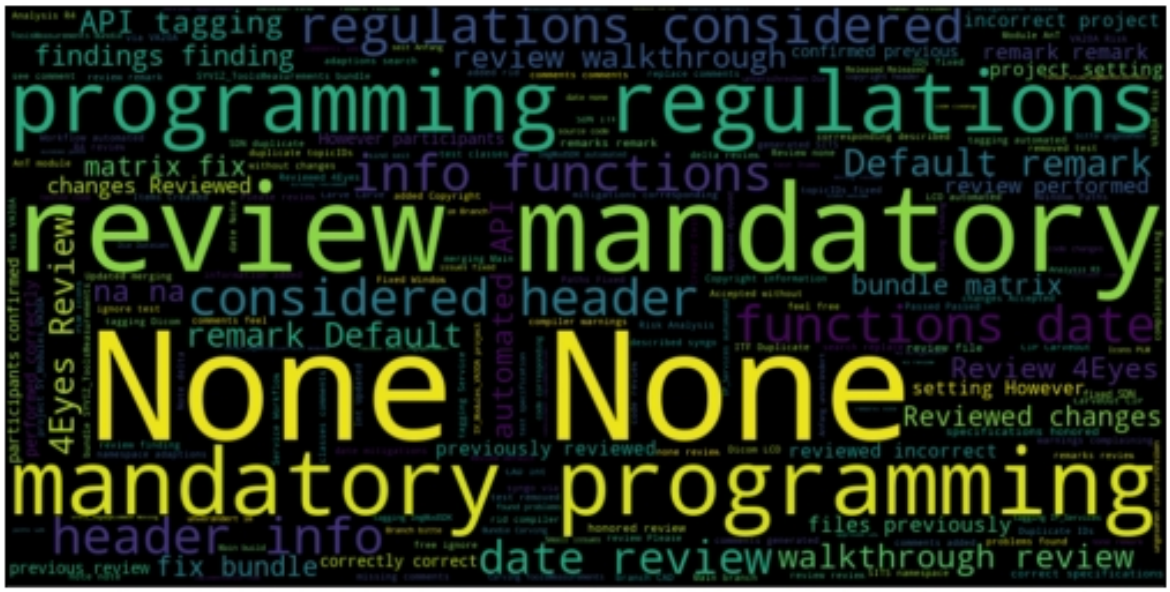

Figure 3-2: Remarks wordcloud

The code review remarks from our company's dataset is a hardly used medium for actual comments on their inspections. Most of comments are short and without content. We can conclude that the company needs to define a useful commentary function and give the reviewers incentives for using it. In the meantime, their process lacks in effective code reviewing.

2.2 Safety Relevant Files

The data we got from our company has several company specific attributes we haven't considered due to difficult transferability. Still we want to take advantage of one special token, the safety relevant of a file.

Within our exploratory research we checked if the safety relevance has influence on the company's code review process. We expected more review entries for files that have set their token *is_safety_relevant* to *true*, as safety relevant files have a lower error tolerance and need more precise inspections. Hence, we wanted to check if safety relevant files have a higher number of entries because of probable higher standards and less tolerance. The result was that the arithmetic mean for entry-number per file is **2.63** for safety relevant and **2.66** for not safety relevant files with a median of **2** in both cases. Accordingly, the findings are contrary to what we assumed before and safety relevant files have an average lower number of entries. Even though the difference is low and with **0.28%** we have a low share of safety relevant files, the results are concerning for the companies' code review process. We can recommend that the company defines its standards for safety relevant files or have a more accurate definition of this attribute in general.

2.3 Weekday and Working Time Distribution

Another additional approach was to examine the distribution of time and workdays. Therefore we checked the timestamps a review recievews when it gets accepted. As discussed in chapter Dataset Characteristics the date has missing accuracy, since review entries don't have any form of time documentary. Still we checked if the review acceptance timestamps have patterns that indicate to issues in their review practice.

Figure 3-3 shows the distribution of the weekdays when the code review got commited. As we can observe, the distribtuion is normal and with low differences between the usual working days, Monday to Friday. As expected at Saturday and Sunday the number is very low due to the fact that these two days are out of the working hours for our company. Concludingly the reviews are well distributed within the working week and don't have any atypical behaviour.

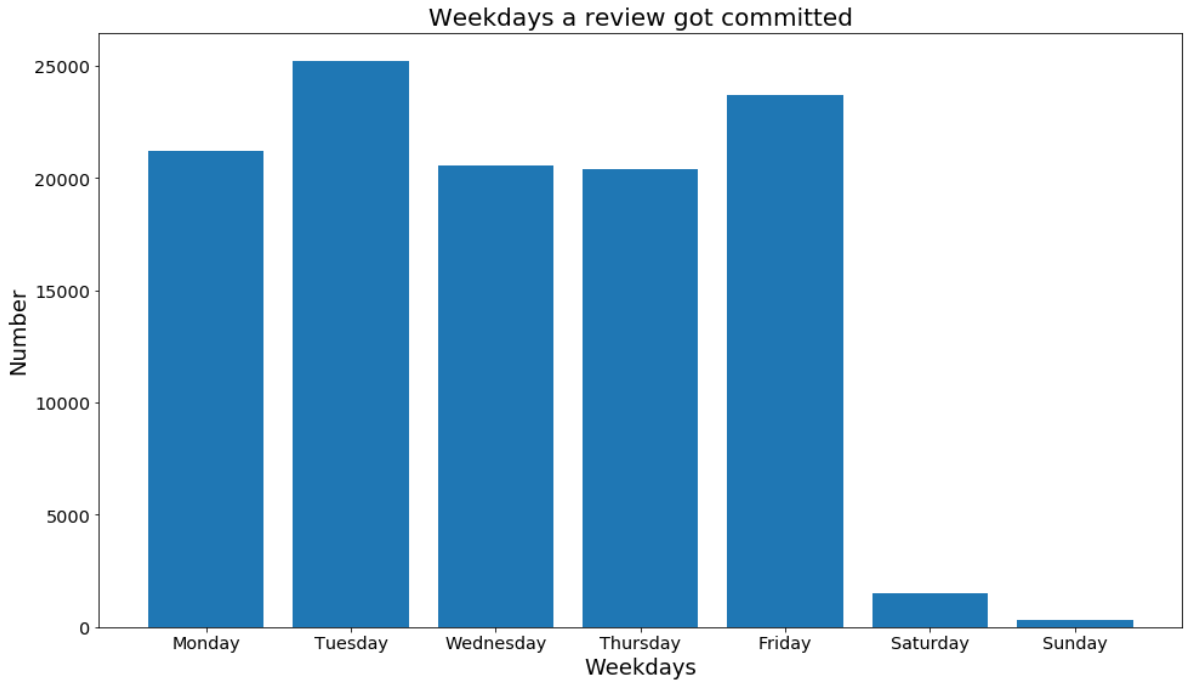

Figure 3-3: Review commit workday distribution

The distribution of time is illustrated by Figure 3-4. According to the usual office hours, we were expecting the main activity from 8am to 5pm with a slow rise before and a decrease afterwards. Looking at our findings, we can see that the activiy rises as expected through the morning with a small valley during the lunch time from 12am to 1pm and then slowly decrease by the finishing of the day. Hence we have can observe the anticipated results during the working hours. On the contrary, the very first hour of the day has an high peak at midnight from 12pm to 1am. Since we can't expect people working at midnight and especially as it is the only peak during the night, the result needed further investagtion. Within a closer examination of the exact times when the reviews got commited, we could find out that almost all values (**3824)** in this timeframe are submitted at exactly **00:00:00** o'clock. Thus it is evident, that the findings are due to either an system error or a manual manipulation. We can suggest that these manipulated timestamps are the result of a late data import, which was mentioned from the software company. However, as we don't have further information about reasons for that import, it would be an interesting question for future research.

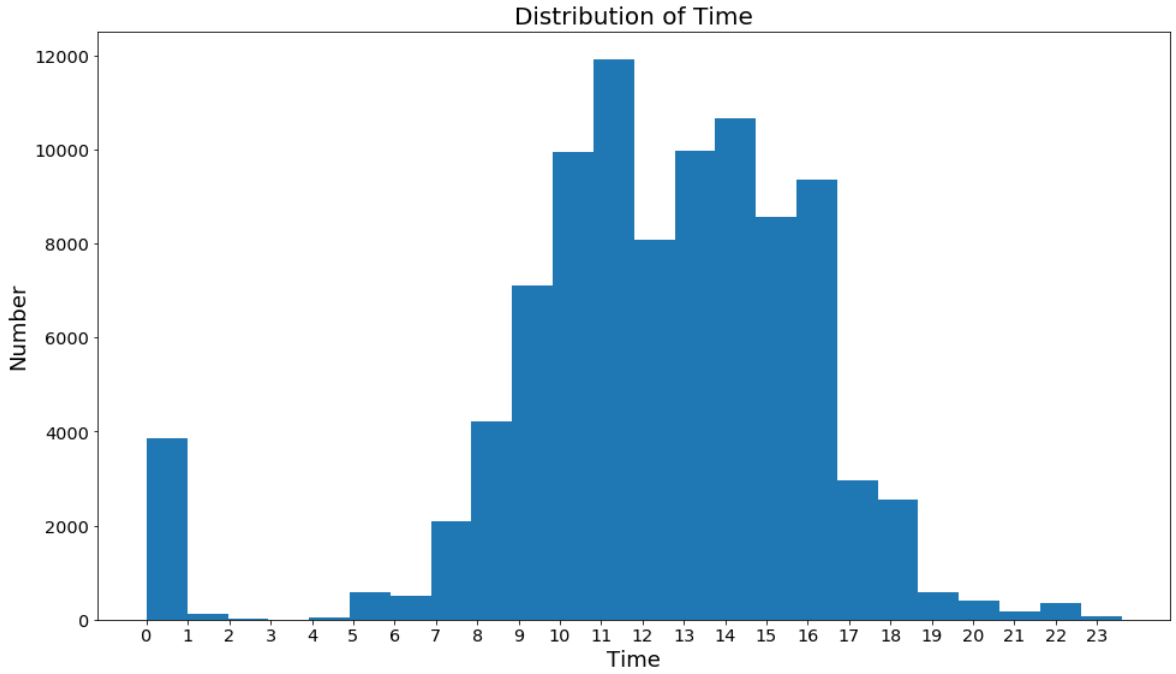

Figure 3-4: Review commit daytime distribution

2.4 Social Network Core Analyzation

In our findings from chapter Reviewer selectionwe found out that 80% of the work is done by only 20% of our reviewers. We could verify that with our social metrics and graphs, which consist of reviewers and authors with very strong connections inside the network. Consequently, we extracted the main core of our network and analyzed the corresponding metrics. To get our network center, we used the k-core algorithm defined by Batagelj and Zaversnik. This algorithm is implemented in *networkx*[6] and determines the core as described in the following chapter.

2.4.1 K-core Algorithm

The k-core algorithm is based on the idea of creating a subgraph of all nodes that have a degree higher than k. Its implemented by recursively deleting all vertices and line incident with them of degree less than k from a given graph. The logic of the algorithm is simply illustrated in Figure 3-5. As a first step we calculate the degree of all vertices and ordered them increasingly according to its degree. Afterwards we created the k-core by starting from the vertices with the smallest degree and deleting all that have a neighbor with a higher degree. While deleting we updated all connected nodes by lowering the degree and resorting all vertices after. The resulting list of nodes is an approximated core of our network and consist of updated degrees only considering the vertices inside this subgraph (Batagelj & Zaversnik, 2003).

[6] https://networkx.github.io/documentation/networkx-1.7/reference/generated/networkx.algorithms.core.k_core.html

```
1.1 compute the degrees of vertices;
1.2 order the set of vertices V in increasing order of their degrees;
2 for each v ∈ V in the order do begin
    2.1 core[v] := degree[v];
    2.2 for each u ∈ Neighbours(v) do
        2.2.1 if degree[u] > degree[v] then begin
            2 2.2.1.1 degree[u] := degree[u] – 1;
            2.2.1.2 reorder V accordingly
    end
end;
```

Figure 3-5: K-core algorithm (Batagelj & Zaversnik, 2003)

2.4.2 Core Analyzation

After applying the k-core algorithm to our network we have a resulting graph of the **35** nodes with the highest degree showed in Figure 3-6. The graph is helpful to see how the main reviewers and authors are connected. In this network all actors are both reviewer and authors and are represented by an undirected and unweighted graph.

By having a first look at the graph, we can already see a dense connection between all actors. There is no abnormality to observe and every node is well linked to all others. This impression is supported by examining the network metrics from Table 3-1 and Table 3-2. The network has **56.4%** of all possible edges and connects all vertices fast and easily with an average *closeness* of **1.4** steps to all other nodes. Another observation is that in contrast to our full network, inside the core a different person has the highest values, such as a *degree centrality* with **94.11%**. This give an interesting insight, that we can differentiate between core importance and total network importance.

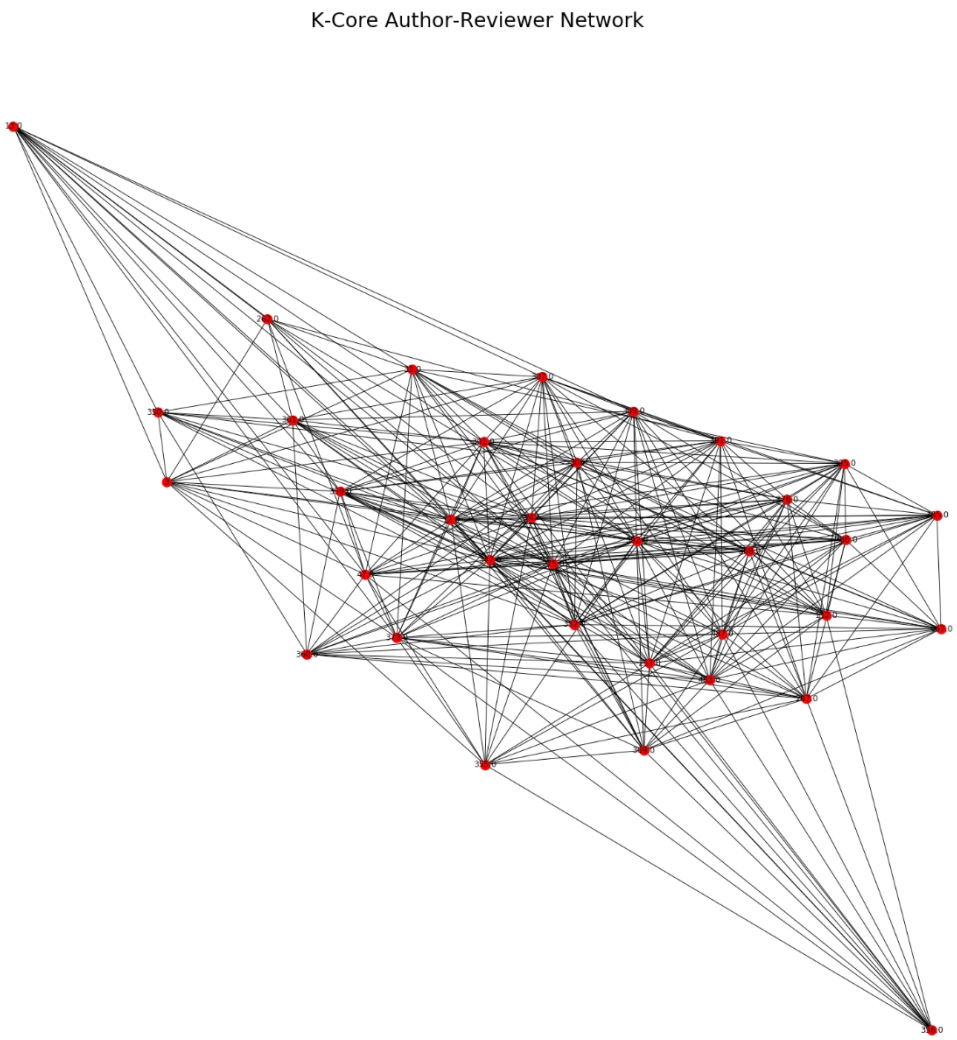

Figure 3-6: K-core author-reviewer network

.

Characteristics	Value
Number of nodes	35
Number of edges	336
Density	56.4%

Table 3-1: K-core - social network general characteristics

Metric	Average value	Best value
Degree	19.20	32
Degree centrality	56.4%	94.11%
Closeness centrality	0.70 (1.44)	0.94 (1.06)
Betweenness centrality	0.0132 (7.4)	0.0481 (27.0)
Eigenvector centrality	0.1652	0.2578
Pagerank	0.0286	0.0452

Table 3-2: K-core - social network metrics

As a conclusion the core network gives an interesting angle to our network. We can see that the author – reviewer main core is well connected and shares knowledge and work actively. As well we found out that inside our core, the individual importance is different distributed. The core implementation shows us a generally good overview about the most important actors in the network based on their connectivity to other people.

Appendix A Dataset Statistics and Metrics - Summary

Statistic/ Metric	Result
Number of entries	254 212
Number of reviewers	682
Number of authors	415
Entries per reviewer – median	61
Entries per reviewer – lower quartile	18
Entries per reviewer – upper quartile	284
Monthly entries per reviewer – median	6
Monthly entries per reviewer – lower quartile	2
Monthly entries per reviewer – upper quartile	22
Number of requested reviews vs. number of different reviewers – linear regression slope	1.29%
Different reviewers per author – median	8
Different reviewers per author – arithmetic mean	14.4
Different reviewers per author – standard deviation	23.9
Requested reviews per author – median	61
Requested reviews per author – arithmetic mean	507.7
Requested reviews per author – standard deviation	1 356.1
Social network – number of nodes	777
Social network – number of edges	5 477
Social network – density	1.81%
Social network – average degree	14.09
Social network – average in-/ out-degree	7.38
Social network – average degree centrality	1.90%
Social network – average in-/ out-degree centrality	0,95%
Social network – average closeness centrality	0.35 (2.90)
Social network – average betweenness centrality	0.0025 (736)
Social network – average eigenvector centrality	0.223
Social network – average pagerank	0.0012

References

Askham, N., Cook, D., Doyle, M., Fereday, H., Gibson, M., Landbeck, U., … Schwarzenbach, J. (2013). *The Six Primary Dimensions for Data Quality Assessment*. *Group, DAMA UK Working*. Retrieved from https://www.dqglobal.com/wp-content/uploads/2013/11/DAMA-UK-DQ-Dimensions-White-Paper-R37.pdf

Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. *Proceedings - International Conference on Software Engineering*, 712–721. https://doi.org/10.1109/ICSE.2013.6606617

Balachandran, V. (2013). Reducing Human Effort and Improving Quality in Peer Code Reviews using Automatic Static Analysis and Reviewer Recommendation, 931–940.

Batagelj, V., & Zaversnik, M. (2003). An O(m) Algorithm for Cores Decomposition of Networks, 1–9. Retrieved from http://arxiv.org/abs/cs/0310049

Bavota, G., & Russo, B. (2015). Four eyes are better than two: On the impact of code reviews on software quality. *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings*, (i), 81–90. https://doi.org/10.1109/ICSM.2015.7332454

Bosu, A., & Carver, J. C. (2013). Impact of peer code review on peer impression formation: A survey. *International Symposium on Empirical Software Engineering and Measurement*, 133–142. https://doi.org/10.1109/ESEM.2013.23

Bosu, A., & Carver, J. C. (2014). Impact of developer reputation on code review outcomes in OSS projects. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, 1–10. https://doi.org/10.1145/2652524.2652544

Bosu, A., Carver, J. C., Bird, C., Orbeck, J., & Chockley, C. (2017). Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft. *IEEE Transactions on Software Engineering*, *43*(1), 56–75. https://doi.org/10.1109/TSE.2016.2576451

Bosu, A., Greiler, M., & Bird, C. (2015). Characteristics of useful code reviews: An empirical study at Microsoft. *IEEE International Working Conference on Mining Software Repositories*, *2015–Augus*, 146–156. https://doi.org/10.1109/MSR.2015.21

Brandes, U. (2001). A faster algorithm for betweenness centrality*. *The Journal of Mathematical Sociology*, *25*(2), 163–177. https://doi.org/10.1080/0022250X.2001.9990249

Cohen, J., & Brown, E. (2012). Best Kept Secrets of Peer Code Review Authors, 159.

Czerwonka, J., Greiler, M., & Tilford, J. (2015). Code Reviews Do Not Find Bugs. How the Current Code Review Best Practice Slows Us Down. *Proceedings - International Conference on Software Engineering*, *2*, 27–28. https://doi.org/10.1109/ICSE.2015.131

dos Santos, E. W., & Nunes, I. (2017). Investigating the Effectiveness of Peer Code Review in Distributed Software Development. *Proceedings of the 31st Brazilian Symposium on Software Engineering - SBES'17*, 84–93. https://doi.org/10.1145/3131151.3131161

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*, *15*(3), 182–211. https://doi.org/10.1147/sj.153.0182

Ferreira, A. L., MacHado, R. J., Silva, J. G., Batista, R. F., Costa, L., & Paulk, M. C. (2010). An apporach to improving software inspections performance. *IEEE International Conference on Software Maintenance, ICSM*. https://doi.org/10.1109/ICSM.2010.5609700

Freeman, L. C. (1977). A Set of Measures of Centrality Based on Betweenness. *Sociometry*. https://doi.org/10.2307/3033543

Freeman, L. C. (1978). Centrality in Social Networks. *Social Networks*, *1*(1968), 215–239. https://doi.org/10.1016/0378-8733(78)90021-7

Hamasaki, K., Kula, R. G., Yoshida, N., Erika, C. C. A., Fujiwara, K., & Iida, H. (2013). Who does what during a Code Review ? An extraction of an OSS Peer Review Repository. *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR' 13)*, 49–52. https://doi.org/10.1109/MSR.2013.6624003

Huang, S.-K., & Liu, K. (2005). Mining version histories to verify the learning process of Legitimate Peripheral Participants. *ACM SIGSOFT Software Engineering Notes*, *30*(4), 1. https://doi.org/10.1145/1082983.1083158

Izquierdo-Cortazar, D., Sekitoleko, N., Gonzalez-Barahona, J. M., & Kurth, L. (2017). Using Metrics to Track Code Review Performance. *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering - EASE'17*, 214–223. https://doi.org/10.1145/3084226.3084247

Jiang, Y., Adams, B., & German, D. M. (2013). Will my patch make it? And how fast?: Case study on the linux kernel. *IEEE International Working Conference on Mining Software Repositories*, (Section II), 101–110. https://doi.org/10.1109/MSR.2013.6624016

Kemerer, C. F., & Paulk, M. C. (2009). The impact of design and code reviews on software quality: An empirical study based on PSP data. *IEEE Transactions on Software Engineering*, *35*(4), 534–550. https://doi.org/10.1109/TSE.2009.27

Kononenko, O., Baysal, O., & Godfrey, M. W. (2016). Code review quality. *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, 1028–1038. https://doi.org/10.1145/2884781.2884840

Kononenko, O., Baysal, O., Guerrouj, L., Cao, Y., & Godfrey, M. W. (2015). Investigating code review quality: Do people and participation matter? *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings*, 111–120. https://doi.org/10.1109/ICSM.2015.7332457

Laitenberger, O., Leszak, M., Stoll, D., & Emam, K. El. (1999). Quantitative Modeling of Software Reviews in an Industrial Setting. *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, 312. https://doi.org/10.1109/METRIC.1999.809752

Lopez-Fernandez, L. (2004). Applying social network analysis to the information in CVS repositories. *International Workshop on Mining Software Repositories MSR 2004 W17S Workshop 26th International Conference on Software Engineering*, *2004*(May), 101–105. https://doi.org/10.1.1.2.477

MacLeod, L., Greiler, M., Storey, M. A., Bird, C., & Czerwonka, J. (2017). Code Reviewing in the Trenches: Understanding Challenges and Best Practices. *IEEE Software*. https://doi.org/10.1109/MS.2017.265100500

McIntosh, S., Kamei, Y., Adams, B., & Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality: a case study of the qt, VTK,

and ITK projects. *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, 192–201. https://doi.org/10.1145/2597073.2597076

Meneely, A., Williams, L., Snipes, W., & Osborne, J. (2008). Predicting failures with developer networks and social network analysis. *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering - SIGSOFT '08/FSE-16*, 13. https://doi.org/10.1145/1453101.1453106

Nelson, S., & Schumann, J. (2003). What makes a code review trustworthy? *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, *0*(Section 3), 10. https://doi.org/10.1109/HICSS.2004.1265711

Newman, M. E. J. (2014). *Networks: An Introduction*. *Cambridge Quarterly of Healthcare Ethics* (Vol. 23). https://doi.org/10.1017/S0963180113000479

Ohira, M., Ohsugi, N., Ohoka, T., & Matsumoto, K. (2005). Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. *ACM SIGSOFT Software Engineering Notes*, *30*(4), 1. https://doi.org/10.1145/1082983.1083163

Ouni, A., Kula, R. G., & Inoue, K. (2017). Search-based peer reviewers recommendation in modern code review. *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, 367–377. https://doi.org/10.1109/ICSME.2016.65

Rahman, M. M., Roy, C. K., & Kula, R. G. (2017). Predicting Usefulness of Code Review Comments Using Textual Features and Developer Experience. *IEEE International Working Conference on Mining Software Repositories*, (1), 215–226. https://doi.org/10.1109/MSR.2017.17

Rigby, P. C. (2011). Understanding Open Source Software Peer Review : Review Processes , Parameters and Statistical Models , and Underlying Behaviours and Mechanisms by.

Rigby, P. C., & Bird, C. (2013). Convergent contemporary software peer review practices. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, 202. https://doi.org/10.1145/2491411.2491444

Rigby, P. C., German, D. M., Cowen, L., & Storey, M.-A. (2014). Peer Review on Open-Source Software Projects. *ACM Transactions on Software Engineering and Methodology*, *23*(4), 1–33. https://doi.org/10.1145/2594458

Rigby, P. C., German, D. M., & Storey, M.-A. (2008). Open source software peer review practices: a case study of the apache server. *Proceedings of the 30th International Conference on Software Engineering*, 541–550. https://doi.org/10.1145/1368088.1368162

Rigby, P., Cleary, B., Painchaud, F., Storey, M. A., & German, D. (2012). Contemporary peer review in action: Lessons from open source development. *IEEE Software*, *29*(6), 56–61. https://doi.org/10.1109/MS.2012.24

Sauer, C., Ross Jeffery, D., Land, L., & Yetton, P. (2000). The effectiveness of software development technical reviews: A behaviorally motivated program of research. *IEEE Transactions on Software Engineering*, *26*(1), 1–14. https://doi.org/10.1109/32.825763

Shimagaki, J., Kamei, Y., McIntosh, S., Hassan, A. E., & Ubayashi, N. (2016). A study of the quality-impacting practices of modern code review at Sony mobile. *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16*, 212–221. https://doi.org/10.1145/2889160.2889243

Thongtanunam, P., McIntosh, S., Hassan, A. E., & Iida, H. (2015). Investigating code review practices in defective files: An empirical study of the Qt system. *IEEE International Working Conference on Mining Software Repositories*, *2015–Augus*, 168–179. https://doi.org/10.1109/MSR.2015.23

Tukey, J. W. (1977). *Exploratory Data Analysis*. *Analysis* (Vol. 2). https://doi.org/10.1007/978-1-4419-7976-6

Yang, X. (2014a). Categorizing Code Review Result with Social Networks Analysis : A Case Study on Three OSS Projects, 200–201.

Yang, X. (2014b). Social network analysis in open source software peer review. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, 820–822. https://doi.org/10.1145/2635868.2661682

Zanetti, M. S., Scholtes, I., Tessone, C. J., & Schweitzer, F. (2013). Categorizing bugs with social networks: A case study on four open source software communities. *ICSE '13 Proceedings of the 2013 International Conference on Software Engineering*, 1032–1041. https://doi.org/10.1109/ICSE.2013.6606653