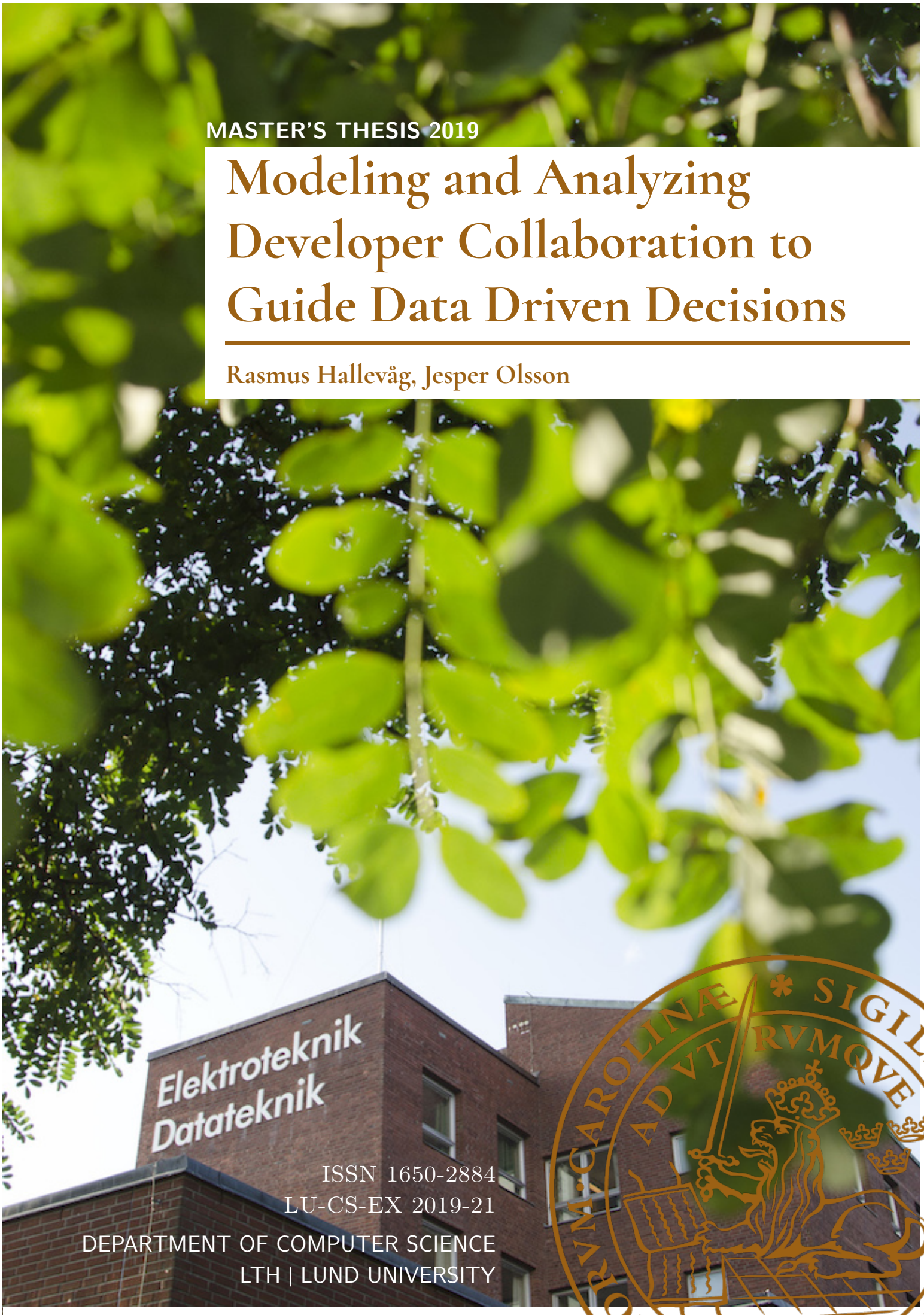


MASTER'S THESIS 2019

Modeling and Analyzing Developer Collaboration to Guide Data Driven Decisions

Rasmus Hallevåg, Jesper Olsson

ISSN 1650-2884
LU-CS-EX 2019-21

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-21

Modeling and Analyzing Developer Collaboration to Guide Data Driven Decisions

Rasmus Hallevåg, Jesper Olsson

Modeling and Analyzing Developer Collaboration to Guide Data Driven Decisions

Rasmus Hallevåg
dat13rha@student.lu.se

Jesper Olsson
dat12jol@student.lu.se

August 26, 2019

Master's thesis work carried out at Praqma AB.

Supervisors: Lars Bendix
Bo Nyström

Examiner: Ulf Asklund

Abstract

In a software development context a high number of decisions have to be made. For many of these decisions the insight of experts in the area is the deciding factor but the manual work to understand the context of a problem can potentially take a large amount of time for these experts and might not be entirely accurate due to human biases and heuristics. With such problems utilizing automatically mined metrics could be of aid. Finding new metrics that affect performance and quality can also lead to more relevant information being presented.

If team organization should follow the concept of collective code ownership or if the knowledge should be more stratified is a common debate in the software world. Creating an accurate representation of development structure and comparing it to performance metrics could help answer the debate and inform practices which would improve development. Two research questions have been formulated in this thesis to best explore this subject, namely if developer structure could be modeled and if there are some correlation between it and metrics relating to throughput or complexity.

As modern version control systems store a large amount of metadata, extracting and presenting the information stored could help investigate the research questions and potentially aid data driven decisions.

This was done through the creation of a prototype gathering and displaying representations of collaboration with other metrics. Some observations such as having more than 5 developers leading to more complex code was observable. However, further work continuing the research may be necessary to create and validate an accurate model.

Keywords: Mining Software Repositories, Data Driven Decicions, Collaboration, Ownership, Finding Correlations

Acknowledgements

We would like to thank our two supervisors Lars Bendix and Bosse Nyström for their support, feedback and guidance allowing this thesis to be created. We would also like to thank everyone in the Praqma Malmö office for the welcoming reception and good discussion of ideas. Special thanks to Samuel Ytterbrink from Praqma Gothenburg who provided many helpful hints regarding the extraction of data from git and also helped create a model for finding the integration speed from a git repository.

Contents

Chapter 1
Introduction

Humans are not naturally rational in their decision making, assistance with decision making is something that could be of great help in many contexts. This is especially true in a software development context where a multitude of decisions have to be made and with a high degree of frequency. Psychological studies of decision making has shown that personal biases and human heuristics affect decision making to a degree that prevents expected models of rational decisions from being accurate[1]. In a software development context it is common with a reliance on experts and seniority to make various decisions based on a hopefully high degree of domain knowledge with the proper biases and intuition. While completely replacing these expert opinions with machines would not be possible with the technology of today, presenting the right information could help the decision makers save time by presenting the right information.

While relying on decisions made by experts on a subject may be necessary in some contexts, doing it universally can come with flaws. Constant reliance on intuition of individuals could be problematic if there is no way to verify this intuition as there are no guarantees that the intuition is correct for this context. There are also problems with the availability of these experts, the only person who might know of an important section might have quit working on it and can no longer offer input. Also, experts does not scale, if an organisation expands then there will be an increased need for experts which might not always be possible so satisfy, maybe because it's hard to find people with the correct domain knowledge which would lead to people with less expertise needing to make decisions who's intuition might not be as developed.

Gathering the needed information in order to make an informed decision can also be a demanding task. If an expert is required to do this then that is time the expert can not spend on more productive tasks. So there is also a problem when it comes to available resources in order to make a good decision. By giving some aid in the decision making time could be saved for the decision makers, letting them focus on their work rather than manually figuring out the context of a problem.

As a starting point the research of this thesis considered the debate regarding how de-

velopers in a team should organize to increase performance and code quality. This could for example be through collective code ownership or non-collective code ownership. In the field, there exists differing opinions of how this should be optimized. Following this is the question of what definitions could be put on this. Should it be some sort of ownership, telling who feels like the responsible one for a section of software? Should it be a measurement of knowledge, showing who has the deepest level of understanding of the a section in a certain context? Should it be something different entirely? This leads into the first research question of the thesis being if there is a way to accurately model internal developer structure and level of collaboration of a project or part of a project.

While examining a succinctly and accurately modeled version of such a value alone may be of some interest there will then not be possible to directly consider its effects. It would therefore be of interest to examine how the value compares to metrics linked to performance or code quality. This leads into the second research question being if there is a relationship between the level of collaboration between developers and other metrics such as complexity or integration speed.

Research Question 1. *How can internal developer structure and level of collaboration in a project or part of a project be modeled?*

Research Question 2. *Is there a relationship between the level of collaboration between developers and complexity or integration speed?*

As a potential method of investigating the research questions or solving other problems in the field, automatically collected metrics could be utilized. These metrics could present the context of a problem and aid with decision making.

The new information could for example be used to find bottlenecks in the code and structure, or act as a ”canary bird” that show early indicators that things may be heading in an unsatisfactory direction. Historical values could be utilized for analysis and finding the ”canary bird” metric that can be used to assess future performance.

To investigate the research questions the appropriate data values needed to be extracted. While many sources could be utilized for this the version control system (VCS) git was chosen as the sole source of data due its ease of integration with a command line interface and to the authors familiarity with it.

Using data related to the developer structure and level of collaboration ideas for modelling this can be presented. By collecting a varied number of other metrics and plotting them against each other impact or causes of this can be analyzed. This is also something that can be done on various levels of the structure. Due to the particular data gathered this theses primarily focuses on full project level (in this case a single git repository) or individual file level. The data is then collected for a specified time frame or a connected series of time frames.

Due to various reasons the research was done in an open source setting and github was utilized as the specific source of repositories. As of 2018, over 30 million developers have made an account on github, contributing to over 90 million different repositories[2], making it suitable for finding a diverse selection of open source repositories. By mining data from these software repositories metrics can be generated automatically, fast and be a part of a bigger picture of metrics that can guide and inform decisions.

Chapter 2
Background

In order to be able to fully appreciate the thesis this chapter will set out to act as a reference and base, describing and explaining concepts that the thesis is building upon. A background to the field in general, what has been common viewpoints in the past compared to the present and how the development has shifted will give a more profound view of where the ideas for research questions were drawn and which problems or areas that will be able to be explored based on those research questions.

The chapter will start with giving context to the lay of the land. Then move into the context from which the thesis was created. The chapter will also lay out the methodology followed in the thesis and motivation behind said methodology. Important parts of the chapter to understand include the how and why the research was carried out in the way it was.

After giving context to the thesis, there is a theory section which is aimed at computer scientists who might need a refresher or exposure to the area of software configuration management, both in general and in relation to the topic of mining software repositories in particular. Reading this part can help a reader to fully appreciate the entire body of work that this thesis entails and depending on the confidence in the area it might be beneficial to read this section before research method. Its goal is to act as an introduction to the area and explain some of the key concepts used in this thesis. It will range from definitions of concepts in play to different applied technical techniques used.

2.1 Context

To understand the purpose of this thesis and its Research Questions some knowledge of the context the research was performed in could be of help. A description of the general context and state of the field leads into motivation of the research questions. This is followed by descriptions of more specific contexts of the thesis discussing the partnered company and the scope of the datasets.

2.1.1 General

Reflection of the subject field was the biggest inspiration of the Research Questions. There has long been a debate over how developers should organise to increase the quality and success of the code a team produces for a project. As Kent Beck suggests in eXtreme programming[3], programmers should express collective code ownership over the code they produce, arguing against compartmentalizing a team and creating silos. This notion is then supported by principles that shift development like development in pairs which increases the amount of eyeballs on all piece of code produced. The increase in eyeballs does not only act as a real time reviewer but as they the two collaborate and familiarize themselves with any piece of code there is an increase in people who understand that part of the system. As more people understand more parts of the code written by the team it becomes more robust to things like layoffs or vacations, as there will not be a case were the one person who has written is the only one who understands it, and might be on a 3 month parental leave.

Different companies have tried versions of this, notably the likes of Microsoft with Windows XP, but the real impact of changing a structure or practice this way remain unclear. Only going anecdotal reports does not uphold any scientific standard, and while some companies are compliant with a ”trust me, it works” from Microsoft or a programming guru like Kent Beck it does not satisfy inquisitive persons from going, ”Why does it work?”. If there is no way of measuring a change, then there is no way of knowing if the change was positive or not.

This shift in mentality has been very noticeable in the recent time. The 2018 state of DevOps report by DORA hit like a bomb, and has gained almost a rockstar following. The report got released along with a book called ”Accelarate” written by the conductors of the report [4]. Nicole Forsgren with a rigorous background in statistics along with the larger than life figure Jez Humble who co-wrote ”Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation” which is one of the most important pieces of literature regarding the science behind agile development and Gene Kim who is one of the most important thought factories behind the DevOps revolution. This star studded roster explains the importance of metrics and measurements and also carefully go through how the results which statistically proves that four measurements has a direct correlation with how the successful the project will be. The four key metrics are lead time, deployment frequency, mean time to recover, and change fail percentage. Being able to prove a relationship between another metric and one of these metrics would therefore by inference prove the relationship of that metric and the likeliness of success for that project.

Previous attempts have been made to analyse the impact of amount of developers on a file, mainly interested in the amount of bugs connected to the file. This has been done with a spread results. Christian Bird et al.[5] has conducted research where the results suggest that the modules with more people working generates the highest frequency of bugs. Meanwhile, Foyzur Rahman and Premkumar Devanbu have looked on a finer level and claim that bugs are more likely to be found by ”the collective” and not introduced by them [6]. Because of how bugs are reported, where only bugs found are made visible for that statistics, amount of found bugs vary greatly, and more bugs found might not mean that there are more bugs in the code. Bugs can be more or less visible, and finding more bugs can be a positive things, as only bugs that are found can be fixed, at least intentionally.

Another distinction is the notion of shared, or collective code ownership and non-ownership.

In a collective code ownership Beck means a state where everyone is responsible for the code, and therefore also allowed to change everywhere. This requires a knowledge to be spread which is why pair programming is a recommended practice along with it for knowledge to be spread in order to facilitate everyone's ability to own and change everywhere. If how it is not well facilitated and no one is responsible for anything then a state of non ownership can be reached. In this state changes can be made everywhere but responsibility for the code is not shouldered by neither the team or a person. When no one is responsible then quality suffers.

This is what has sparked the basis for our first research question *How can internal developer structure and level of collaboration in a project or part of a project be modeled?* Which is a generalisation of how to measure collective code ownership. In order to get a point of reference in which analysis can be done upon this is pertinent. If it can be modelled effectively then it can be used to cross-correlate and analyse its impact on a project. Finding an effective model which can be proven to yield the correct metric is therefore of high importance in order to make further development in the field.

The second research question is a continuation of the first one, if a model is found. *Is there a relationship between the level of collaboration between developers and complexity or integration speed?* By analysing the states of a project and how they shift, can a relationship between them be established. If it can then this will yield information which can be used to guide decisions of how development should be made to increase the quality and or productivity.

If these research questions were to be thoroughly answered it could help development teams everywhere in making informed decisions. If the level of collective ownership is shown to have low or project-dependant effects it would allow for working in a preferred manner without worrying about adhering to a specific standard for structure. If high levels of effect are found it could serve as a help for teams looking to increase performance through restructuring.

It is also interesting as mapping out variables and parameters of this kind can be used as a basis for machine learning. Neural networks can be used along with the parameters in order to make predictions of how the future will develop. This would allow for a lot of future work.

2.1.2 Praqma

The research has been conducted at the facilities of Praqma. Praqma is a consultant company who sends out their experts to customer companies. The experts then aid development and work with the company to set up agile/lean practices, service solutions or, makes assessments on how the company works. Their work revolves around being experts not only in the philosophy of development but also being experts on a wide range of tools and activities surrounding development. Praqma employees frequently hold courses in git and help other companies migrate from other version control systems towards git. Their knowledge in git provided help for understanding the underlying mechanisms of git and how it could be manipulated to gather the data required to carry out the thesis.

What Praqma gained in this collaboration was a open source implementation of the prototype which was created to investigate the research questions and a blog post summarizing the work. The prototype is intended for free use by companies and Praqma's consultants can use it as a basis for future feature extractions. What the thesis gained from the collaboration

is access to a network of people with experience in the subject field to discuss ideas and get assistance with technical troubles.

By collaborating with Praqma there was a hope of using their connections in order to gain access to traditional companies developing closed source software to investigate the effects for that context. Companies showed concern for business secrets and this became an issue despite efforts to anonymize the data and making sure no information from the actual source code or the structure of their organisation would be leaked were made. Because of the limited time period the thesis was carried out with a decision to analyze open source repositories. This decision was made as the time it took to get a response from people within companies could range several weeks.

2.1.3 Open- and Closed Source

Instead of focusing the analysis on one company, analysing multiple open source projects allowed for a wider range of data sources. However the structure of open source projects differ from normal companies despite some some aspects such as quality and throughput being universally desirable. In open source projects anyone can use, contribute and be involved in the development meaning that the structure of how people work on it can vary greatly. Typically development in open source projects is more distributed where developers all around the world contribute to different parts of the project because of differences in geographic location. There is also a looser definition of a team, as there usually are a set of few developers who are seen as the (hopefully benevolent) dictators, who decide if the contributions made by other developers will be added into the main repository.

Some disadvantages of doing the analysis of this thesis on open source projects is that traditional collaboration is less common due to geographical distribution and that there may not exist a clear team structure. There may also be difficulties in interviewing insiders from the project as they might not be available or interested. An advantage is that a larger quantity of projects can be examined as they can be directly downloaded and there is no need for an elusive non-disclosure agreement.

In a closed source environment there are hierarchies in titles and positions at the company. Companies also create teams or divisions with a separation of labour which is distributed within the company. The teams have one set of structure in the hierarchy of the company, then a structure of the social dynamics of the working place, who is friends with whom and who is sitting close to whom.

Disadvantages of using closed source environment for a data mining project include the difficulty of finding a company that allows outsiders to examine metadata and present parts of it in a report. An advantage would be that it would be easier to perform an in-depth case study if legal rights are obtained.

These contrasting structures makes the context of development disparate meaning that the analysis made for one set of them not necessarily apply to the other. Analysis will have to be made as to which metrics can be applied generally to both open and closed sourced project and which that are specific.

As no suitable closed source project could be found a decision was made to examine open source projects. This also influenced the decision to examine a larger quantity of repositories in less detail as this was made possible by utilizing public open source projects.

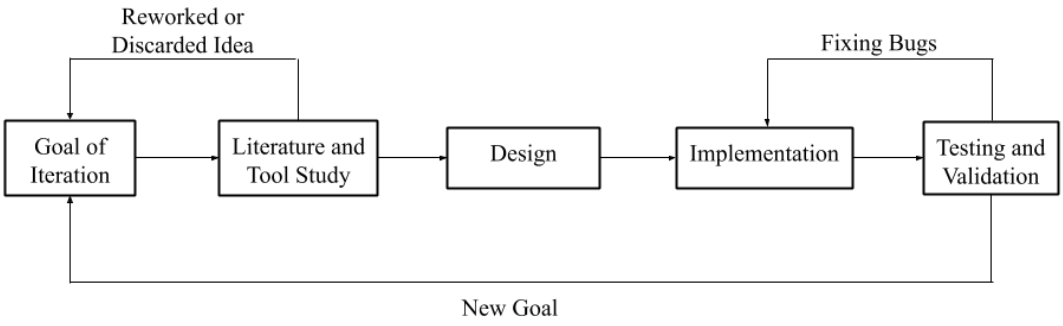


Figure 2.1: The iterative process of the Research Method

2.2 Research Method

This section aims to give an insight to how the research was structured. It will tell the story of how the steps were taken to move from ideas to a proof of concept generating results which could be analyzed. Along with stating how the research was made, it will also present motivations for the choices that were made. After this chapter it should be made clear as to why the thesis is built on git, why the certain models were chosen and why 107 different open source projects were analyzed.

2.2.1 General Method

Large parts of the research used an iterative approach that repeated a method several times to find different results. While a more rigid plan was initially considered the iterative approach was deemed more fitting as it would be more forgiving of errors and allow planning of later stages based on results from earlier ones. A flowchart summarizing an iteration as used for the research in this paper is shown in figure 2.1. The goal of using this approach was to divide the research questions in more manageable subgoals rather than the potentially insurmountable obstacle that a more rigid approach might have presented.

The first step of this approach is to decide on the goal of the iteration. In most this meant extracting a specific type of data and potentially processing this data to create new values, with the final method for processing still not being decided upon. This is followed by a short literature and tool examination to see if something similar have been done before. If similar extractions have been done with an accessible tool it may be utilized for extraction. If earlier research uses models similar to the end goal they may be used as is or in an altered form for later parts. In some cases ideas can be deemed too difficult or impossible with the utilized resources and discarded or reworked. With a larger picture in mind a decision can be made of what should actually be extracted and processed. This may include several complementing or alternative values. The plans are realized in an implementation phase where scripts that collect and process data are written. Finally the obtained results are examined and tested and validated. If the output appears to be faulty the code is examined and found bugs are corrected in order to increase the validity of results. If the results are deemed to be overly limited in usage or redundant due to similarity to other values they may be skipped in later iterations. The testing and validation is done continually and missed issues from earlier phases

can be fixed in later ones. While more extensive testing may have been preferable many of the issues are linked to specific repositories that were used and as the list of used repositories was extended more issues were illuminated. As the used data sources was changing throughout the research to complement current needs and grow more exhaustive this was difficult to plan around.

2.2.2 Workflow

For a more specific explanation of the overall workflow the research took off with a general literature studies with a goal to try and find an effective quantification of an "Expert Factor" that would provide a measurement of collaboration and internal developer structure. Another early goal was to determine the scope and specifications for a final prototype. A goal was to consider several definitions of this Expert Factor and test them through an experimental method where different representations would be tried and compared against each other to find an acceptable representation. Similar representations could be implemented in the same iteration while differing ones were saved for later iterations. From early on inspiration was taken from the fractal figures and fractal values as presented by D'Ambros et al. [7] and this was used as a basis for one of the utilized models. This representation was seen as appealing due to its scalability in being possible to examine at differing levels such as on single files or on larger component, as well as the data required for it being relatively simple to extract. This value had previously been compared to the number of bugs and comparing it to other values was a goal for later stages. Other inspiration was taken from a master thesis relating to network analysis[8] where a project was presented as a network where contributors are represented as nodes and collaborations are represented as edges. This network was built using the same data as the fractal value but served as an alternative way to examine the problem and allowing for differing values to be extracted from the built network and a version of this network representation was used as a basis for another used model. Other models for viewing collaborations such as the line based model discussed by Tubor Gîrba et al.[9] was evaluated, but as it distributes removed/changed lines over all developers in the file evenly it was deemed to not be robust against cases such as reformatting and adding documentation among other things. The implementation cost of that model along with its added flaws led to it not being pursued.

Aside from the Expert Factor a goal from the start was to extract a variety of other data to compare it to. A goal was to select points of comparison accepted as being linked to performance. For this reason examination of the Accelerate[4] metrics was performed. Out of the four presented measurements the stability metrics which contain the mean time to recover after system failure and a change failure rate measuring how many changes lead to bugs both required sources outside of git. These two metrics would be highly interesting to extract but because this thesis did not have access to the sources required to extract them it was decided against. The sources to extract those metrics could be either tags or labels in a version control system which would need to be unified and properly filled in, or access to a bug tracker with commit messages that connects to the issues in the tracker. While extraction is definitely possible it is highly specific to each project making it hard to create a general extraction required in order to examine multiple projects which was done by this thesis.

The other two measurements were the throughput metrics containing deployment frequency and the lead time between a change is made and it running in production. While

Github contains a release functionality that could have functioned as a deployment frequency this feature was found to have varying methods and degrees of usage in the examined open source projects leading to deployment frequency not being measured in this thesis. Finally, the lead time was considered the most accessible metric utilizing only git and a version of this was chosen for further examination. While there was no guarantee that this is equivalent of code running in production for every examined open source project the time between code being first committed in a branch and it being integrated into the main branch was used as a definition of this for the results and discussions of this thesis. For a sanity check the balance of commits in branches to total number of commits was also considered. If a majority of commits are done directly in branches before merging into main it may be a valid assumption that the code is tested and cleanly integrated before merging in.

The implementation phases was to lead to the creation of a software prototype. The purpose of this software prototype was to serve as a proof of concept tool for extracting and processing the desired values. There was an attempt to balance quick and efficient implementation with making clean and readable code as to ease later examination and bug fixing. As it was only meant as a proof of concept usability was not a priority.

As a way of producing a modular prototype where work was easy to divide and to simplify understanding for people only interested in either data gathering or visualization, the prototype is divided in two parts. One is a collection of scripts for extracting and processing data from a git repository and saving selected values in a file. The other is used to visualize the values from the generated files by generating graphs. Due to the nature of the research creation of a running prototype was deemed an important step as validation of proposed models would have been overly difficult if a paper prototype was used.

To test and validate the prototype as well as to perform some basic analysis of results the generated graphs and files were examined. If values seemed odd or faulty the source code was examined and bugs were fixed. This was repeated until the values were found acceptable.

Rounding off the research there was a phase of more dedicated analysis where results and findings of every phase of the work was searched for and examined. Due to time constraints and finding older bugs in the later examination this phase was cut somewhat short but still provided some valid results. An earlier goal for validating the results was an interview with knowledgeable contributors from one of the examined projects, however no suitable agreements for such an interview was possible and this had to be cut from the scope of the thesis.

2.2.3 Alternate Methods

There exist other research wherein alternate approaches to mining a software repository is presented. In one paper by D'Ambros et al.[10] a general approach to software repository analysis utilizing a Release History Database where relevant information is stored. This database is used both in a similar manner to this thesis with an analysis of developer effort, and for different analysis such as finding hot-spots and coupled files. As the particular database presented in the paper requires examined repositories to use the version control system CVS alongside the bug tracker Bugzilla it would not have bee suitable for the research of this thesis. Utilizing a database to save all data that was to be examined may have been a preferable approach toward later stages but a modular approach with different files collecting different data was seen as preferable in early stages due to increased ability to quickly try out new features and fix issues in older data handling. For later stages switching to a database approach

was considered but decided against due to time constraints.

One aspect that could have been considered is where the hot-spots of the code are. There are some slight variations of what the definition of a hot-spot is. D'Ambros et al.[10] sees it as volatile, dependent and often buggy code and looks at a wide variety of metrics and how they change over time to decide this. Adam Tornhill[11] uses a simpler definition that only looks at complexity and change rate. Tornhill's simpler definition fit the scope of this thesis better but no in depth analysis of a model based on this was used due to limitations in scope and tine.

Gîrba et al.[9] suggests defining ownership of code by looking at who last edited every line of code. A formula for approximating this in CVS is presented but in git the last editor of every line is explicitly extractable. This method was examined in earlier phases. This has some advantages over examination of commits such as being applicable to all code currently running in production while ignoring code that is no longer relevant. It does however have some drawbacks such as it being more difficult to view recent changes as more relevant, contributions that only remove code not being considered and small edits of lines giving full ownership of the line to the editor over the original author as any formatting change such as indention or reordering would give a large ownership percentage compared to the change that actually was made. Overall there were stronger arguments for examining commits and it was chosen as the main basis for this thesis but similar studies could be made using owned lines.

Alonso et al.[12] uses a different view on the subject matter where the expertise of a developer is measured by words used in subject matter of changes and commit messages. This provides a differing method but ultimately felt disconnected from the chosen research questions of this thesis.

A method of defining ownership using bugs and bug fixes is presented by Rahman et al.[6]. While it would have led to interesting analysis where bugs and bug fixes appears it is not generally extractable from git alone and would require external bug trackers.

Orucevic-Alagic and Höst[13] presented the definition of the network representation and uses it in a case study examining the larger scale android network. The work of this thesis examines it in a different context by looking at a larger number of networks in less detail.

2.2.4 Why Git?

As the sole source of data, the version control system git[14] was chosen. This was done for several reasons. One reason git was chosen over other version control systems was that it is easily usable from the command line, making data extraction utilizing the command line through parsing easier. This also leads to git being easy to integrate into scripts, which made it suitable for building the prototype upon. The widespread usage of git, particularly in an open source context, opened a possibility to examine a wide variety of sources. Git also supports migrating from several other version control systems[15], allowing for a wide variety of historical data despite the widespread usage of git being relatively new. Another reason is that the authors has experience in git. Said experience allow commands to be utilized without spending too much time on tutorials. As to why no complementing sources of data such as a bug tracker or a mailing list was used the reasoning was that these sources are often project specific and there was unwillingness to lock down to one project. If there had been a possibility to conduct the research on a proprietary repository rather than just open source

repositories there would likely have been some tailoring for the specific source but as no suitable proprietary projects were found this was deemed too limiting.

2.2.5 Picking Specific Sources

In order to investigate the structural collaboration between developers central to RQ1 there was a need to analyze repositories with collaboration as a focal point. Multiple people writing on a file are collaborating on a file level, multiple people working on the same project are collaborating on a project level even though they might not work on the same file. If there is only one developer connected to a repository then no collaboration has occurred either on project or file level. That is the reason for selecting projects where collaboration can be determined to have existed. While having a bigger corpus and including all different types of repositories would have been preferable for the sake of more generalizable results, the time frame for this thesis made it so that a selection of focus had to be made.

For specific projects to look at 107 different GitHub[2] repositories were chosen. An attempt was made to select repositories of varying size and origin but avoiding trivially small repositories with 1 or 2 contributors or next to no source code files. As the tool Lizard[16] that was utilized to extract complexity metrics requires examined files to be of certain programming languages projects that mostly used these languages were selected. As the final tool looked at the same period of time for comparison there was an attempt to select projects that have had activity between 2017 and 2019. Dynamically selecting an active period for each repository would have been possible but was not done due to time constraints. While the Linux kernel repository was initially examined it was ultimately decided against because of the high level of activity leading to performance issues and values being on a different scale than those of other repositories. In some rare cases the naming in the file structure provided problems for the parsing, for example when spaces were used in folder names. As these projects were rare it was deemed more effective to replace them than to implement support for every possible file structure.

The reason for analyzing over a hundred repositories is to find statistically significant generalizable results. Instead of focusing in on a few repositories and analysing them more in depth. If access to a closed source project was granted then it would likely have been the other way around and gone into full depth about that specific project, validating the results with the developers in that project. Going in-depth could also have been possible for a few selected open source repositories, but with access to a high amount of open source repositories the thesis valued establish relations between different metrics over analysing information could be extracted for the given context.

2.3 Theory

To ease understanding of this master thesis some fundamental concepts relating to the field of Software Configuration Management (SCM) will be laid out. Firstly the concept of Configuration Identification and why we care about it will be discussed. Then the topic will be broadened to configuration status accounting (CSA) and configuration management database (CMDB) which is the theory which this thesis will apply in practice. The version control system git is used as the sole source of collected data throughout the thesis and some basic functionality of

git will therefore be explained. Git contains certain elements of a CMDB and CSA which will be used to extract additional information in line with CSA. How git functions as a CMDB and CSA will be also be covered followed by a section of higher detail covering the methods in git that were used in this thesis. Outside of SCM there will be a short explanation of the weighted directed networks used for parts of the thesis.

2.3.1 Configuration Identification:

Identifying which items are tracked is an early step in gathering information from these items. Configuration identification deals with defining which parts of software system will be controlled as in stored and given unique nomenclature.[17] Kelly [18] defines a configuration item (CI) as any part of the system which is independently identified, stored, tested, changed, delivered and or maintained. She also argues for how this does not only include source code, but all things developers could care about including things like documentation, pictures and notes or test data information.

The definition of a CI can also be looked at in contrast to the definition of an Artefact. An Artefact is anything related to the system regardless of if it should be identified and maintained, making CIs a subset of Artifacts. For a CI there is a guarantee of persistence but an Artifact might be transient. The CIs will be worked on and transformed by developers during the course of a project and as a result the history of how the CIs have changed. A representation of different versions of a CI being stored is shown in figure 2.2. In this figure three versions of the same configuration item are shown. While version 3 is the latest the historic contents of versions 1 and 2 are saved with the contributors of each version serving as an example of stored metadata.

Bersoff and Davis outlines that there needs to be the history of changes that were made, who made them and why they were made.[17] This type of metadata is related to CSA which will be covered later but it is important to highlight the connection between CIs as the items that are important and will be given the treatment of collecting and recording data regarding changes. This means that CIs also includes metadata, which is stored in version control systems as git. The information about a CI is therefore also a CI, and this is what this thesis in interested in extracting and analysing. Some of the metrics analysed like amount of commiters, lines added and integration speed comes directly from these metadata CIs, while others such as number of functions and cyclomatic complexity is generated by static analysis of the source code CIs.

From the wide range of CIs which will exist in the project this thesis is interested in source code files and the metadata related to these files. The reason being is to limit the results to coding as a way to clean the large output of different results and staying related to the subject field and interesting to both Praqma and the Computer Science faculty. Limiting the results to source code also allows for code-specific values such as complexity to be collected.

2.3.2 Configuration Management Database:

Tracking changes and files is an important step of the research of this essay and to do that they must be stored somewhere. In order to store and share the CIs between each other there needs to be some sort of electronic library where all the CIs are stored, such a library is called configuration management database (CMDB).

Figure 2.2: Versions of a CI

Submitting and retrieving files should be easy and as a library the CMDB is expected to be well ordered so that searching for what currently is needed is easy.

It can be seen to have a primarily administrative purpose, but great benefits exists if the CMDB is fast and contains all information necessary for work to begin as what is unique for a CMDB is that transformations on files are very common and necessary for the development to carry on forward.[18]

Both Kelly and Daniels describes a database where not only files are important, but information surrounding the files[18][19]. Daniels separates the information into three categories; control which deals with versions, if it is reviewed and planned future work for the file etc; identification, which contains identifiers for the file, connected requirement or story and which developers who are responsible for it; documentation which contains things like documented changes and reason for them.

After data mining files a large collection of raw metadata can be obtained and transforming said raw data to more specific and useful information is an important step.

While what is supposed to be included in the CMDB can be subject for riveting debates, Daniels offers one view of how he thinks it should be structured. The main idea being that it is not only important to be able to retrieve and update files, it is required for a CMDB to also be able to handle information about the file. The types of information that should be stored is handled by the field of configuration status accounting.

CMDB is central to what information this thesis extracts from the different tools. How it is accessed and stored, and which types of information could be beneficial for the tools to either append or for a future tool to include as a selection criteria.

2.3.3 Configuration Status Accounting:

Configuration Status Accounting (CSA) is an important part of datamining CIs and provides background on the origin of the collected information. CSA is defined by Daniels[19] as the discipline of recording and reporting the status of CIs in the project, providing feedback and tracability for the project.

The recorded information can include things such as which versions exists of the CI, which one is the most current one as well and current status of change requests. There may also exist metadata regarding changes of files like who has changed what, when did the changes occur or who is working on it right now. For collections or individual CIs accounting could include test information regarding performance or functionality.

2.3.4 Version Control Systems:

There are various CM tools which aims to act as a CMDB, recording and tracking changes to CIs. Usually they are called version control systems (VCS), of these some prominent ones are Git, PerForce and Mercurial. Central to them is that there exists a CMDB called the main or global repository which stores all previous configurations of the CIs.

Developers can either copy the repository to a local machine, usually referred to the local repository or their workspace or check out individual files that they which to work on. Transformations made to their workspace will not affect the global repository until they select which changes they want to add and then apply it on top of the global repository. The model of checking in and out files is described as the Checkout/Checkin model by Feiler[20].

2.3.5 Git as a CMDB

Git is a VCS containing features such as independent switchable and mergeable branches and a distributed workflow where the repository is cloned rather than checked out[14]. It was chosen as a basis for the thesis because of its widespread usage which makes finding suitable projects easy and the open source nature allowing for usage without licences. This section will describe what data is stored in git and explain the methods behind querying git to extract its data.

Git stores a delta containing the changes to files as what they call a commit. An example of this is shown in figure 2.3. In this figure changes are stored as a delta keeping track of all changes between versions. In the figure fields representing the files A, B and C are shown in color for a version when they have been updated for that particular version and in white with dotted borders if the latest change took place in a previous version, representing the metadata about when the file last was edited. Each delta contains a set of code changes that is the difference from one version to the other but it also stores information about the commit. This information is accessed through calling the git log, any version is retrievable and there are selection criteria that can be carried out.

When a developer commits a change to a git repository, metadata regarding this commit will be stored in a retrievable log. This data includes author, timestamp, files changed, number of lines added and deleted and a potential commit message from the author. It is possible to see the differences in a file compared from any two arbitrary versions of it.

This thesis will be focused on extracting data from the change history of the files. Specifically binding authors efforts made to the files, considering the sizes, dates and amount of interactions with files.

2.3.6 Extracting Information in Git:

A number of commands exists to retrieve the data from the git CMDB[21]. `git log` and its summarized form `git shortlog` can access data about each commit and can be used on the full project, individual folders or individual files. By default `git log` retrieves commit hash, author, date, and commit message for each commit. Options exists to additional data including changed files, additions, deletions and the option `--follow` can be used to look at all commits to a file including before a potential rename or move. `git blame` can look at each individual line in a file and show at what commit it was last edited and who was the

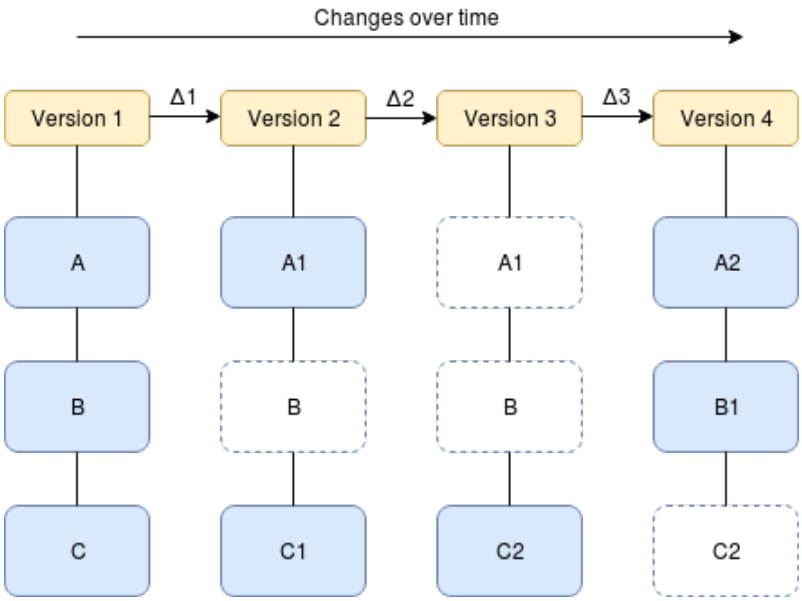


Figure 2.3: How git stores and tracks changes

author of this commit. By analysing the output of these commands and the raw data can be collected and processed into a more specific piece of information.

2.3.7 Network Analysis

The networks used throughout the thesis are weighted directed networks where there between every pair of nodes i, j exist two weighted vertexes, namely w_{ij} from i to j and w_{ji} from j to i. Given this basis a simplified definition of the vertex strength of a vertex in a weighted graphs given by Barrat et al.[22] would be given by $vs_i = \sum_{j \in N(v)} w_{ij}$ where $N(v)$ is the set of all vertexes. For directed graphs this definition can be expanded to give one in-vertex strength $vs_i^{in} = \sum_{j \in N(v)} w_{ji}$ and one out-vertex strength $vs_i^{out} = \sum_{j \in N(v)} w_{ij}$

Chapter 3
Design

To explore the research questions of this thesis a prototype incorporating the utilized models has been created. This chapter will explain the overall design and methodology of modeling various values as detailed explanations of the specific models used and how they have been integrated with the prototype. With clear specifications of how they were extracted and processed. The data extraction follows from these models explanations of what work was carried out in this thesis and how this was done along with motivating why it was done in this way and comparing the selected method to alternate methods. The subchapters will present each parameter and model examined in this thesis and how it relates to the research questions. The section will be of interest to a reader that want to understand the nuts and bolts of the designed prototype and data collection.

Aside from the technical explanations the utilized models of putting the data together are explained and motivated. For creating non-trivial pieces of information using metadata relating to files, knowing the model of how these values are to be built is important. Investigating how to to put these together to have it model the required aspects with sufficient accuracy. In the case of a new model or a usage in a new context tests of validity and utility may also be in order.

3.1 Overall Design

To explore the research questions, mathematical representations of the concepts are needed to be established, these mathematical representations is in chapter referred to as models. Different aspects such as the level of collaboration or the integration speed in a project will need to be modeled and measured.

By experimenting with several models of the same subject matter or using one proven successful in previous research more trusted models could be utilized. To verify and test these models as well as gathering the data neccessary for the cross-referencing of research question 2 a prototype for gathering data for and utilizing the selected models had to be constructed.

As manual extraction and processing, while used in the initial phase of building the prototype, would be time consuming and prone to manual mistakes a programmed prototype was constructed rather than a theoretical paper prototype that might not have allowed for sufficient testing and validation.

There was no software available that allowed for extraction of all the models needed for the thesis. Where it was possible external tools were used when following the specifications of the models in the thesis, in those cases the tools were integrated into the prototype to allow automation of execution. By automating the extraction it allowed for examination a wide range of repositories while also allowing for extensions, both during this thesis and for potential as a basis for future research.

The following section will explain how this prototype works in a more general sense, giving some background that could be of interest in understanding the more specific explanations.

3.1.1 Specification

The prototype is based on a collection of created scripts utilising bash, python and R. As performance was not a primary focus these scripting languages were powerful enough to get a satisfying level of performance making it possible to iterate and rerun an extraction and processing for a specific project in less than five minutes. Bash scripts was chosen for its wrapping functionality, binding different scripts together and having direct access to the same git commands designed for the bash terminal. Python was chosen because of its simplicity when it comes to libraries for line by line manipulating along with its performance being good for a scripting language. R was the library of choice when creating visual representations as there exists several libraries for appending different statistical methods in visualisation and also due to its ability to generate interactive plots. The interactivity increased the usefulness making it possible to hover over outliers and see additional information which could be loaded into a point of the plot [23].

A representation of how the prototype works is shown in figure 3.1. The scripts were separated and generated temporary files to dump data into, those files were processed by another script, turning that raw data dump into structured comma separated value (CSV) files. These csv-files contained the processed information and another script allowed for unifying the different CSV-files depending on the level of extraction. The file level data all had a field with the filename where all CSV files on that level could be unified into one big csv-file containing all extracted and processed data regarding the specific files. The same was the case for the project data, where intermediate files held data which could be processed into one big project data CSV file. On a project level different projects could be appended into one big project data CSV file, or multiple time periods for one project could be appended for a trend analysis on that project.

Because of the modular design where scripts could generate data into a intermediate form then be processed by an independent script it allowed for extending functionality as the extraction could be kept but processing changed or vice versa. It also allows for future research to be built on top of it as the scripts function as separate units and are unified on a high level, if an arbitrary number of the extractions are wanted as a base for new comparisons it would be easy to extend or modify the base prototype for that need.

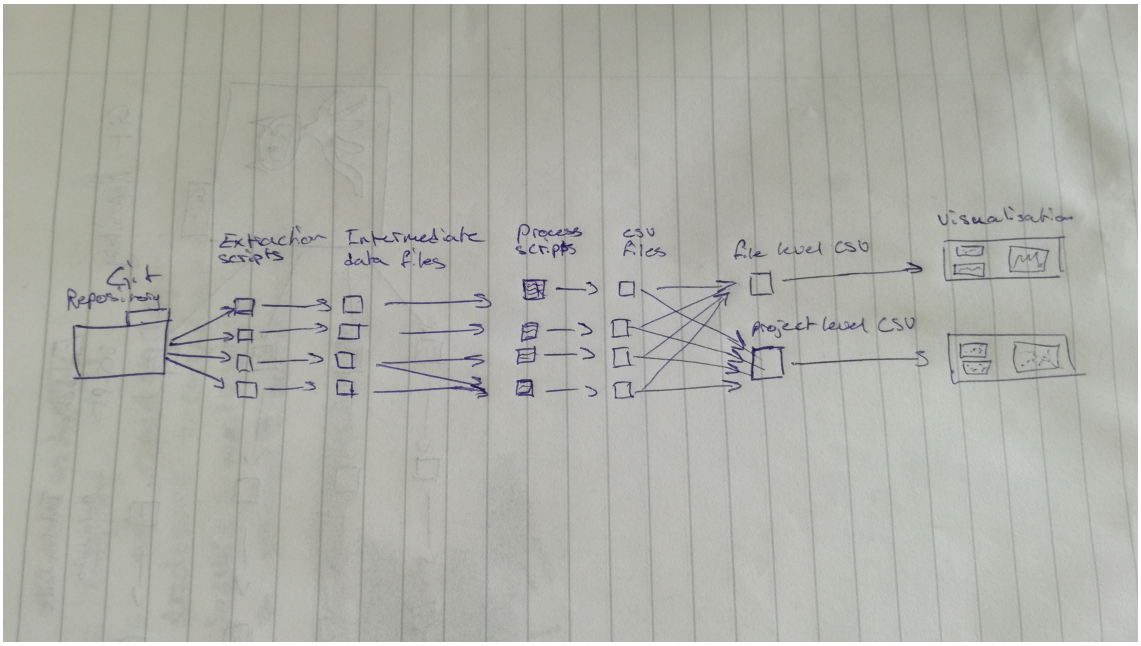


Figure 3.1: Data flow for prototype

3.2 Measuring Collaboration: Fractal Value

Measuring collaboration is the cornerstone of this thesis, exploring and evaluating different ways of modelling collaboration is our first research question. The model chosen will directly impact the second research question where the thesis will relate the collaboration metrics to performance metrics.

As the basis of the thesis relies on the model chosen for representing collaboration, exploring and analysing different ways of representing collaboration was done. Hence this chapter will emphasize motivating the choices that were made and highlight alternative paths that can be of interest for future research.

The following section will describe the origin and specification of the fractal value which is the primary collaboration model chosen.

3.2.1 Motivation

The fractal value is one of the measurements of number and distribution of contributors working on a file or component. With it it becomes possible to tell apart the almost chaotic parts of very high collaboration with more rigid structures where one or a few people do most of the work.

A big reason for picking the fractal value as one of the main values to examine was because it was customizable and scalable, making it possible to examine at different levels, allowing for single file examination as well as full project examination. Another reason was that it was simple and generalizable. All that is needed is extracting commits and their authors which is possible on every git repository and would also be possible in other version control systems. There was also a possibility to experiment with a basis for the model other than number of commits which is something that was done in the early stages of the research done by this

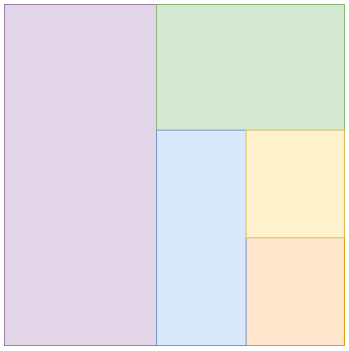


Figure 3.2: Fractal representation of distributed development on a component

thesis.

To compare results of using the model on high and low levels a decicion was made to examine the value file and project levels. It would also have been interesting to examine the value on a level between those but there were some troubles in deciding and limiting the extraction of the value on a proper level. A component level would have led to interesting analysis but an accurate and generalizable way of dividing a repository in components was not found as preliminary examination of different repositories showed varying structure of components. Another medium-scope level that could have been examined is on a directory level which is something that has been done in previous research using the value[7]. This would have been done by calculating the value using commits that edited a file being located in the specific folder to be examined and repeating this for every folder containing at least one file that there is interest in. While examining this level may have led to a better representation of the value than the file and project representations it was ultimately decided against due time constraints leading to an overall focus on the file and project level and the fact that no other metrics were specifically collected on a folder level.

As some of the details are lost there are some drawbacks of the fractal value as compared to the fractal figure visualization. For example, the pattern with few balanced developers and the one with one major and many minor developers may result in the same fractal value despite different distributions but by cross-referencing the value to the number of contributors some more detail can be seen.

3.2.2 Specification

The fractal value originated in a paper by D'Ambros, Lanza and Gall[7] and was originally meant as a complement to the fractal figure visualization shown in figure 3.2. Using this technique for visualizing distributed development one figure represents a component of a repository like a file or a folder and the different-colored sections of the figure represents one contributor. The relative sizes of the figure shows how large the contribution of every developer is in comparison to the other developers. Four major differing patterns of distribution were found using this model, one developer, few balanced developers (shown in figure 3.2), one major with many minor developers and many balanced developers. Out of these patterns, the first and last pattern can be inferred from the value with a value of 0 or a value nearing one respectively. However, a flaw with the value compared to the figure is that it may

be difficult to tell the difference between few balanced developers or one major and many minor developers.

The fractal value is a distillation of this figure into a singular value using the formula $1 - \sum_{\alpha_i \in A}(\frac{nc(\alpha_i)}{NC})^2$ where $A = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ is the set of contributors, $nc(\alpha_i)$ is the number of commits in the selected scope by contributor α_i and NC is the total number of commits in the selected scope. The scope is customizable and can include a file or set of files along with a timespan. The value can go from a minimum of 0 meaning only one developer to a theoretical maximum of 1 where nearing it means that there is a large amount of contributors that all have done very minor contributions. Higher fractal values would point to a lack of clear ownership while lower would mean stronger ownership at the expense of collaboration.

In the prototype a fractal value is calculated on file and project level. From the git log all commits over a specific time period, the author of the commit and the changed files are stored in a temporary file. Merge commits were excluded from the examined commits, partially because they primarily put together other contributions rather than being a new contribution and partially because the log of merges did not always show what files were affected. The projectwide fractal value is calculated using all of the stored commits while file-specific fractal values can be calculated for every individual file using the commit data where that particular file was edited. The file-level values can also be put together as a set of mean value, maximum value and standard deviation covering the values on project level, giving two seperate representations on this level. While the value is possible to extract for every file in the final prototype it is limited to source code files of certain programming languages as source code files was of greater interest from both an academic and industrial viewpoint and to preserve compatibility with other metrics such as complexity where the free tool used was limited to certain programming languages.

3.2.3 Alternatives

An alternative of using the basic version of the fractal value would be a slight alteration using a basis other than the number of commits, replacing $nc(\alpha_i)$ and NC from the definition with appropriate values. Earlier versions of the prototype calculated versions of this value using added lines, deleted lines and owned lines as defined by who last edited a line in addition to number of commits. The kind of line examination done had flaws as some outliers had extremely high number without being an important contribution due to things like automatically generated files. This reason along with the results being similar for all four models led to the focus being put on number of commits for later stages. Pure additions and deletions are still collected and used for some cross-reference with other values but the distribution between developers is no longer considered as results from previous experimentation pointed to number of commits leading to the best representation of the value.

A utilization of the owned lines described by Gîrba et al.[9] This visualization contains a line showing who the owner of a file is and the way that the owner changes led to the identification of several patterns of collaboration or lack thereof. While this would lead to interesting analysis on its own it would be difficult to distill into one value to utilize for cross-correlation and therefore not examined by the research in this thesis.

Liu et al.[24] presented a differing method for visualization looking at a more temporal level where the when and by who of activity is examined. This is shown in a graph where the number of operations per day is shown with different lines representing different developers.

Much like the ownership line this value may have been interesting to examine by itself but is difficult to represent in one value for comparison to other values.

Meng et al.[25] examines ownership at a line level, making a distinction between shared lines of code that have been edited by multiple authors and non-shared lines with one author and compares the position and amount of shared lines with the position and amount of bugs. Examination of individual line level being outside the scope of the thesis was a reason for not choosing this representation. Measuring the amount and percentage of shared lines may have led to a useful value for comparison on both file and project level but was not done due to time constraints and concerns about prototype performance.

Fritz et al.[26] conducted a more detailed case study where authorship and interaction between developers and source code were measured. This study included detailed information about work patterns such as deliveries, acceptances and every individual interaction with the code. As this data was not possible to extract from a git repository and may in some cases require studying the actual work being done by developers it was deemed inappropriate for the scope of this thesis. For a more detailed case study such an approach may be useful but for examining a large quantity of sources it becomes unviable.

3.3 Measuring Collaboration: Network Analysis

As an alternative method of exploring representation of collaboration and internal developer structure a network model with nodes representing developers and edges representing bonds between developers was used.

3.3.1 Motivation

A motivation for using the social network approach was the wide variety of information that could potentially be extracted from it. As suggested in a paper by Hangal et al.[27] using similar methods with data from twitter and academic papers, a directed and weighted network modelling influence between people could be used for things like finding specifically influential ties or people or for finding an especially strong path.

Tenggren and Johansson's thesis on the subject[8] provided some more examples of what information could be extracted from a network graph built in a similar context. The main suggestions are Vertex Strength, Clustering Coefficient and three different versions of Centrality. Out of these the Clustering Coefficient appeared to have low effect by the results discussed by Tenggren and Johansson and was not chosen for consideration in this thesis. While measurments of Centrality might have been useful time constraints along with there being three definitions, some of which were unsupported by the NetworkX package led to Centrality not to be studied in detail. This left vertex strength which was extracted and saved for future use.

A drawback of the vertex strength is that it was shown on a developer level, making it unsuitable for comparison to values of other levels. To utilize the value a mean value along with a standard deviation and a maximum value was calculated.

3.3.2 Specification

The building of the network is based on the method presented by Orucevic-Alagic and Höst[13] with a weighted and directed network of developers. For every file used to build up the network the edge from a developer having contributed to a file to other developers having any contributions on the same file is increased by the percentage of contributions done by the first developer. This is repeated for every file of the right format in the repository. In the prototype this is done using the commit data for the selected time period. In a final network developers with considerable contribution in a large amount of files with several other contributors will have a larger amount of strong ties. Examining the network and its connectivity could also give some clues to the structure of the team.

In the prototype the network was built using Python's NetworkX package [28], which supports the desired network type and a number of operations to perform on on the built network. As in-data the same temporary files used for calculating the fractal value was used as all relevant data was stored there. A difference is that more specific information about specific developers can be obtained.

Aside from the standard network where every level of contribution on a file led to further influence coming in and out of the node of a contributor, a clean graph where only developers whose percentage of commits were over a specified and customizable threshold was built as a way to only model more significant collaborations. A result that was shown after some quick experimentation was that a threshold of 15% worked well in conjunction with the standard network as it struck a balance between not being too similar to the standard network while not throwing away too much useful data.

3.4 Complexity

When discussing "complexity" of code there are several different definitions that could come to mind. It could mean "difficult to understand" or "with multiple paths". While sometimes necessery depending on the problem which is implemented, complexity is generally a trait which is avoided. Complexity is a part of technical debt making code harder to maintain and extend, it is therefore chosen as a highlighted metric in this thesis research questions. Is there a relationship between the levels of collaboration and complexity. Does lone wolf programmers write more complex code, is complexity perhaps a ”too many cooks”-syndrome or does complexity stem from things other than levels of collaboration. To investigate further, models for complexity was chosen and the following subchapters will discuss what complexity measurments are used in this thesis and why these were chosen.

3.4.1 Motivation

As the main measure of complexity the cyclomatic complexity originally described by McCabe[29] was selected. It was chosen as it is one of the most common representations of complexity and has been thoroughly tested through years of usage. Furthermore, free tools for calculating the cyclomatic complexity or an approximation of it exists online.

While a pure measurement of complexity may not be directly related to performance. Previous research by Gill and Kemerer[30] suggests that combining cyclomatic complexity

with lines of code for a measurement of complexity density provided more noticeable results as there appeared to be correlation between level of maintainability and complexity density. Despite this there was interest in seeing if there was some connection between internal developer structure and complex code, making it an intriguing value to test against. As both cyclomatic complexity number and lines of code is observable obtaining a complexity density would have been possible but was not done due to mismatch with the collected data as complexity is looked at on a function level and would not have worked well as a comparison to the total lines of code in the file.

For extracting values relating to complexity the free tool Lizard[16] was used. This tool provided some noticeable drawbacks. First, it only measured complexity on a function level rather than a file or higher level execution level. It also ignored main methods in some languages such as Python leading to the value not being measurable in some smaller files. These considerations meant that the observable cyclomatic complexity number may be considerably lower than the actual number. It was however deemed good enough for an approximation of the complexity. The limitation to certain programming languages was not seen as a big obstacle as enough popular languages were supported to be generalizable for the used repositories and some limitation in examined files could help focus the results.

Aside from the cyclomatic complexity some simpler complexity metrics such as lines of code and function count may be of interest. These measurments are simpler and more accurate and may in some cases lead to more visible, if slightly trivial results. They may also be used in conjuction with the cyclomatic complexity, with lines of code as a way to measure a "Complexity Density" in future studies and with the number of function as a sort of sanity check where limitations of the used tools may be observed.

3.4.2 Specification

McCabe's Cyclomatic Complexity measurement of counts the number of linear independent paths that the execution of the code can take. In the case of imperative code with no decision points the cyclomatic complexity would be one. Decomposing a section of code to a control flow graph G with E edges, N nodes and P connected components gives the definition of cyclomatic complexity number $V(G) = E - N + 2P$. In figure 3.3 an example of a control flow graph for a program with a for-loop being followed by an if-then-else statement. With 9 edges, 8 nodes and 1 connected component the cyclomatic complexity would be $9-8+2*1 = 3$

The Lizard tool works on source code files for a subset of common programming languages. As exact measurements of cyclomatic complexity are difficult and may require the execution of code Lizard calculates an approximation based on parsing and does so on a function level. Aside from the cyclomatic complexity lines of code, function count and parameter count is counted and these values were saved as potential points of reference for the analysis. As these values are done at one specific point in time rather than over an interval the prototype looks at the complexity metrics for the date specified by the end of the interval. As the cyclomatic complexity was measured on a function level rather than a file level the value of the most complex function in a file was chosen as an approximation of the complexity of the file.

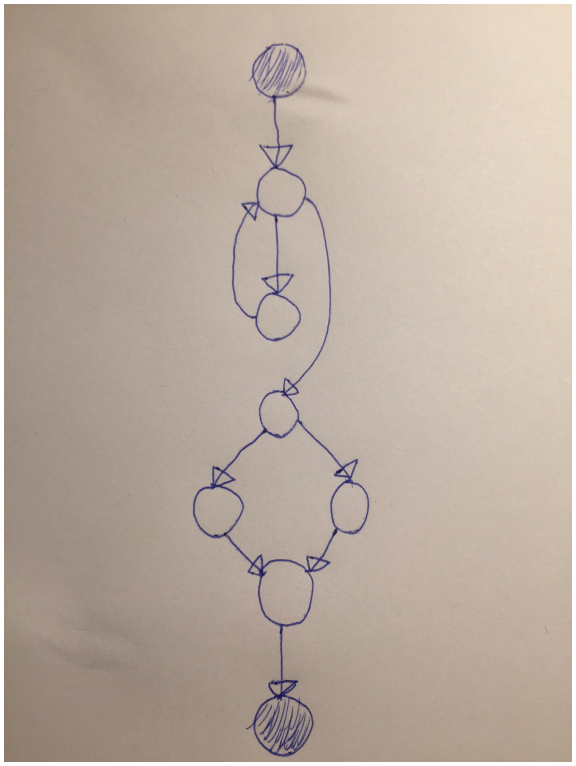


Figure 3.3: An Example of a control flow graph

3.4.3 Alternatives

An alternate method to measure complexity would be counting the number of indentations in a source code file. While this may have been simpler to extract it may also be less exact. While many programming languages and code standards utilize tabs as execution gets nested it is not universal and points of decision following each other rather than being nested would only be counted once. Despite some of the drawbacks of the Lizard tool the measurement of it seemed more thorough than merely counting indentations.

Another alternate measurment of complexity is the Halstead Complexity Measures[31]. These are a series of measures using operators and operands for several different values. While the multiple values could give a more complete picture than the singular cyclomatic complexity only one measurment of complexity felt neccessary for later stages and the cyclomatic complexity was chosen. Some disadvantages of using the Halstead Metrics in the context of this thesis is that the list of considered values for analysis was growing large to an extent that would provide an inconvenience during analysis and due to the fact that free tools used to extract the Halstead Measures tended to be language specific.

3.5 Integration Speed

To answer research question 2 it was a necessity to gather a throughput metric and integration speed was chosen for this for reasons that will be argued for in the following subchapter. Integrating often as opposed to collecting a technical debt and doing a big bang merge is becoming more and more common[32]. Kent Beck includes continuous integration as one of

the principles of eXtreme programming [3] and Martin Fowler vouches for continuous integration on its own merits [33]. Its widespread application and proven usefulness makes it an interesting metric to investigate and potential relationships to the measurments of collaboration could be explored. This subchapter will go into further motivation as to why integration speed is interesting to look at, how it is extracted for the thesis and prototype and lastly if there were any alternative ways which could be interesting.

3.5.1 Motivation

Nicole Forsgren et al.[4] has through their research identified factors based on speed as some of the most important in predicting success for a project. These include three key performance metrics primarily based on speed, lead time for changes, deployment frequency and time to restore service. Being fast gives positive qualities, such as time to market, the faster it is possible to deploy, the faster this new feature can be released to the market. Fast is also relative, but as Nicole Forsgren et al. found clear clusters of high performing teams, the projects with the highest performance could deploy multiple times a day.

These key performance metrics are interesting as finding other metrics with causalities against the key performance metrics can give rise to information about the usefulness of a practice. The key performance metrics can for the most part not be directly mined from git and need more data from outside sources such as issue trackers or code analysis. This thesis is restricted to data that is available in git in order to analyse a large set of open source repositories. Therefore it was not possible to reliably extract the key performance metrics from a arbitrary repository. Therefore a substitute for a throughput metric was generated, and will be argued to be a fair substitute in the following paragraph.

In order to be able to deploy fast there also need to be some sort of changes to deploy, or else it will not be a new deployment but a verification of the old build. To allow for deployment of anything new every day, which is possible for the highest performers, work also has to be integrated every day. Another concept, which Martin Fowler argues for in his blog is Continuous Integration[33]. Continuous Integration is a set of practices aimed for frequent integration the work which is becoming more and more commonplace. It has been proven by Michael Hilton et al. that projects following continuous integration and integrating often release more frequently than those that do not[32]. Integrating more frequently is therefore proven to have a relationship with deploying more often, which is one of the key performance indicators according to Nicole Forsgren et al.

Integration speed is an interesting metric to investigate as it has been proven that applying continuous integration leads to beneficial results, but also due to the connection with the key performance metrics of Forsgren, Humble and Kim.

3.5.2 Specification

Git has a referencing system in which every commit has a reference to the previous commit in the chain, its so called "parent". In the case of a merge a commit can reference several parents. By examining a specific commit there is no way to go forwards in time, as no reference to the children of a commit gets generated when a new commit is made. By walking through all parents there is no way of distinguish if a commit has been integrated or not as there is no information that states this in the metadata of the commit.

A method was developed along with Samuel Ytterbrink of Praqma (see Algorithm 1). The method was based on git's parent referencing system. Because the first parent of a commit will always refer to the commit parent in the branch which the merge is happening in, which in the case of only looking at the main branch will always be the main branch. With this, it is possible to utilize the ".." selection criteria. ".." in git allows for selecting all commits that exists in one tree structure which does not exist in a different specified tree structure. Figure 3.4, explains how the two tree structures gets selected. HEAD in the figure is pointing to a merge point. By following the commits of the first parent, which in git is referenced by "^1", the green dotted set of commits gets selected and by following the second parent, "^2", the red dotted set gets selected. By selecting the difference of the two sets a new set containing only the commits from the branch will be created.

The command `git log <commit-hash>^1..<commit-hash>^2` will therefore give the metadata of all commits that existed in the branch but does not exist in the main branch, which will be all the commits that were added to that branch and integrated to the main branch at the merge point. It will however have problems with recognizing commits that has been integrated through "rebasing" and if commit squash is done with a pull request. While it has some limitations it was deemed the best choice with more detailed explanation in the "alternatives" section.

```
Set v
foreach MergeCommit m do
    d ← Date(m)
    foreach commit c | c ∈ SecondParent(m) ∧ c ∉ FirstParent(m) do
        v.add(DateDiff(d, c))
    end
end
```

Algorithm 1: Pseudocode for extraction of integration speed

The integration speed can then be measured by looking at the commit metadata which is available in the form of commit date. The closer the date where the commit was made is to the merge point where it was integrated, the faster it was integrated.

Due to it being possible to also get information off which files changed in each commit, the age of the commits which forms our integration speed can be connected with the files integrated. Making it possible to extract the integration speed on a file level, where analysis of why certain files takes longer or shorter to integrate and why that might be can be investigated. The data can also be processed into the project level, where the commit speed is aggregated for all integrations in a given time period. That allows for doing trend analysis on a specific project as data on integration speed for each time interval in the project can be done.

3.5.3 Alternatives

Other methods for analysing if a commit had been integrated or not were considered. An idea was to create a new data structure and parse through all commits to build up a new tree containing the correct data that was required. While such a solution would have yielded in precise data specifically because it could have been implemented to trace commits which has been merged by the "rebase" functionality git has, which employs a series of commits from one branch on top of another branch instead of the standard "merge". Because of time and

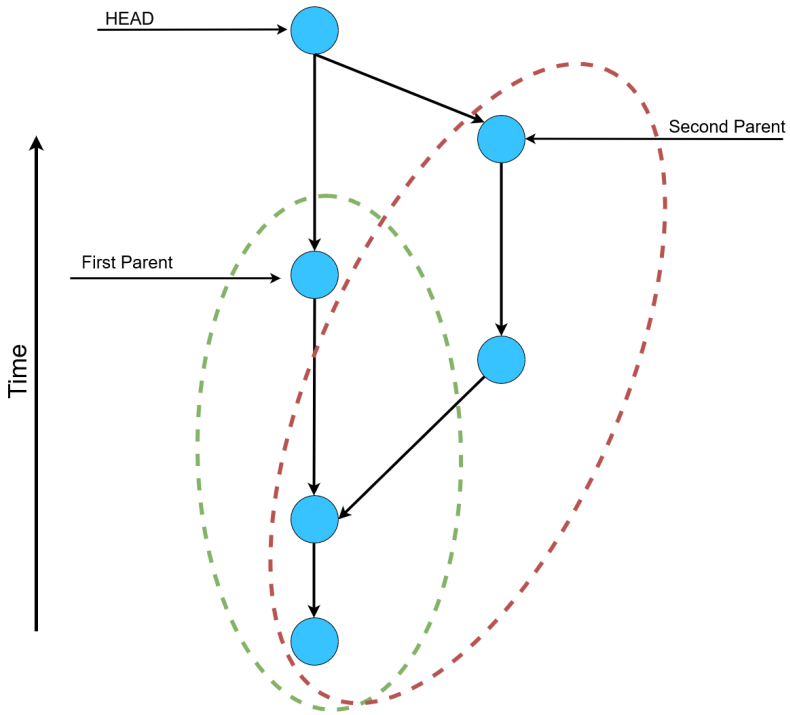


Figure 3.4: The two different sets of commits (blue balls) which comes from following either the first of second parent

scope such a method of commit analysis was not chosen, as it would require to reimplement a big part of git and also extend its functionality.

The simpler version described in the previous section relies on the first and second parent in a merge point, it works well for the standard merge case, but git allows for different ways to circumvent this system. Mainly it is circumvented by the re-base system but also in the case of a pull-request where the commits have the option to be squashed, creating one new commit and merging that commit in. Because of these cases it is not entirely accurate, but as a compromise was needed to be made, as we didn't have the time to reimplement a new tree structure for git it was deemed sufficient for analysis.

Alternatives to integration speed as a metric itself was also considered, the different throughput metrics described in Nicole Forsgren et al. [4] where all pursued. But as they would require access to more information available in for instance issue trackers it was decided to keep the prototype general and applicable to any open source repository using git. There are a multitude of issue trackers available, in order to get the data reliably then it would be required to gain access to the issue tracker in use and have an implementation for that issue trackers api ready. Even if access to an issue tracker is in granted it is also possible for developers in the project to not filling in the information needed. The integration speed metric is interesting on its own while also having a strong connection to the dora established metrics.

3.6 Visual Representation

The visual representation is built on scripts based on the R language. Specifically the library ”readr” for importing data on a CSV format [34], ”ggplot2” for plotting of points with statistical aids such as confidence intervals[35], correlation coefficients and regressions. The library ”plotly” was used in order to make the plots interactive [36].

The packages all pertain to how the data is visualised. When plotting file data the amount of data points can exceed 10 000 for big repositories, plotting such an amount of data generated more requirements on the plotting tool. In order to generate the highest quality information, the plotting tool had to be interactive because statically showing thousands of labels in a plot makes it crowded and hard to analyse as text and points are overlapping. This was achieved with the ”plotly”-library which also made it possible tosave the plots as generated html-files which opens up for the plots to be a part of a webpage and automatically update.

The ”ggplot2” package supports the plot generation and includes many features which are useful for analysis.[35] It is easy to make the plots contain more information than the standard x,y point pairings as you can add opacity, size or colour to the points based on any of the fields. That is excellent for hotspot analysis, trying to find which files where more care is needed to be taken, for example Adam Tornhills hotspot definition depends on size in lines of code, and change frequency or commits[11], both of these values can be put into the ”ggplot2” plotting function as colour intensity and size.

For statistical analysis ”ggplot2” has functionality for plotting out regression lines, histograms, boxplots displaying the median and quantiles among other functions that were not used in the thesis. The regression functions which are of most importance as analysis will partly be based on them uses the statistical package or R which is developed by the core R team and also a part of the core of the R language[37]. R is widely used by professional statisticians and data analysts who rely on the correctness of these statistical packaged every day[38].

Chapter 4
Results

The prototype, with its implementation of the models described in the previous chapter, has been applied to the collected set of repositories. The application has for different parts been done on either the full set of 107 projects as a way to compare output from a larger set of data or on one project for a more focused lower level analysis. Because of an interest in making comparisons across levels such as file or project level without major differences in the structure of examined projects the analysis done on only one project was done with a more unified case study on only one project being selected. The Gerrit Code Review repository was chosen for this because of its constant level of activity over recent years and an unfortunately lost opportunity of discussions with people from within the project being a consideration in earlier phases. The models and prototype have been used to gather the results which the analysis will be based on. The results of the prototypes extraction will be presented in this section.

The visualisations will display the metrics defined by the models in the design chapter, but other relevant metrics are also explored, those include amount of commiters, lines of code, number of functions. Because of how some metrics are better explored in a specific level, the metrics have been gathered on different levels, per file, per developer and per project. These different levels of granularity are explored, more details will be in the section and in the discussion relating to it.

The git metadata was dumped into files, which in this thesis is called the ”raw data”, this raw data was processed and refined which resulted in the ”processed data”, the processed data was compiled and through visualization techniques, it will be formed into ”information”. Of course information can be extracted from any point, from the metadata or raw data to the visualizations, but with every step, the data transforms into higher level information. To gain information from the raw data, high domain knowledge and experience with that context is required, but as the data becomes processed, compiled and visualized the information becomes accessible and understandable by people who do not have a deep domain knowledge.

Results that relate to the previously presented research questions through both validation and comparison are prioritized but other intriguing results are also shown. Presenting varied

data the structure should be such that individual readers may find what relates to their specific interests. The result will be the basis upon with the discussion is based on, observations will be made from the presented data and will be referred to in the discussion. It can be of interest if the reader wishes to verify the observations made in the discussion, or to spark the imagination of what results can be extracted by the prototype and what potentially could be extracted and used in their context.

Throughout the chapter several different values are utilized in graphs and discussions. For more detail of how and why these particular values were selected the previously appearing Design chapter may be of interest as the results of chapter builds on the discussed models and values of the previous chapter.

The chapter will start with some graphs and discussions relating to the lower levels that were examined, namely the developer level and file level explorations. After that the higher level exploration done on a project level will be shown and discussed. In the penultimate subchapter an alternative method of analysis comparing different timespans on the same project will be the focal point. Finally there will be a short summary of the results of the chapter.

4.1 Developer Level

Examination on a developer level served as a lower level of examination alternative to the file level. On this level several aspects of the network representation from section 3.3 could be examined, serving as a method of continued exploration of the representation of collaboration and developer structure central to research question 1. However, due to some concerns about how examining these values on this level could lead to excessive scapegoating, examination of the complexity and throughput metrics important to the comparisons of research question 2 was not examined at this level. Furthermore, because of how tools only measured cyclomatic complexity on a function or file level and could be difficult to link to specific developers without analysis that was considered out of the scope of this thesis it was not measured on this level.

4.1.1 Network Analysis Output Excerpts

Most of the output on this level was not used in plots due to the lower amount of considered values not being able to lead to good comparison. It was presented through output in the terminal and some excerpts examplifying what how this data appears will be presented. These were generated using the data from the gerrit project using different intervals for comparisons to examine how the network works on different kinds of timespans. Figures 4.1 through 4.3 features the vertex strengths, serving as a way to measure the level of collaboration of a developer, for some of the most significant developers while figure 4.4 shows the strongest ties in the network for one developer, showing who someone have collaborated with, in this example the strongest ties of David Pursehouse is shown.

In figures 4.1, 4.2 and 4.3 the vertex strength and how it varies based on interval length can be shown. Someone with a high vertex strength could be identified as the most influential developer. Increasing the timespan gives higher values as more activity is utilized in the calculation of the values. A potential conclusion for this is that it may be more accurate

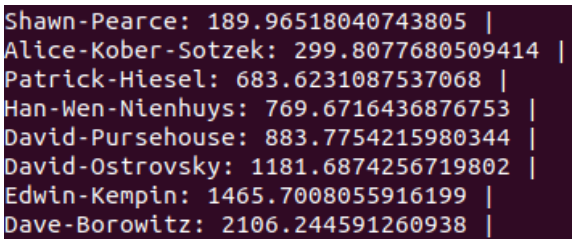


Figure 4.1: Some Vertex Strength levels for developers in the gerrit project between June 2017 and June 2019

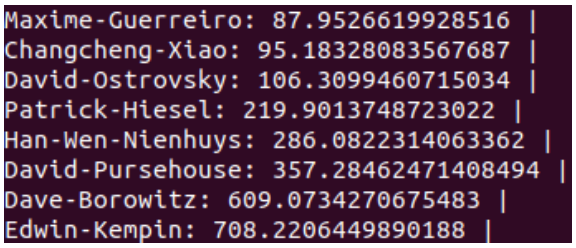


Figure 4.2: Some Vertex Strength levels for developers in the gerrit project between June 2018 and June 2019

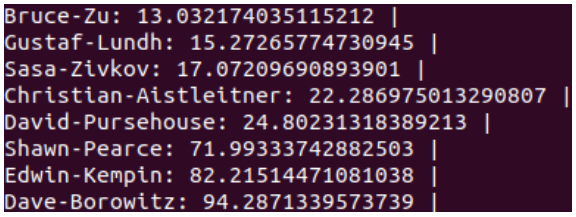


Figure 4.3: Some Vertex Strength levels for developers in the gerrit project between February 2013 and June 2013

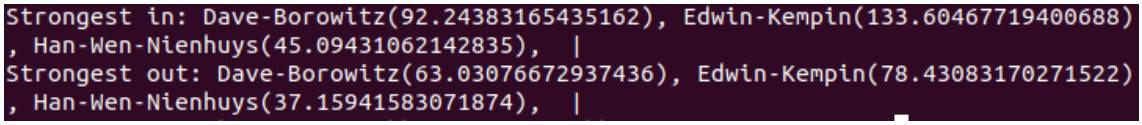


Figure 4.4: The Strongest Ties for one developer in the gerrit project between 2018 and 2019

if looked as as a comparison between developers for just one project and interval as it may otherwise widely vary without it giving more information than the amount of activity.

The strongest ties shown in figure 4.4 could be used for something like finding ideas for who someone often works with. The in-ties in this excerpt are high for developers having high contribution of files the specifically examined developer works in and the out-ties are high for developers working in files where the specifically examined developer has high contribution. Looking at both of these ties separately may be redundant as it in this and several other observed cases linked to the same people. There also exist a tendency of this functionality to favor the most influental developers as can be seen when comparing to figure 4.2. While this is to be expected it may be to such an extent that it is not an accurate way to find internal teams.

4.2 File Level

The main lower level exploration was done on a file level. The representation of collaboration of research question 1 is something that was shown to be better represented on lower levels so large parts of the validation of selected models such as the fractal value explained in the Design chapter was done on this level. Some of the comparisons necessary for exploring Research Question 2 can also be performed at this level as the complexity metrics in the previous chapter are file-based and the integration speed can be examined on a file level.

4.2.1 Graphs

Because the fractal value is a continuous value that can be compared to other values scatterplot graphs for those has been chosen. In the scatterplot graphs a point represents a file in one project and the x- and y-axis is two values that are plotted against each other. A line showing potential correlation through regression or interpolation is also shown in the graphs alongside its confidence interval. These lines can aid in identifying potential correlations, as well as suggesting how trustworthy they are based on the datapoints. For the file level graphs the Locally Estimated Scatterplot Smoothing (LOESS) which is a regression that allows for curved lines was used.

When displaying graphs with a discrete x axis, barplots were the chosen method. Because scatterplots does not easily display additional information as of where the median is and how the spread is distributed barplots were used in figure 4.8, 4.9 and 4.10. The barplots in this section works the following; a white rectangle displays the interquartile range, that means the range from the first quartile to the third, which is the centermost 50%; inside the white rectangle there is a black line, marking the median; the whiskers that span out from the white rectangle with a length of 1.5 times the interquartile range, points outside of this range will show up as scatterplot points and can be treated as outliers in the data. The choice of 1.5 as coefficient for the whiskers is based on the standard setting of the R statistical package [37].

Because the need to reduce biases and get data which was spread from files with different amount of developers, the processed data chosen for the plots have been selected as a randomized subset from all the processed data from all files of the 107 repositories. If only one projects files was chosen it could be that potential relationships seen could be affected by biases, affected by some context specific to that repository and a general study was seen

as preferable to a case study given the circumstances. From the 107 repositories 10 000 files were selected randomly with the `shuf` tool from the GNU Core Utilities, extracting random lines from a compiled file containing the processed data of all files. The reason for picking 10000 files was that it was the highest which would not make the computers used in this thesis crash, but it is also deemed sufficiently large in order to not have the data be biased towards one repository while also getting a spread of data representative of the entire data set.

Because the repositories contains different amount of files this selection method is inherently biased towards repositories containing more files as those repositories files are more likely to be selected than repositories with lower amount of files. Instead of randomly selecting, it would be possible to categorize the files and repositories to try and find results regarding specific types of projects or files, but such in depth analysis was outside of the scope for this thesis. From figure 4.14 it is possible to see the there are 70 repositories with between 0-1000 files, 34 repositories with between 1000-7500 files and 13 repositories with between 7500-20000 files. Files from the 13 repositories are therefore roughly represents 50% of the file data with the other roughly 50% piece is coming from the other two categories.

4.2.2 Ownership Model Verification

Because this thesis ownership model is built on commits which was argued for with design specifications and previous literature models regarding added lines were disregarded. To validate the choices made figure 4.5 was generated to display the differences between commits and added lines. The LOESS fitting produced a clear line pointing towards a strong correlation, the more changes to a file there is the more added lines are also included. But what is also clearly visible is that there contains a lot of outliers, especially when there is a few amount of commits. The reason for this can be how git records added lines when a file is renamed or as a result of auto-generated files it could also be the result of a ”git squash” compiling multiple commits into one. Another reason could be the inclusion of other open source libraries which might be added to the repository.

To better understand the other plots relating to files it can be useful to understand the distribution of developers in the files analysed which is plotted in figure 4.6. From the 10000 random samples around 50% where developed by only one developer. That means that the majority of files in open source is developed by only one single developer. Because files can be of any varying size and importance to the main logic of the program you can not from the graph tell if 50% of the work is done isolated in the files. From the processed data prepared by the prototype it would be possible to explore this question but due to scope it was omitted but could be interesting for future research.

4.2.3 Performance metrics

While no universal connection between the fractal value and integration speed or complexity is directly observable throguh figure 4.7, files with a fractal value of zero, meaning only one developer stretches across both extremes for the comparing values. While the median is lower for one developer the amount of outliers seems to be higher. Causes of the high amount of outliers for one developer could be the higher amount of data points in this column as about half the data set contained files with only one developer as seen in figure 4.6.

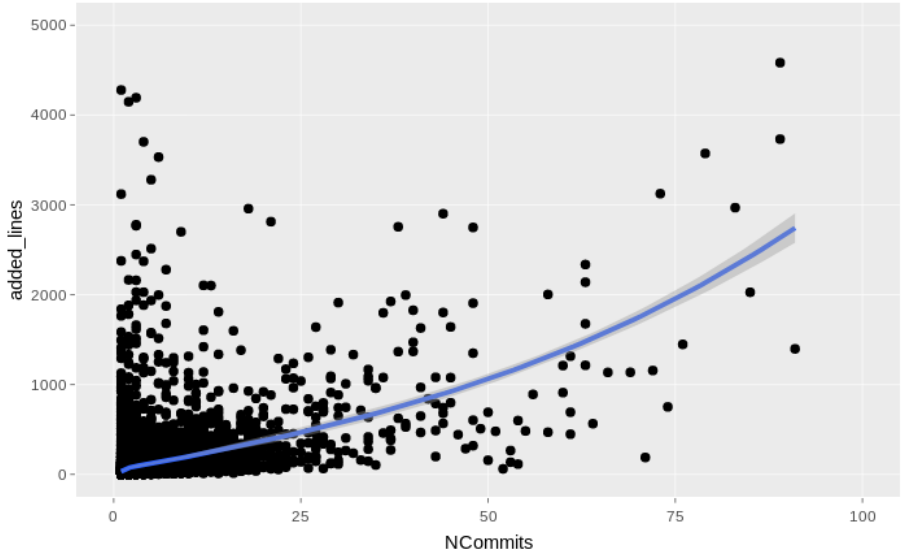


Figure 4.5: The number of commits against the amount of added lines

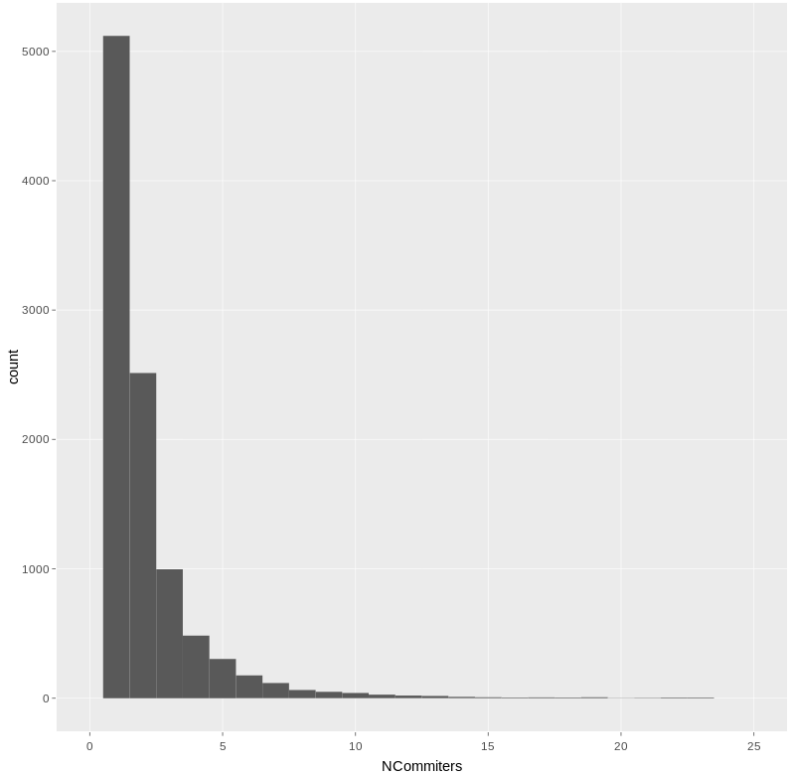
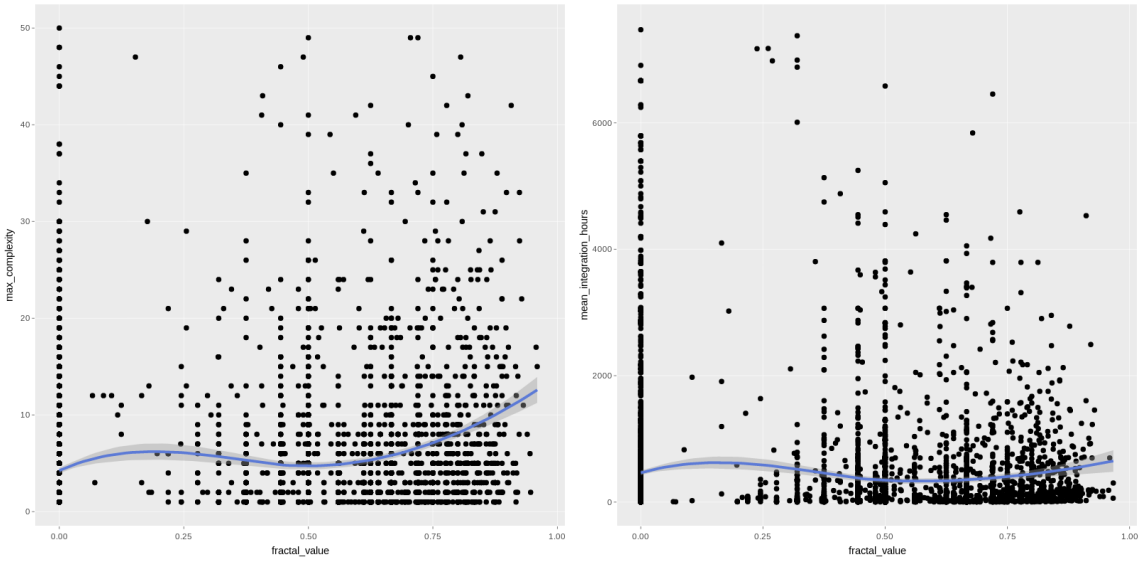


Figure 4.6: The distribution of the amount of developers on a file



(a) The fractal value of the file against the cyclomatic complexity

(b) The files fractal value against the integration speed

Figure 4.7

From figure 4.8 it seems that an increase in developers working on a file does impact the max complexity of the file. The median is and interquartile range is is similar when there are between one-five developers but as the amount of developers gets higher, so does the median and it is followed by the LOESS fitting line. When looking at figure 4.9 which plots the the commiters against the average complexity of all functions in a file it is seen that the median and interquartile range is stable and around the same between 1 to 15 developers. That means that while the average complexity of the functions in one file remains the same regardless of number of developers, but having more developers makes it more likely that one function within the file has a higher complexity.

From figure 4.10 which analyzes the integration speed and its relationship with number of developers it is shown that there being between 1 to 10 developers on a file leads to similar medians and interquartile ranges. The speed therefore does not seem to be impacted by amount of developers. The LOESS fitting points to that the integration speed is lower for single developers than compared to two developers. This can also be seen in the fractal value plot (see figure 4.7b) where the fitting predicts that fractal values between 0.5 to 0.75 has the fastest integration speed. For integration speed it seems that collaborating is beneficial compared single developers.

4.3 Project Level

Examining at a project or repository level is a higher level exploration complementing the lower level ones. While the presented information may be less exact than corresponding lower level values a wider variety of data could be utilized. For example the network representation otherwise where values otherwise were examined on a developer level could be examined on the project level, allowing for alternate methods of exploring both research questions.

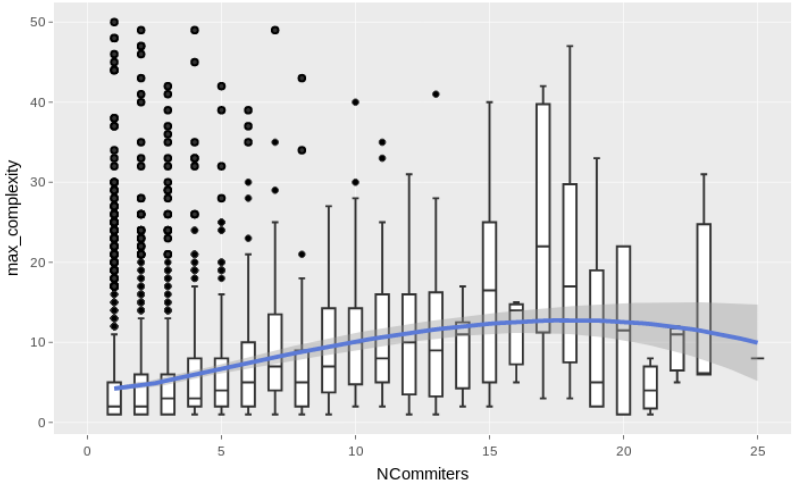


Figure 4.8: The amount of commiters to the file against the max cyclomatic complexity for a single function

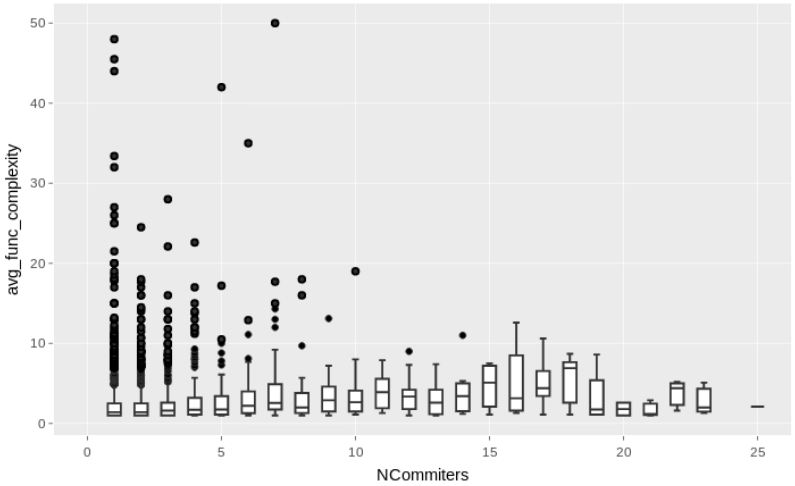


Figure 4.9: The amount of commiters to the file against the average function cyclomatic complexity

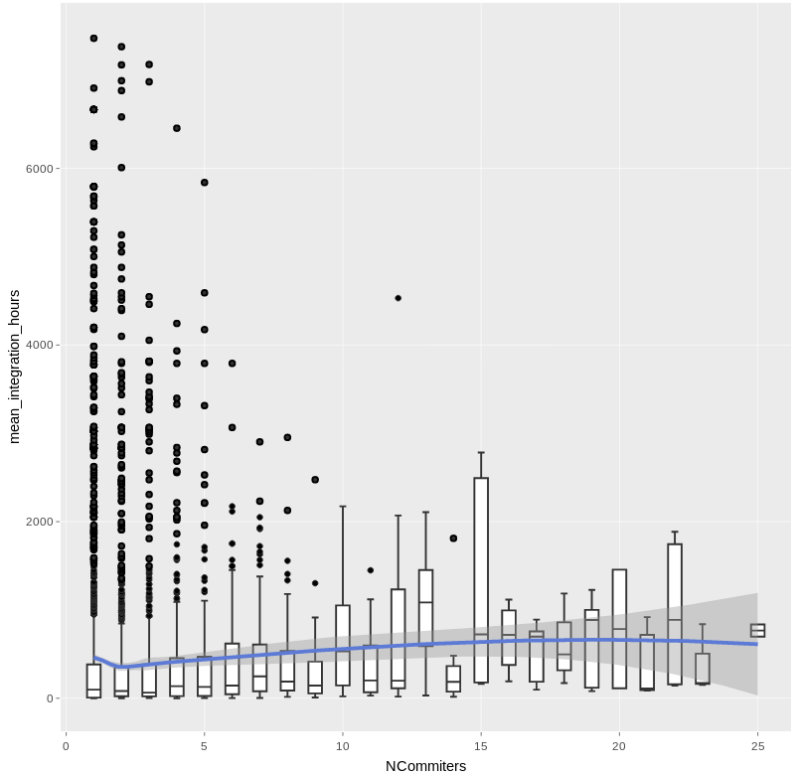
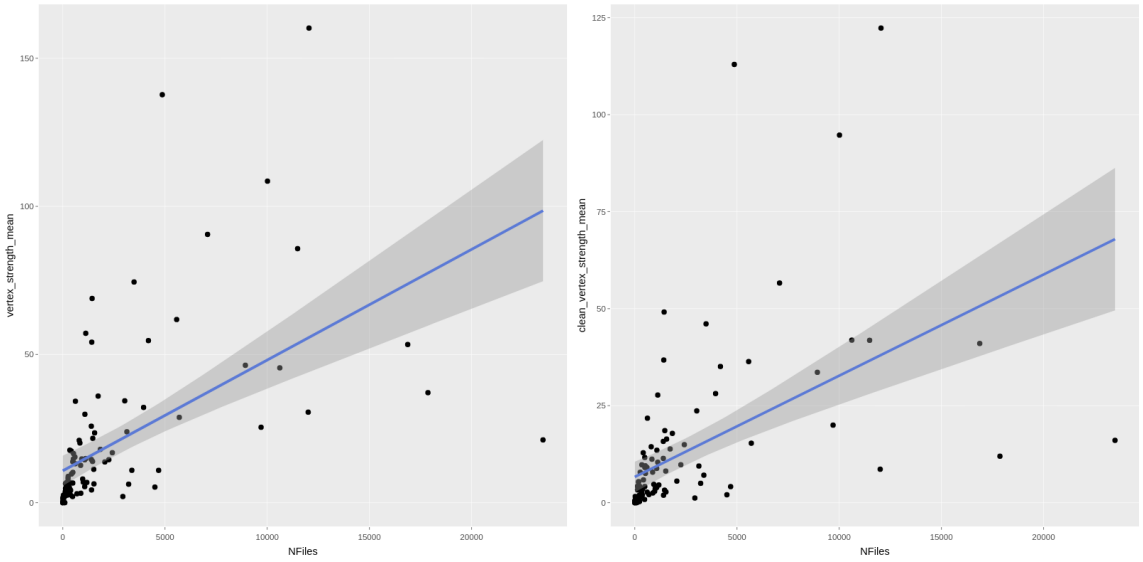


Figure 4.10: The amount of commiters to the file against the integration hours



(a) The mean vertex strength plotted against the number of files for 107 projects between 2017 and 2019

(b) The mean vertex strength of the cleaned graph plotted against the number of files for 107 projects between 2017 and 2019

Figure 4.11

4.3.1 Graphs

The graphs of the project level can be seen as scatterplots where every point represents one project and two values are plotted against each other. Like in the similar graphs for the file level there also exists a line showing potential correlation through regression with a confidence interval. For the project level a linear regression with a straight line was utilized when deemed necessary.

In figure 4.11 the number of files and vertex strength should be dependant due to definition but not enough so that the model appears primarily based on this. The values appear correlated based on the images. This makes sense given the definition but some concern over its utility on the level may arise due to being to influenced by this aspect. Despite the dependency large fluctuation still appears to be present and examining the outliers could lead to further results.

Finding some connection between a representation of developer structure and integration speed would give clues of how to optimize the developer structure. However, the fractal value and the integration hours appear to be unrelated as seen in figure 4.12. Two possible reasons for this is that the aspects that are modeled by the fractal value is completely unrelated to throughput, or that the lost detail of project level leads to correlations not being observable with the model.

The comparison of vertex strength of figure 4.13 and integration hours have similar motives and results to the plot in 4.12 where connections could not be proven. A difference is that few projects of larger vertex strength were found so there may be some connections for such projects that need further data for proof. If correlations between number of files and integration speed were found it could show that repository size can affect throughput. Again, no clear correlation could be observed in figure 4.14, showing that a connection here

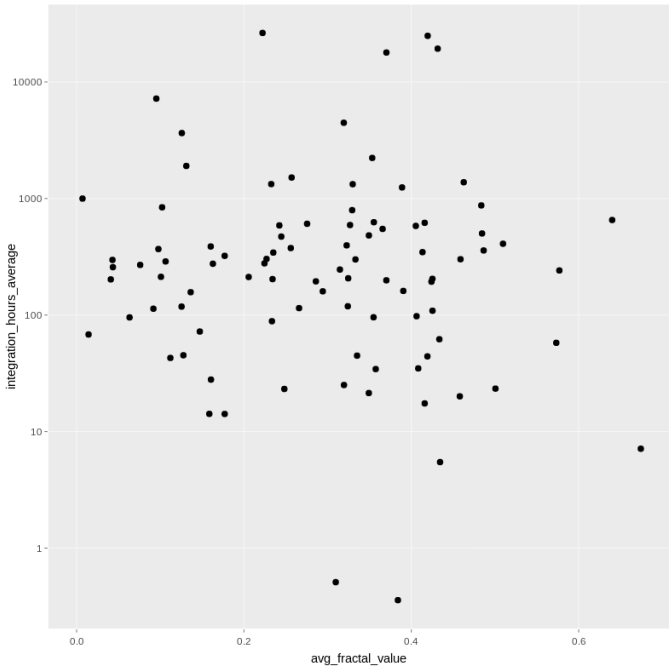


Figure 4.12: The mean fractal value against the mean integration hours for 107 projects between 2017 and 2019

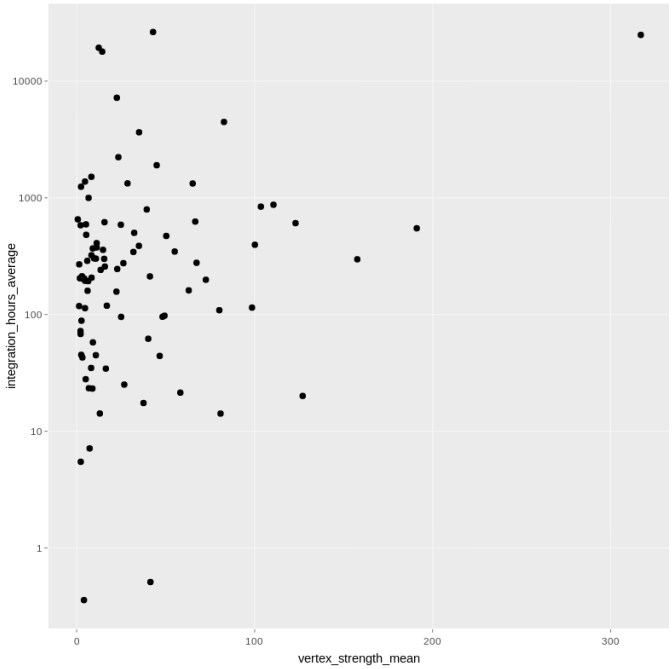


Figure 4.13: The mean vertex strength plotted against the mean integration hours for 107 projects between 2017 and 2019

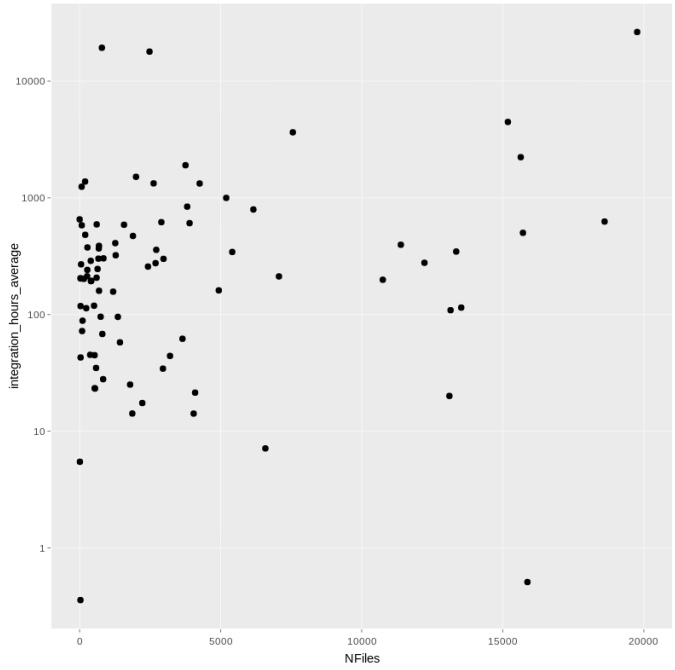


Figure 4.14: The mean integration hours plotted against the number of files for 107 projects between 2017 and 2019

is unlikely.

If connections were found between complexity and fractal value on a project level it would mean that developer structure could affect code practices which could help with further decisions in the field. While analysis of figure 4.15 there may be a small correlation the confidence interval shows that it is statistically insignificant and should not be assumed to exist at this point. Mean values of complexity may also not have a huge significance as larger amounts small files will decrease it despite complex files existing in the project.

Like the analysis done in figure 4.15 the comparison of vertex strength and complexity in figure 4.16 could show the connections between correlation or developer structure and complexity. While no clear correlation can be observed one interesting detail is the fact that even though most projects have both low vertex strength and low complexity the projects with very high mean complexity also have low mean vertex strength, meaning low levels of collaboration. A reason for this could be that some developers write more complex code when being the only developer on a file.

4.4 Trends over Intervals

Aside from examining the results on several files or several projects, following the trends of one project served as an alternate method of analysis where how a value changes over time can be examined. With more proof of the effects of certain values this kind of analysis could be utilized in the industry where negative and positive trends could be observed and dealt with accordingly.

Because of the more limited datasets and simpler plots, finding correlations is somewhat more difficult using this method but further validation of selected models can be obtained

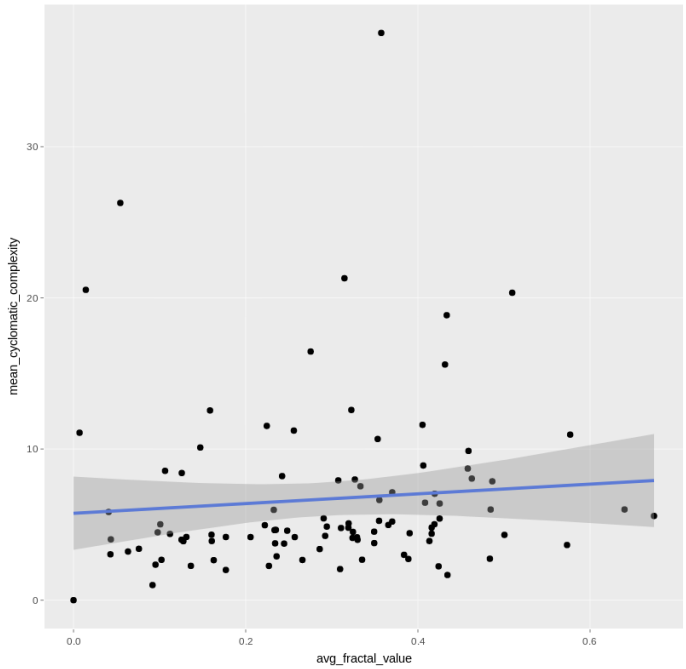


Figure 4.15: The mean fractal value plotted against the mean cyclomatic complexity for 107 projects between 2017 and 2019

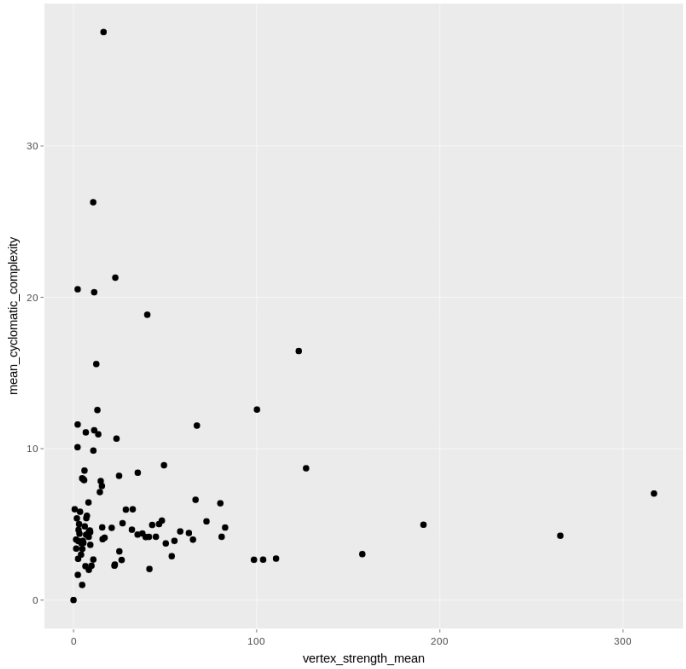


Figure 4.16: The mean vertex strength plotted against the mean cyclomatic complexity for 107 projects between 2017 and 2019

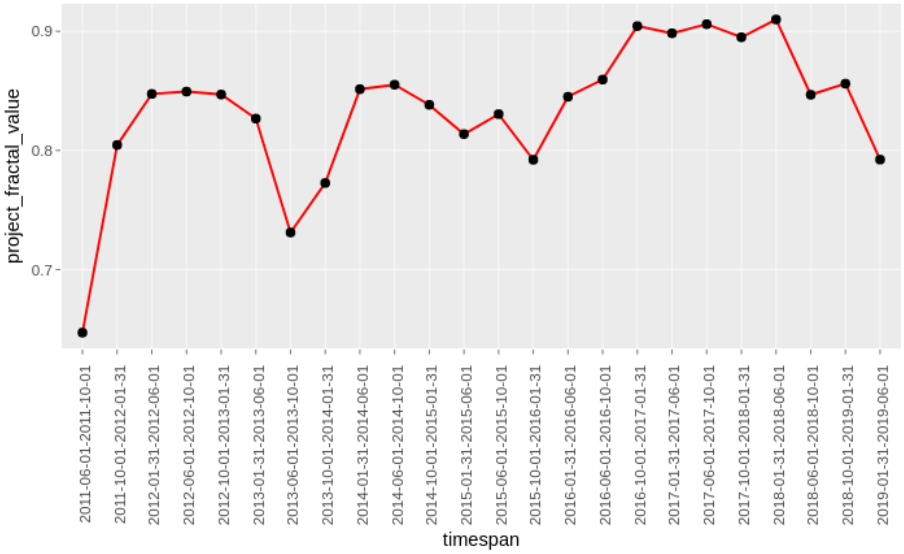


Figure 4.17: Fractal value for the full gerrit project

by the examination of how they vary between equal and equally spaced timespans. This can be used to further help validation of suggested models for measuring collaboration, as well as those utilized for checking the complexity and throughput.

4.4.1 Plots

Over an interval line plots were possible to produce. These follow one project over a period of time with equally spaced and equally long intervals with the line showing an interval to interval change of the value. The line then follows one value at a time. While it would have been possible to impose different values in the same image as the scales often were very differing it was chosen to be limited to one with an accurate scale on the y axis. Overly short intervals such as 1 month was not preferable due to the low amount of data being collectable for such a short span and overly long intervals requires projects with longer histories to allow for analysis between the timespans, timespans between 3 and 6 months generally worked well for this visualization. The graphs in this section all follow the Gerrit project between June 2011 and June 2019 with an interval length of four months. The project fractal value shown in figure 4.17 can work as an overall measurement of how people worked on the project in each time period. When lower such as in the first interval one developer have done a large amount of work and when higher the work is distributed between several. Observing fluctuations the changes can be observed.

The mean fractal value from figure 4.18 instead looks at a file level and gives lower values. While the utilization of all edited files led to a hypothesis of lower fluctuations but the value still fluctuate about as much as the project fractal value in this example. This may represent phases where components that require several developers are changed.

Utilizing the fractal values an approximation of the developer distribution and how it varies from month to month can be observed. The average value over every file and the full project value led to very different results. Both of these representations appear to have flaws as the project value is an overview with lack of detail and the mean over files loses most detail

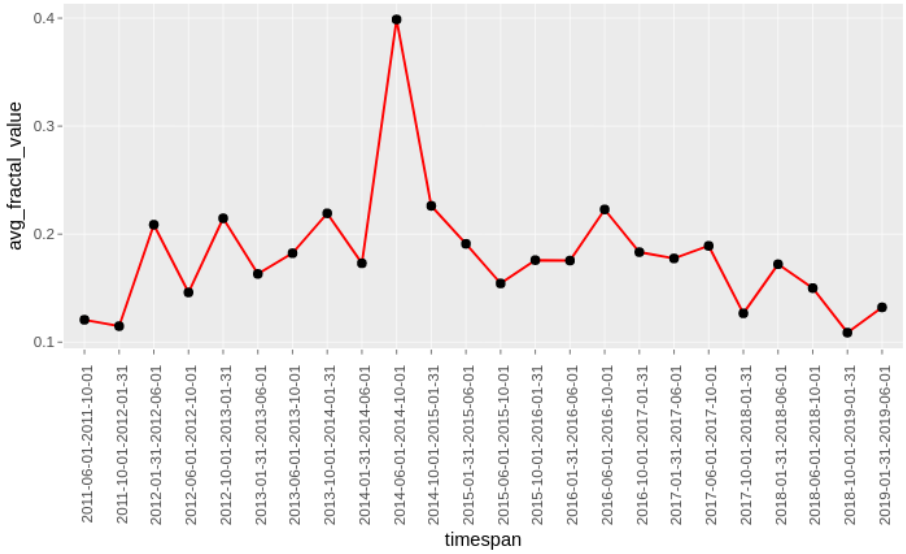
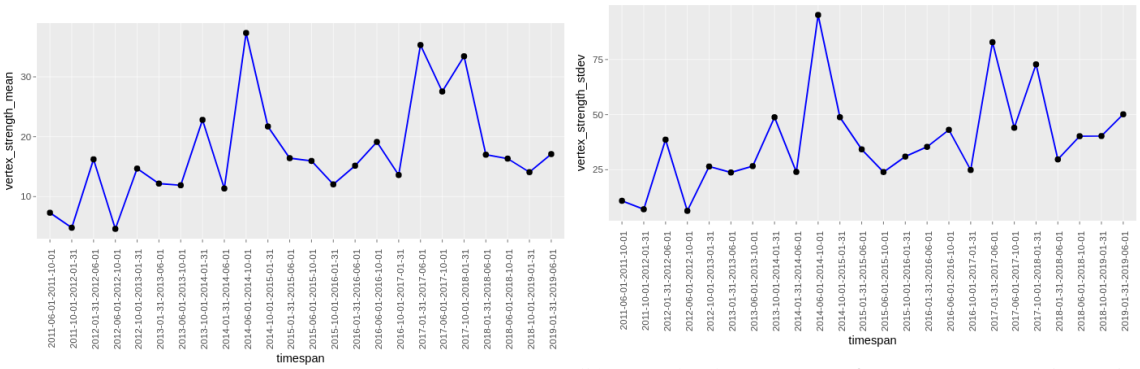


Figure 4.18: Mean fractal value of individual files in the gerrit project



(a) Mean vertex strength in the gerrit project

(b) Standard Deviation for vertex strength in the gerrit project

Figure 4.19

in the conversion.

The mean vertex strength seen in figure 4.19a should have similarities to the fractal value but with a bigger focus on collaboration. With some shape similarities to mean fractal value this hypothesis holds. The months where shape differ could be examined in more detail. These intervals are where collaboration is higher but contributions are weighted towards a lower amount of developers.

With constant distribution from zero the standard deviation and the mean value of the vertex strength both shown in figure 4.19 follow very similar distributions and examining both is probably unnecessary.

The added lines seen in figure 4.20 shows a level of activity per interval that could be compared to others such the amount of commits and commiters to find something like average commit size. While not completely trustworthy as some lines may be automatically generated in bulk or taken from elsewhere but periods of larger additions could be examined in detail.

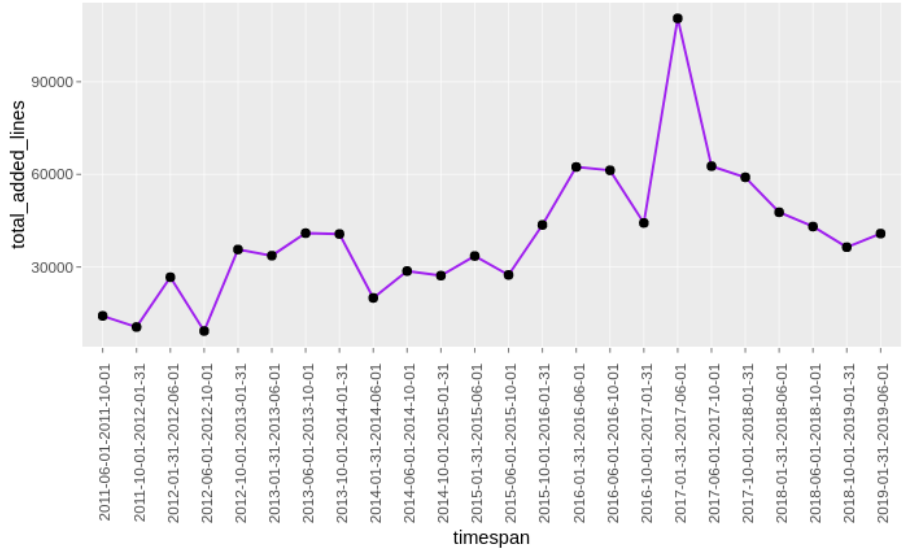


Figure 4.20: Added lines to the gerrit project

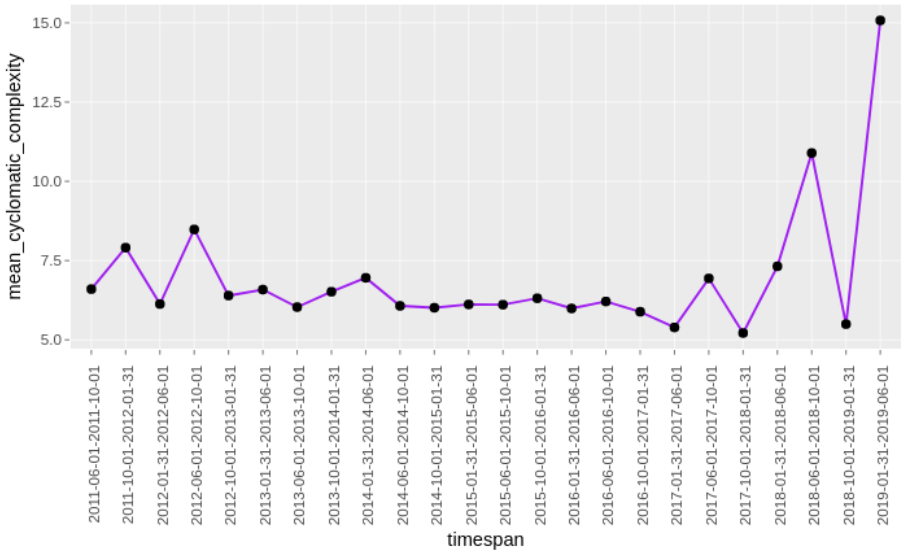


Figure 4.21: Cyclomatic Complexity mean of the gerrit project

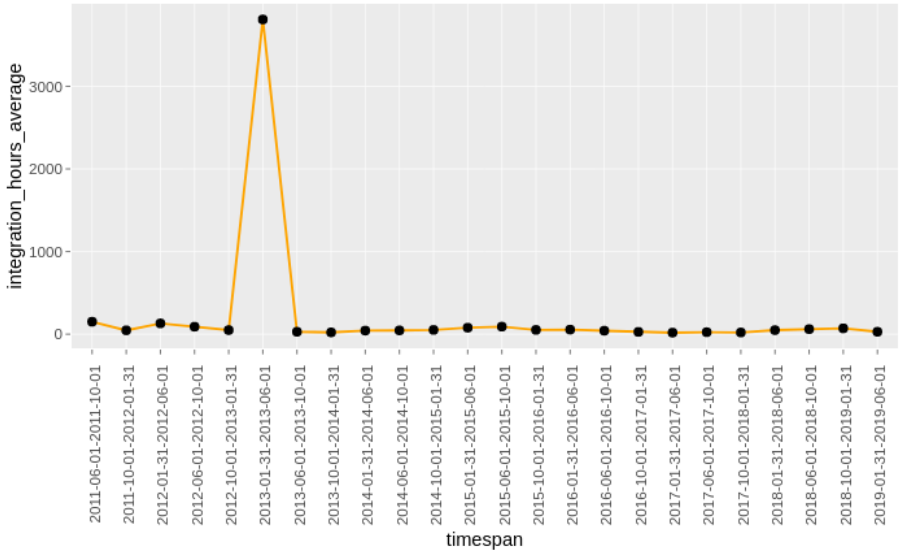


Figure 4.22: Mean hours to integration in the gerrit project

While examining complexity may be interesting there are some flaws with doing it in this context as cyclomatic complexity only measures an average of files that have been edited in the timespan. Still, from figure 4.21 it can be observed when the more complex files are toched which could be compared to other values.

The exploration of figure 4.22 showing mean integration hours of the examined project showed a considerable slower integration in the June 2013 period, the details of this merge was not examined further but could be searched for in case of interest. Similar spikes seemed somewhat common in other examination of this value.

The maximum integration hours of figure 4.23 showed spikes not visible in the mean value of figure 4.22 such as the October 2018 interval. This would point to the slow integration only being a small part of the full integration. Further examination of the details may be interesting but was not done due to being outside the scope of the research of this thesis.

The number of integrated commits in figure 4.24 roughly matching the number of total commits in figure 4.25 helps validate the examination of integration hours as a significant part of files are used for this value. In cases where the number of integrated commits are considerably lower integrations are rarely used in the projects, leading to the integration time being less accurate and important.

The number of commits is another baseline value for looking at the level of activity of a project and was used as the main measurement for this in several used models of this thesis. By analysing the value in figure 4.25, high- and low-activity intervals could be identified. The number of commiters is another baseline value and there should usually (but not always) be rough similarities to the number of commits. This can be seen by comparing figure 4.26 and figure 4.25, comparison between the values in the figures could also give an idea of the the average amount of commits per commiter.

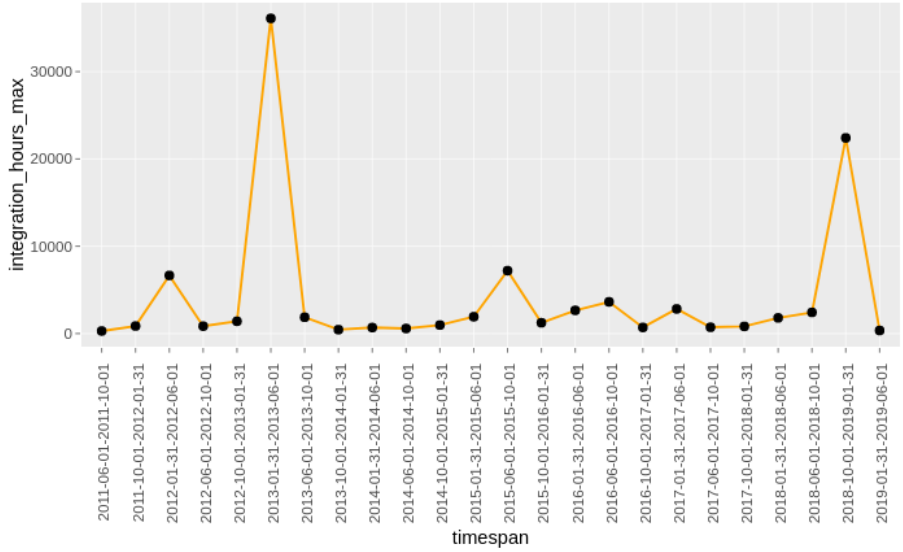


Figure 4.23: Maximum hours to integration in the gerrit project

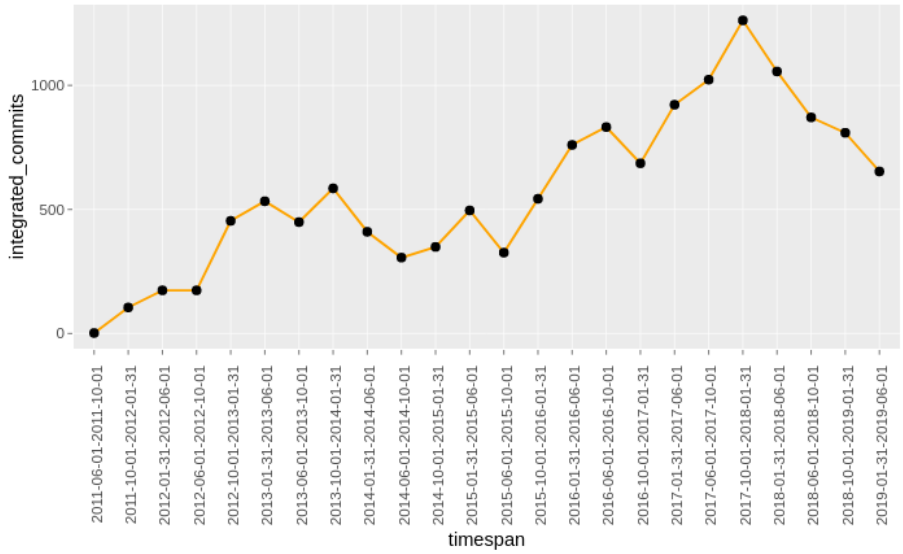


Figure 4.24: Number of integrated commits in the gerrit project

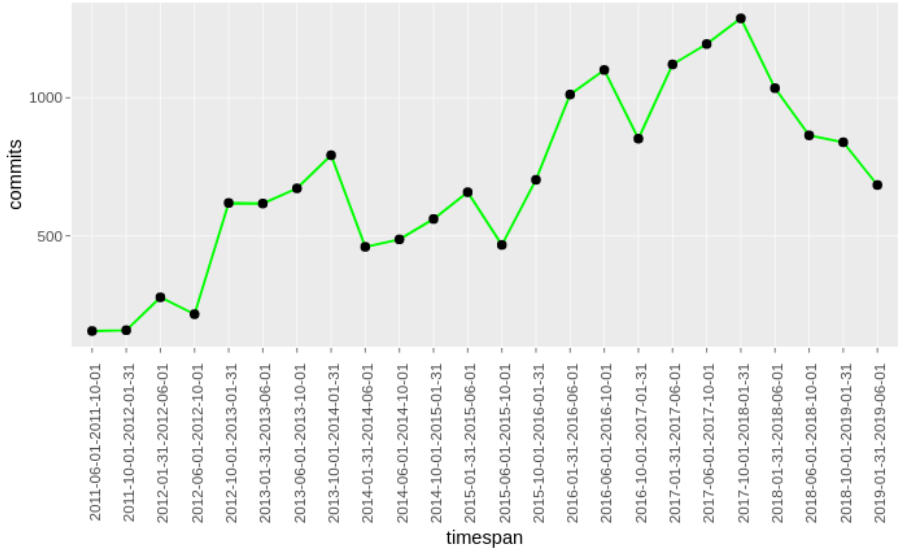


Figure 4.25: Number of commits of the gerrit project

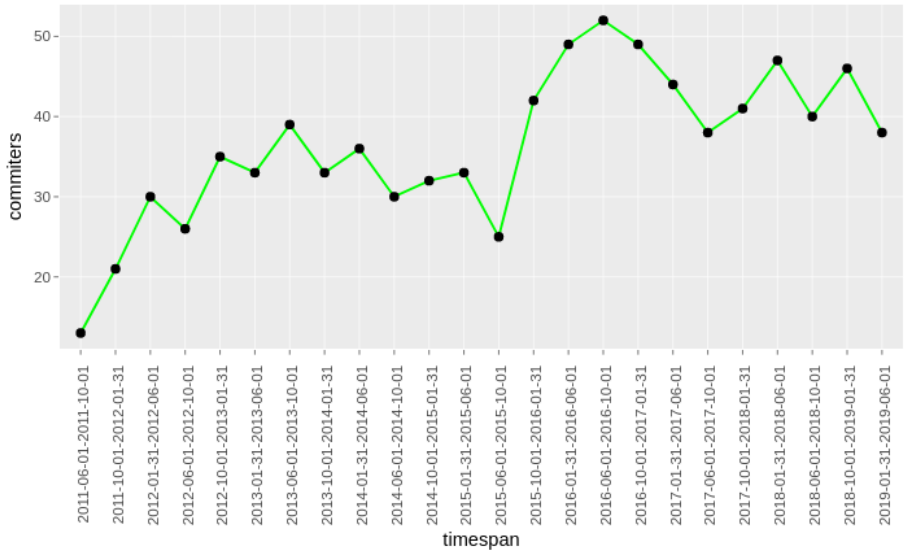


Figure 4.26: Number of commiters of the gerrit project

4.5 Summary and Conclusions

The results obtained from the thesis have been examined on different levels, on lower levels such as developer and file level results could be examined on a more detailed level while on higher levels such as the project or interval levels a more general level of analysis could be performed. While only limited analysis was done on developer level a method of finding the most influential developer as well as an attempt to identify teams.

On the project level no specifically interesting and statistically significant correlations could be found. This could mean that there is no connection between the examined metrics but because of an overall lack of any connection for certain values a likely reason is that many of the models and values works badly on the project level. A reason for this could be that many of them are mean values where much of the detail is lost in the conversion of the original values.

While not many new insight could be obtained from examination of the interval graphs some insight in how the project level metrics relate could be discussed. This kind of way of analyzing a project may also be of greater interest in the industry.

Chapter 5

Discussion and Related Work

Here a general discussion of the results from the previous chapter will take place. It will be where the analysis takes place, and is important for highlighting interesting results by comparing and discussing them and in that way help the reader get the most from the results in the previous section. The results generated in this thesis will then be related and compared to results generated by other papers. Discussion and analysis on how this thesis fits the landscape of papers carried out in this area. Are the results contradicting some parts, or perhaps it can provide a deeper aspect or explanation of results which others have found.

It will end with a section suggesting areas of interest which can be explored by future research. Hopefully to inspire and help guide future generation of researchers to find interesting problematics and to give insight what might be possible in the future of this area. It will also include parts which presents works that was initiated in this project but was not completed due to changes of scope and direction. Discussing the early foresights which can be used as a reconnaissance suggesting preliminary results which would require future research in order to draw any conclusions from them.

5.1 Discussion

In this section, there will be some overall discussions about the results and how they relate to the research questions. The section will first discuss how well the selected methods of modelling collaboration and developer structure, as specified by research question 1. It will follow with discussion relating to research question 2, explaining how well correlation to between the structure metrics and throughput or complexity metrics. Finally some general observation made from the results will be discussed.

5.1.1 (RQ1) How can internal developer structure and level of collaboration in a project or part of a project be modeled?

To answer research question 1 two different suggestions of how to model collaboration and developer structure was created and tested. The models are created from concepts that seemed to capture collaboration well but when quantified and turned into a single number it became hard to gain intelligible results.

The models explored turned out to not capture the collaboration to a satisfactory level and future research can take that knowledge and tackle the problem from different aspects. As research question 2 was impacted by how the models for research question 1 was decided it was hard to gain conclusive results, most of the generalizable results comes from looking at a simpler model based on amount of committers, which does not capture collaboration that well.

A conclusion that could be drawn by comparing figures 4.12 through 4.16 with figures 4.7a and 4.7b is that while the the fractal value produce somewhat intelligible results on a file level, the loss of detail in converting them to project level by methods such as taking the mean value makes them unsuitable for the project level. For more conclusive discussion of a project-wide level of examination, more research may need to be conducted. When conducting similar research it could either be of use to focus on the lower level resolutions, or if a desire for higher levels exist, be careful in the selection of models so that something suitable for the level can be obtained. Conclusions that can be drawn from the project level results is that complexity and integration speed does not correlate with the average fractal value or vertex strength, making the representation somewhat a failure of both research questions on that level. The clearer results of examining the number of commiters carries the implication that keeping examined values simple is generally preferable to creating more advanced models. Despite the intriguing results using the metric, the number of commiters is not an accurate measurement of collaboration as it says nothing about the balance of work between the commiters. For example, if there are 5 contributors on a file they may have done the same amount of commits or one contributor may have done 90% of the commits. These outcomes would give differing fractal values but look the same when examining number of commits pointing to a disconnect from the research question.

Even though the examination of figures 4.7a and 4.7b shows that exploring the fractal value for each file gives clearer results than the projectwide exploration of figures 4.15 and 4.12 the representation may still be flawed as some results are more observable by simply using the number of contributor as seen in for example figure 4.8. This along with the insight from figure 4.1 through 4.3 implying that the vertex strength representation is fairly evenly distributed between developers implies that more work in creating an effective representation of collaboration may be a necessity for to accurately model it.

When examining the similarities of additions and commits to a file which was an important part of creating the model, as Adam Tornhill stated that these two values are highly correlated[11], we found that the correlation is not that clear which can be observed in figure 4.5. Outliers with high amount of additions of lines seems to occurs more commonly with low amount of commits which could be explained with automatically generated files and things like one developer reformatting a file with the help of a tool, another explanation

can be that it shows the copying and pasting between files. Because of these reasons commits were pursued further, and used as a base for the fractal value and network analysis.

To aid in finding models of ownership in a project, literature was used as an aid. The goal of these literature studies was an analysis of models that have been explored in the past that could serve as a basis for the implemented prototype.

Due to the functionality of the prototype it was necessary to run the data collection over a timespan and as the goal was to model collaboration some considerations had to be taken when selecting this timespan. A logical assumption to make is that a person who changed something in a file five years ago has done little collaboration with a person who changed in the file today, when the context and file might have changed considerably. Because the model for fractal value and vertex strength is based on the fact that developers whom have developed on the same file have shared a collaboration the timespan had to be limited in order for changes to the same file to still be considered changes. Analysis is for those reasons done using data from a specified time interval, for the file and project level graphs it been set to a two year span from 2017 to 2019. For the interval plots the set interval is four months and has a starting point from 2011 up until June of 2019.

The span of two years was experimentally determined as the period needed to be large enough for all 107 repositories to have recorded any activity during the period, while still being small enough for a collaboration to still be regarded as relevant. By utilizing weighting or alternative extractions of ownership the time period could be extended while preserving the notion of collaboration. Weighting the commits by the time passed since they occurred is one alternative possible to pursue, determining such a weight was outside the scope of this thesis. This would require some sort of definition of how long knowledge and ownership stays after an edit has been made and when editing the same sections should be considered collaboration or not. In the case of going into more detailed exploration such as a line based approach where the owner of a line is the developer that last edited it, it could be of use with older, now static, contributions not affecting analysis of the current state to a large degree.

5.1.2 (RQ2) Is there a relationship between the level of collaboration between developers and complexity or integration speed?

To answer research question 2 the models used to answer research question 1 was compared to the simultaneously gathered measurements of complexity and integration speed to explore if correlations could be found.

Some correlations are visible when examining the produced graphs at a file level but what might be of higher interest is the fact that these correlations were not visible when examining the same values where resolution has been changed to the project level. In these cases it might be because of impreciseness and dilution that happens when compounding the metrics a single number.

From max complexity, average complexity and integration speed, the median and interquartile range was the lowest when there were around one to five developers (figure 4.8 and 4.10). When there are more developers than five then the max complexity is shown to have a larger increase while the average complexity of the functions in the file are stable. What is lacking in the average complexity of the functions are the resolution of scale, perhaps when

multiple people are developing there is an addition to smaller helper functions which could bring the average down while complexity in the bigger functions increase.

Integration speed is more stable and amount of developers does not seem to have a big impact on the speed. While having low amount of developers is better it can be seen from figure 4.10 that the regression slopes downwards and also the median is lower for two and three developers rather than one. Reasons for this could be that when collaborating the responsibility of the file is not only towards the project as a whole but towards the other persons. When two or three people collaborate, then they are all responsible and can spur each other to integrate and share their work. Also interesting for integration speed is that it seems as though the amount of outliers is the highest between one to three developers. This could be due to the nature of open source work where because work does not necessarily get assigned a single developer or a small group can work on one file for a longer time before choosing to contribute and integrate it to the main repository.

As there appears to be connections between complexity and number of commiters (figure 4.8) and a smaller connection between commiters and integration time (figure 4.10), both on a median level and for outliers, it seems fairly likely that the level of collaboration is correlated with complexity and integration time, giving a hypothesis of the answer to research question 2. However, as the suggested methods of modelling this in a more thorough manner than simply examining the number of commits may need more work to be generally utilizable and other factors could be examined in more details for a better verification of the answer.

5.1.3 General Discussion

This section aims to include discussions and analysis which are relevant and interesting but only tangentially connect to the stated research questions.

No generalizable insight could be obtained from examination of the interval graphs. While not generalizable it still provided interesting grounds for analysis specific to the repository analysed that way. It can be used for analysing the trends and monitoring the relevant metric, giving ground for analysing why spikes have happened and to see how a implemented change in development methods impact these metrics. Analysis of this could also be of use when attempting to predict future trends of the metrics which could potentially be done utilizing machine learning algorithms. The resolution of looking at four month spans was experimentally determined in order to get enough data in order to be relevant, this could be because of how open source projects can have lots of variance in development intensity and activity. A more active and consistent repository opens up the possibility for shorter timespans to become relevant for analysis. Because we suspect that this sort of consistency is more common among close sourced repositories, this way of analyzing a project may be of greater interest to the industry.

This thesis especially with RQ1 has been battling definition of what ownership is and why anyone should care about it and the debate will keep on happening in the future. Bird et al. argues that because ownership is actionable it is an interesting metric to investigate.[5] Kent Beck promoted collaborative ownership and pair-programming with his book about eXtreme programming. [3] Because processes and decisions can be made which change how developers collaborate and develop files can be produced and enforced it is interesting to understand how this ownership relate to other metrics.

Is it something assigned, like property, as if a file is like a house with papers stating who

owned it, and no matter how many people collaborate to build it will always be owned by the one that signed the paper for the houses creation. Is ownership assigned such that people own the parts they built? Or perhaps ownership has nothing to do with the actual building of the house and the terminology has blinded the researchers including the authors of this thesis to focused on concepts that does not apply for software development.

If you want to tear down a wall in a house, you do not care for who painted the facade, you are interested in the electrician, plumber and architect to know if there are cables, pipes or if the wall is supporting the roof. Ownership does not drive development of houses and certainly does not drive software changes, transferring knowledge does. If a change is going to be made to a file, the developer in charge of the change is not interested the ownership property but rather in the knowledge. The one who owns a change is likely to have knowledge about it, but if the change was made a long time ago than the knowledge might be forgotten. Like wise someone who has not made any changes to that file might have excellent knowledge of it if they have reviewed the file. Instead of ownership, perhaps the interesting metric should be knowledge when it comes to investigating the properties of collaborating.

5.2 Threats to Validity

In this section, some potential flaws throughout the process will be discussed, both in a more general sense and in some specific cases.

The selected 107 repositories used were picked in based on the qualifiers of activity and spread of contributions, repositories of less than 3 commiters or less than 200 commits were omitted in favor of repositories where a substantial number contributions had been made by multiple developers in order to better simulate a context where developers collaborate to develop. Because the need to analyse multiple projects around the same time period the repositories selected needed to have activity around the same time period. A sizable subset of the inspected repositories tended to have bursts of high activity followed by periods of low to no activity, this burst of activity had to fall inside the common time period of evaluation in order to be useful for our analysis. A conscious effort was also made in order to get variance in size of the repositories. While this can lead to some correlations being more easily observable with values being both lower and higher it could present a threat to the generalizability of the results.

Because the selection of files for plotting were random among all source code files from the repositories it is more likely that files were selected from repositories containing more files. Containing more files does not necessarily state anything about the type of project or if it is smaller or larger in terms of lines of code as the amount of files in a project depends on a wide variety of parameters such as the designed architecture, interfaces and helper files, programming language. But does introduce a bias of the result where files from bigger repositories are over-represented.

Because of the above mentioned selection processes have been used they constitute sources of bias in the data set, where the data set might not be a fair representation of the average open source repository but rather a selection of repositories which contain multiple contributors.

While the two-year span was useful for allowing all of the selected repositories to be used, it may not have been the optimal length for this. A shorter time span could have led to

collaborations being more accurately determined as a commit might no longer be relevant after two years of activity.

As git logs are used as the basis for most of the gathered data operations that alter the git log such as `git squash` and `git rebase` leads to the accuracy of the models to be reduced. Squash specifically lowers the resolution by compiling a large chunk of commits into one while rebase is a bit more intricate. Rebase redirects all commits from one branch to another by reapplying them on top of another branch in a way rewriting history of the tree. For complexity analysis squash and rebase does not impact the metric; the collaborative metrics based on commits are impacted by squash; rebase fully circumvent the integration speed model by rewriting the tree structure while squash still gets recorded but reduces the metric by compiling and setting the commit date to a later than the actual. Due to the difficulty and complexity of rebasing long diverged changes as merge conflicts need to be resolved for each commit it is likely not commonly done and did probably not influence the result significantly. Squashing on the other hand can be more common, which can have impacted both collaboration metrics and integration speed. The distribution of squashing is unknown in repository is unknown, it might affect all developers equally or certain developers are more likely to squash or not. Equally it could have resulted in that integration speed was recorded to be lower than it was supposed to be.

Some of the utilized tools had certain flaws that may have negativly affected the end results. The lizard tool used for finding the complexity had a flaw where the complexity of main methods was not calculated for some programming languages, leading to a common situation where the presented complexity is lower than the actual complexity, in some cases even going down to zero but as that value should not normally occur for cyclomatic complexity the data points of those values were ignored in the graphs and analysis. Ignoring these values also provided a threat to the validity as the data points then become selective and more limited in amount, making generalizations less accurate.

5.3 Related Work

Because this thesis is not the first one to explore this field there exists a space containing other research attempting to investigate the same or similar research questions. The section will contain a small summary of a selection of related papers and more importantly some discussions and comparisons between the conclusions made in the relating research to the one done in the thesis in order to analyse and explain how this thesis fits into the bigger picture of the field. It tries to answer if the result in this thesis strengthen or contradict previous conclusions and if observations made in the field can gain a richer explanation finding the cause of the effects noticed.

5.3.1 Dont touch my code!: examining the effects of ownership on software quality

Christian Bird et al. analyzed the closed source software repositories of Windows Vista and Windows 7, their level of granularity was in the middle between project level and file level, as they focused on software components. Components were a collection of different files

which produced one binary such as a .dll library, .exe executable or .sys driver. By looking at the raw amount of larger and smaller contributors which worked on the files that produced the component and relating it to failures connected to the component they explored how collaboration impacted the amount of failures.[5]

They claim to have found appropriate models for measuring ownership and through quantitative evaluation of relationships between the ownership measures and software failures they proved that people who have a low ownership of a component have a stronger correlation with software failures relating that component.[5]

Because Bird et al. have examined some of the Windows operating systems which are closed sourced projects, it can be hard to compare the results to the findings of this thesis as they are coming from open source projects, but interesting parallels can be made.

Their definition of ownership was based on how many major and minor developers were working on a component where a developer having done more than 5% is considered a major developer and a developer having done less than 5% is considered a minor developer. What they found was that an increased amount of developers on a component, specifically minor developers, was strongly correlated with failures.[5] That can be compared with this thesis result where increasing the fractal value above 0.75, meaning a fragmented development that probably includes several minor developers, had a relationship with higher complexity (see figure 4.7a). The closer the fractal value goes towards 1 the more minor developers are connected to that file, which raises the question if the increase in failures found in the Windows components was caused by the increased complexity of the modules that produced that binary. This is a potential connection which could explain the results found by Bird et al.

5.3.2 Fractal Figures

In a paper by D'Ambros et al.[7] developer efforts are examined in a way to answer questions about how split the development effort of a file or folder is and how this insight could potentially be used to reorganize the team structure if it is found lacking. This is mainly done using an image showing a figure of how the development is distributed between developers. The fractal value also used throughout this thesis is utilized in a case study of open source projects from the Mozilla Foundation using the CVS version control system as a basis. As an example of correlating the fractal value to another metric, the fractal value is compared to the number of bugs for the files in the repositories. Much like the first research question of this thesis the overall goal of the paper is a representation of development structure with a focus on validating the explored representation.

The analysis done in the paper showed that comparing the number of bugs to the level of developer distribution showed that all files a high amount of found bugs were those with higher fractal value, meaning more spread development with no clear main contributor. However, there also appears to be a large amount of files with spread development and low amount of bugs as the upper corner of the presented graph appears to have a clump of data points. It may also be a possibility that the difference of actual bugs is not as high as only found bugs can be measured where a higher amount of developers may make it easier to find otherwise hidden bugs.

Much what was explored in this thesis the fractal figure paper examines the fractal value at a file level and compares it to other metrics. A point of difference is the value to compare to, the fractal paper focuses on the number of bugs which was not examined by this thesis due

to being outside of the scope and being difficult to extract from just using a version control system. The results with comparison to number of bugs were also cleaner and more easily observed than the comparison to complexity and integration speed done in earlier chapters. This may be because of stronger relations for number of bugs than the other metrics, or it may be because of the data set being more uniform as Mozilla projects were used as the sole source of information.

5.3.3 A Degree-of-Knowledge Model to Capture Source Code Familiarity

In a paper by Fritz et al. [26] how to model the degree of knowledge of a section of source code for a developer. This model would enlighten the developer structure by finding the experts of various sections. This model looks at aspects such as initial authorship, finished and unfinished changes after the initial version and a more thorough examination of the amount of interactions between developers and code and the timeframes of these interactions. The model is substantiated by two case studies in closed source commercial development contexts. For exact weightings in creating the Degree of Knowledge a survey where developers answered questions about how familiar they were with code they had recently edited or examined, which was then compared to the collected data.

Through the case studies the Degree of Knowledge method was shown to work fairly well at finding someone knowledgeable about a section of code, finding this person more accurately than utilizing the gut feeling of the developers. A potential flaw in the model is that it may not be completely generalizable due to the smaller amount of case studies. This is also suggested by the lower correlation when using the weights of the first case study on the context of the second case study, meaning that the results may be better when recalculating the weights for new contexts.

Compared to this thesis the study of the paper examines workflow on a lower level with examination of things such as on what weekday changes or deliveries are made. While the difference in scope makes it difficult to directly compare the results to those of this thesis, something similar to the degree of knowledge could potentially be utilized as a basis in future research as a basis for higher level values. There would however, be some problems with this as data about who has opened or looked at a section of code may need to be collected, as well as more work in verifying the weights being recommended.

5.3.4 Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review

Code ownership traditionally relies on authorship. Ownership usually defined by the authorship of the file's conception or the amount of changes in commits or lines changed or any other derivation a developer can apply. The more changes that has been done by a developer, the more that developer is said to have ownership over it.

Thongtanunam et al. complements the traditional idea of ownership used by Bird et al. in [5] which uses a commit based definition and the one used by Rahman and Devanbu in [6]

which uses a line based one. By including code review into the ownership metric they hope to uncover links between code reviews and ownership and substantiates their findings with a case study in an open source context.[39] Thongtanunam et al. found that the majority of developers in their case study who had contributed to any module had not actually done any code change but rather had contributed by code review. They also found that modules with post release faults tended to have more developers who were both minor contributors of code and also minor contributors of reviews. They also conclude that including review statistics can improve an ownership metric.[39]

Because information about reviews are not available through git and require other sources this thesis did not include review statistics in the models for ownership and collaboration. But by narrowing in the scope and focusing on a set with fewer repositories would make it possible to get the data needed to measure reviews and turn it into a metric.

5.4 Future Work

In this section, we will discuss some ideas for how continuation of research in the field could happen, including some ideas where earlier investigation have been started throughout the work done in this thesis.

An aspect that could be focused on for future studies is utilizing weightings of the metadata extracted, which if applied correctly could give a more accurate representation. This could for example be something like weighting a project wide value based on some parameters of the project context or file values on for example the size of the file. Because this thesis implemented a breadth of models and extraction methods time did not exist to properly investigate how weights would best be applied. In order to utilize the weights they need sufficient testing and experimentation in order to find scientifically proven improved model.

With a more focused scope there are a few interesting paths to take, for example investigate knowledge or ownership in a project and comparing patterns of high knowledge developers and low knowledge developers to see what differences and effects there are of various distributions of this.

Instead of randomly selecting files for plotting, it would be possible to categorize the files and repositories to try and find results regarding specific types of projects or files. For example a larger amount of files with a high number of contributors could be used and broader generalizations about these files can be done than in a random context where only a low amount of these files are analyzed.

In case of further analysis of how to represent collaboration and developer structure accurately trying values other than the number of commits may be useful. In this thesis some rudimentary analysis of alternative bases were performed and of the examined values the number of owned lines seemed like it could potentially lead to interesting results. Added lines also looked like it could lead to different results.

For studies focused on mining software repositories it may also be interesting to utilize machine learning algorithms as a way to predict how a value will develop in the future. This could take the form of the interval trend plots with further fields for predicted future values and could be applied to a variety of values.

Something that was used in some related papers that could have been utilized as an examined metric is the number of minor developers and the number of major developers and

how these numbers relate to each other. In the paper where this was discussed[5] the threshold separating major and minor developers were 5% but it is also something that could be experimented with. This way of looking at development would give a simple representation of development structure of a section that could be used for future analysis beyond what has already been done.

Finally performing similar studies using different models may be of interest. For better results these models could also use different kinds of sources like for example an issue tracker. The initial phase of finding a suitable model is an important step of finding new or improved representations and a focus on this part may be of benefit for future studies.

Chapter 6
Conclusions

Because of the need for constant decisions and the ongoing debate regarding how to structure collaboration in a project the field has useful potential applications. A prototype was produced which can be executed on any git repository, collecting metadata, processing and presenting it. To investigate the research questions methods of modeling representations of collaboration, complexity and integration speed through git metadata was done and extracted by the prototype. The metrics created was explored on a file level and on a project level to investigate relationships between them.

Research Question 1: How can internal developer structure and level of collaboration in a project or part of a project be modeled?

Finding a good model for collaboration present difficulties, this thesis has investigated two different high level models, being the fractal value of D'Ambros et al.[7] and the vertex strength extractable from a network representation of development. These models initially seemed promising but both had issues when interpreting results from them as they had flaws describing the true nature of collaboration.

Because of fluctuations when it comes to copy-paste, auto-generated files and reformatting, the models were built on the number of commits rather than being line-based. Commits and added lines were shown to be correlated (see figure 4.5) but behave differently at low amount of commits, probably due to the said fluctuations.

Because the vertex strength represents the collaboration network in a bigger scale it was not applicable on a file level which reduced its usefulness for relating to the performance metrics. The fractal value was able to be extracted on a per file basis and on a project level. The project level was not as useful as file level because of how the data was compiled and resolution became to low to get any conclusive results.

Research Question 2: Is there a relationship between the level of collaboration between developers and complexity or integration speed?

While no clear connections were found on the project level there appeared to be some on a file level. While the produced models hinted at similar results, the most observable results could be seen by examining the number of collaborators.

Having between 1 to 5 collaborators on a file showed similar results but having more than 5 seemed to lead to an increase in complexity. Integration speed was not as significantly impacted by amount of developers. A problem with the pure number of commiters does not say a lot about the nature of collaboration and may not directly answer the research question but the metric still is of interest.

Final Remarks

While automatically collected metrics can not replace human decision with the technology of today and work is required to collect useful information or to construct useful models there is high potential in the field.

Through this thesis new methods of something like finding the integration speed of a git commit have been found and could be used in future applications. Testing of certain models have been conducted so that they can be expanded upon in future studies for better representation. A prototype which can automatically extract and present the data has been created and can be used as a base for future research or as an aid to analysis for companies.

Bibliography

[1] I. Ajzen, *The Social Psychology of Human Decision Making*. Guilford press, 1996.

[2] "Github." `https://octoverse.github.com/`, 2019. Accessed on 2019-07-11.

[3] K. Beck and E. Gamma, *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[4] N. Forsgren, J. Humble, and G. Kim, *Accelerate - Building and Scaling High Performing Technology Organizations*. IT Revolution, 2018.

[5] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! examining the effects of ownership on software quality," *ESEC/FSE Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011.

[6] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceedings of the 33rd International Conference on Software Engineering*, pp. 491–500, ACM, 2011.

[7] M. D'Ambros, M. Lanza, and H. Gall, "Fractal figures: Visualizing development effort for cvs entities," in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2005.

[8] C. Tenggren and N. Johansson, "Social network analysis of open source projects," Master's thesis, Department of Computer Science Faculty of Engineering LTH, 6 2015.

[9] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, "How developers drive software evolution," in *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, pp. 113–122, IEEE, 2005.

[10] M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger, "Analysing software repositories to understand software evolution," in *Software evolution*, pp. 37–67, Springer, 2008.

[11] A. Tornhill, *Your code as a crime scene: use forensic techniques to arrest defects, bottlenecks, and bad design in your programs*. Pragmatic Bookshelf, 2015.

[12] O. Alonso, P. T. Devanbu, and M. Gertz, "Expertise identification and visualization from cvs," in *Proceedings of the 2008 international working conference on Mining software repositories*, pp. 125–128, ACM, 2008.

[13] A. Orucevic-Alagic and M. Höst, "Network analysis of a large scale open source project," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 25–29, IEEE, 2014.

[14] "About git." `https://git-scm.com/about`, 2019. Accessed on 2019-07-15.

[15] S. Chacon and B. Straub, *Pro Git*, ch. 9.2, pp. 413–431. Apress, 2 ed., 2014.

[16] "Lizard." `http://www.lizard.ws/`, 2019. Accessed on 2019-07-11.

[17] E. H. Bersoff and A. M. Davis, "Impacts of life cycle models on software," *Communications of the ACM*, vol. 34, 8 1991.

[18] M. Kelly, *Configuration Management - The Changing Image*, ch. 5 and 7. McGraw-Hill Book Company, 1996.

[19] M. A. Daniels, *Principles of Configuration Management, Advanced Applications*, ch. 4. Advanced Applications Consultants Inc., 1985.

[20] P. H. Feiler, "Configuration management models in commercial environments," Tech. Rep. SEI-91-TR-7, Software Engineering Institute, 1991.

[21] "Git documentation." `https://git-scm.com/docs`, 2019. Accessed on 2019-04-24.

[22] A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani, "The architecture of complex weighted networks," *Proceedings of the national academy of sciences*, vol. 101, no. 11, pp. 3747–3752, 2004.

[23] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019.

[24] Y. Liu, E. Stroulia, and H. Erdogmus, "Understanding the open-source software development process: a case study with cvschecker," in *Proc. 1st Intl. Conf. on Open Source Systems*, pp. 11–15, 2005.

[25] X. Meng, B. P. Miller, W. R. Williams, and A. R. Bernat, "Mining software repositories for accurate authorship," in *2013 IEEE International Conference on Software Maintenance*, pp. 250–259, IEEE, 2013.

[26] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pp. 385–394, ACM, 2010.

[27] S. Hangal, D. MacLean, M. S. Lam, and J. Heer, "All friends are not equal: Using weights in social graphs to improve search," in *Workshop on Social Network Mining & Analysis, ACM KDD*, 2010.

[28] "Networkx." `https://networkx.github.io/`, 2019. Accessed on 2019-07-22.

[29] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[30] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *IEEE transactions on software engineering*, vol. 17, no. 12, pp. 1284–1288, 1991.

[31] M. H. Halstead *et al.*, *Elements of software science*, vol. 7.

[32] C. Usage and B. of Continuous Integration in Open-Source Projects, "Michael hilton and timothy tunnell and kai huang and darko marinov and danny dig," *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, vol. 31, 8 2016.

[33] M. Fowler, "Continuous integration." `https://martinfowler.com/articles/continuousIntegration.html`, 2006. Accessed on 2019-07-15.

[34] H. Wickham, J. Hester, and R. Francois, *readr: Read Rectangular Text Data*, 2018. R package version 1.3.1.

[35] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016.

[36] C. Sievert, *plotly for R*, 2018.

[37] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019.

[38] A. Vance, "Data analysts captivated by r's power," *New York Times*, 1 2009.

[39] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Revisiting code ownership and its relationship with software quality in the scope of modern code review," in *IEEE/ACM 38th IEEE International Conference on Software Engineering*, 2016.