MASTER'S THESIS | LUND UNIVERSITY 2015

Social Network Analysis of Open Source Projects

Christian Tenggren, Nicklas Johansson

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2015-30

Social Network Analysis of Open Source Projects

Christian Tenggren
ada10cte@student.lu.se

Nicklas Johansson
ada10njo@student.lu.se

June 16, 2015

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Alma Orucevic-Alagic, alma.orucevic-alagic@cs.lth.se

Examiner: Martin Höst, martin.host@cs.lth.se

Abstract

A large amount of widespread software used today is either open source or includes open-source projects. Much open-source software has proved to be of very high quality despite being developed through unconventional methods. The success of open-source products has sparked an interest in the software industry in why these projects are so successful and how this seemingly unstructured development process can yield such great results.

This thesis presents a study done on the projects hosted by one of the largest and most well-known open-software communities that exists. The study involves gathering developer collaboration data and then using social network analysis to find trends in the data that eventually might be used to create benchmarks for open-source software development. The results show that several interesting trends can be found.

Keywords: Open-Source, Apache Software Foundation, Social network analysis, Network metrics, Committer

Acknowledgements

We would like to thank Alma Orucevic-Alagic for the great feedback we have obtained during this project. We would also like to thank the department of Computer Science at Lund University for providing great workplaces for us.

Contents

Chapter 1

Introduction

Software development is a complex process where a significant number of software projects exceed their initial budget. This can be partially due to ineffectiveness in the development process [14]. It is therefore in companies' interest to gain knowledge of how to improve the development process as much as possible. One way to do this is to study developers' collaboration networks, which has been difficult due to the prevalence of closed-source projects until the last two decades [30]. It is not until recently that an increasing amount of mature and industry used open-source has come to be, some which receive contributions from professional developers, both on their free time as well as part of their paid work [30]. This has lead to an increasing amount of publicly available data on successful and mature projects, from which it should be possible extract metrics which could provide benchmarks for good practices in software development.

1.1 Open Source

Open-source software has existed ever since the development of software began in the mid 20th century. Much early software was open source but with time the amount of open-source software declined while the prevalence of proprietary software increased. It is only fairly recently that open-source software has gained a lot of attention. The open-source way of developing software differs from the usual in-house development model that are standard at most large software companies today. The main differences are [34][41]:

- Open-source projects often do not have any strict project plan or deadlines
- The developers are not assigned any specific tasks, they work on the things they want to work on
- The amount of developers on a single project can be thousands spread worldwide which means that they do not usually meet face to face but instead communicate using mailing lists and bulletin boards.

Even though these differences exist between open-source development and the models used by companies there are still many open-source projects which have resulted in software that has been equivalent or in some cases even superior to the software developed in the industry, examples are Apache HTTP Server [35], Firefox and Linux. This has sparked an interest in how this seemingly unstructured development model can produce competitive software. Some [41] speculate that this is because developers in open-source projects tend to select work that interest them, but it is hard to find any empirical studies on the subject.

Open-source software is of interest to many corporations today as it can be cheaper and is often supported for longer since there is no company that owns the software that can go bankrupt or decide that they do not want to invest any money in the specific software anymore. It also gives the company more control over the software compared to closed-source software, either by modifying it for their needs or by submitting fixes and changes to the open-source project. Integrating open source components into a company's product can also reduce time to market as less time has to be spent writing code for problems that has already been solved.

Production of high quality software by open-source communities motivated further investigation into how the developers contribute and collaborate in open-source projects. One way to study developer collaboration is through social network analysis.

1.2 Social network analysis

Social network analysis has been very popular the past two decades and it has been applied to many different areas of research like economics [27], social development [18] and spreading of diseases [28].

Social network analysis is a way of measuring specific relationships between different entities in a graph. Each node or vertex in a network represents one entity, such as a developer, and the links or edges between vertices represent some kind of relationship. Exactly what the vertices and edges represent is dependent on what one wants to gain out of the analysis. A real world phenomenon can therefore have multiple representations as a graph.

The networks created can have different attributes. Edges can be either directed or undirected. Undirected edges indicate a mutual relationship between the vertices, while directed edges can be used to represent either a one-sided relationship or a mutual relationship with different weights. A weight attached to an edge can demonstrate how strong of a relationship it represents or how much information that flows through the edge, see visual example of network types in Figure 1.1.

There is a wide range of metrics that one can apply when performing a network analysis. Each metric looks at a specific property of the network. This thesis focuses on centrality indices and the clustering coefficient, described in detail in Section 3.3. The clustering coefficient can give an indication if there exist subgroups in the committer networks and the centrality metrics can show the influence the developers have over each other.

Visualizing the networks are key in finding important individuals and getting a better overview over the structure of the network. Large networks will often have many intersecting edges and vertices overlapping each other, which will make them hard to comprehend.

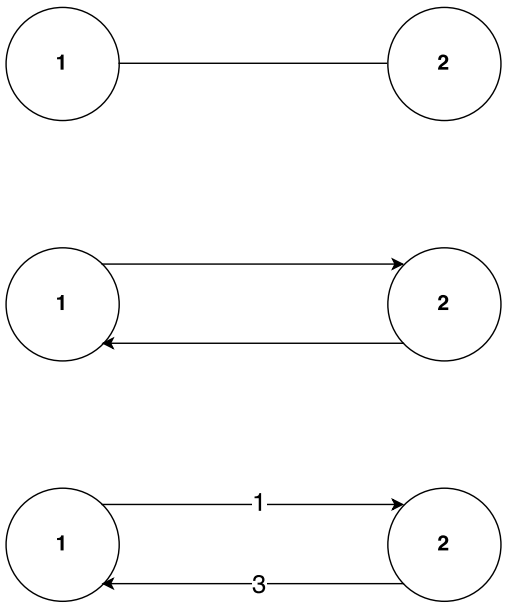


Figure 1.1: Example of undirected, directed and weighted directed graphs

This problem can be solved with some kind of layout algorithm that alters the placement of the vertices to minimize the number of edge intersections and vertex overlaps [37].

This thesis studies a community of developers using social network analysis. The analysis provides knowledge in how the collaboration in the community works and which factors that are common for successful development of high quality software.

1.3 Contributions

The work in this thesis has mostly been distributed fairly evenly between the authors. Christian has been more responsible for calculating metrics with tnet and writing the approach and methodology, while Nicklas has been more responsible for the introduction and the background parts of the report. The rest of the work has been contributed to by both authors fairly evenly.

Chapter 2

Background

The data used as a basis for this thesis was collected from the Apache Software Foundation (ASF). In this chapter it is described how the Apache Foundation is organized and what structure they use to ensure the quality of the projects accepted into the organization. Following that is an introduction to some of the tools used to analyze the gathered data. This chapter also contains a section describing earlier work around open-source that are relevant to this study.

2.1 Apache Software Foundation

The Apache software foundation is a open-source foundation which today comprises over 150 different open source projects [8]. The initiative to create the foundation was taken in 1999 when an already established group of developers decided that a more legal structure was needed for their software efforts [42]. Several of the projects hosted by the ASF have been hugely successful, for example the Apache Web Server [3], Apache OpenOffice [7] and the Hadoop [2] distributed computing engine. All projects are distributed under the Apache License, which lets anyone freely distribute and modify the software [6]. The only requirement for modifying the software is that all files that have been modified must contain a notice stating that the files has been modified from the source form. The parts of the software that have not been modified must retain any original copyright, patent and trademark.

2.1.1 ASF-structure

Each of the projects hosted by the foundation are governed by a Project Managing Committee (PMC), consisting of individuals that has shown great leadership and commitment to the project [8]. The PMC is responsible for addressing (eventual) legal issues surrounding the project and making sure that the community is healthy and that good collaboration is achieved in the project. Mailing lists are the preferred form of communication between

developers, since developers are spread out all across the world asynchronous communication is favored. The mailing lists also makes for easy archiving of discussions.

There are several different specified roles in the ASF community [26]. Only the roles relevant for this thesis are presented below. These roles can be split into ranks of responsibility where ”User” is the lowest rank and ”Committer” is the highest rank.

User A user uses the software and eventually reports bugs and suggests additional features to a project. Users may also help other users via the support forum on the Apache website.

Developer A developer/contributor is a user who contributes to a project by writing code and documentation. Developers are active on the project mailing lists and participate in discussions.

Committer A committer is a developer that has been given write access to the code repository. It is only already established committers that can appoint new committers, this is done by having a vote among the committers. Commiters can commit code written by non-approved developers.

2.1.2 Apache projects

The projects hosted on the Apache Software Foundation can be divided into projects that were created specifically for the Apache Software Foundation and projects that were added to the foundation sometime during the projects' life cycle. The latter type all go through the Apache Incubator process [4]. In order for a project to be considered as a candidate for joining the incubator program, it must first find a 'champion'. A champion is a experienced member of the Apache community that will act as an adversary for the projects and assist it until it either graduates from the incubation or is removed. The project and the champion creates a proposal that is submitted to a sponsor, either a top level project (if the candidate aims to become a subproject) or the Incubator Project Management Committee (PMC). After discussing the proposal, the sponsor will eventually take a vote to decide if the candidate will be accepted into the incubator program.

A project accepted as an incubator is called a podling. The podling is continually reviewed to see how the community around the podling is evolving and to see if it follows the guidelines for Apache projects. The incubation of the project can be terminated at any time during the podling phase. When the Incubator PMC feels that the podling fulfills all requirements [5] it may recommend the sponsor to graduate the podling into a member of the Apache.

2.2 Revision control systems

The projects hosted by the Apache Software Foundation use revision control systems to track changes in the software and provide backups so that it is possible to restore earlier versions of files. Such systems track when a file has been changed and by whom, as well as store commit messages describing and motivating the changes. By comparing different versions of a file it is possible to see what changes have been made to the file between the

two versions. The Apache Software Foundation uses two different systems for revision control, Subversion(SVN) and Git, which currently are the two most well used solutions according to a survey made by Eclipse [19].

2.2.1 Git

Git is a distributed revision control system for software development. It was developed by Linus Torvalds in 2005 and is today the most used revision system for software configuration management (SCM) [19], surpassing SVN which was the most popular system previously. Git distinguishes itself from other revision control systems by using a different branch model [1]. Git supports several local branches that are independent of each other. This means that a developer can start a branch and commit to it multiple times before realizing that the idea he had will not work and just delete the whole branch while not affecting the rest of the project. This according to the creators of Git makes people more experimental which can be of benefit to the projects using Git.

Git focuses on speed and data integrity. Almost all the operations in Git are performed locally so a communication with a server is not needed. To be able to perform operations locally the whole repository of a project has to be downloaded to the local computer, so every contributor to a project has a full backup of the entire project. It is therefore possible to handle a server crash or corruption of files by having one of the cloned projects pushed up to the hosting server. Data integrity is achieved by using checksums. Every time someone does a checkout on a file the contents of the file are used to calculate a checksum which is then compared to the original checksum to make sure that no changes has been applied to that file [1].

2.2.2 SVN

Apache subversion or most commonly known as SVN is a revision control system whose development was started by CollabNet. In early 2010 it was accepted as a project hosted on the Apache Software Foundation. The motive behind the development of SVN was that the old version control system CVS, contained several bugs and some desired features were missing [10]. So SVN were supposed to work similarly as CVS but with more features while being more manageable.

SVN uses a client-server model where developers work towards a single-shared repository. This is done by establishing a connection to the repository, so a developer needs to have internet-access to work on the repository. This model also requires frequent backups of the repository since the developers do not have the entire repository on their local machines. SVN-repositories often have the same structure with 3 different sub-directories at the root:

Trunk This directory contains the current main development line of code.

Branches A branch is a copy of the trunk in which major changes are applied, this will still preserve the integrity of the code in the trunk. If the changes are successful then the code in the branch is merged back into the trunk.

Tags Tags are a point in time on the trunk or a branch that you want to preserve. This could be major releases or a stable point of the software before the code undergoes major revisions.

2.3 Social network tools

After compiling the data from the Apache Software Foundation, it would have to be analyzed. A couple of well-used tools where chosen for the purpose, Gephi, which is able to visualize the networks as well as calculate many metrics, and tnet, a package for R that specializes in calculating metrics on weighted networks.

2.3.1 Gephi

Gephi [12] is an open-source tool built on the Netbeans platform and is used for network analysis and visualization of graph-networks. The user can manipulate the graphs by changing the structures, shapes and colors to find properties that otherwise would be hard to find. Gephi uses a wide variety of layout algorithms to alter the structure of the networks for increased visualization. Gephi can for example be used on a repository to see how the repository has evolved and how the different contributors has worked on the repository, thereby increasing the visualization of how developers tend to collaborate in software projects. With Gephi you can retrieve different attributes for each specific vertex in the graph, for example the degree of the vertex. Gephi also supports different network metrics such as betweenness centrality, clustering coefficient and average shortest path.

2.3.2 tnet

Tnet [43] is an open-source package for the software environment and language R. R is used for statistical computing and graphics and runs on various platforms such as Windows, OSX and other Unix based systems [40]. There are a lot of available software to use for calculating network metrics on unweighted networks, but there is a lack of tools that do the same calculations on weighted networks. This motivated the development of tnet.

2.4 Small World

The term small-world is used to describe a network in where most vertices are not neighbors but it is possible to reach most vertices from every other vertex in the network with only a small number of hops, i.e. the average shortest path length for the network is small. This property is often fulfilled in networks that have several vertices with a large amount of connections, called hubs.

The small-world phenomenon has grown into a significant area of study since the experiments done by Milgram in 1967 [33].

2.5 Related Work

Social network analysis has been applied to a wide variety of subjects, e.g. the collaboration between authors of scientific papers [36] and on neural networks [29].

Using social network analysis for software projects is nothing new and has been used for analysis of several software projects. Lopez-Fernández et al. analyzed several open source projects [31], among them Apache, although treated as a singular project and not as a collection of individual projects. The size and content of the ASF has also changed considerably since 2006. This research showed that even very large projects fulfill the requirements to be considered small-world.

Crowston analyzed the social structure of several Open-source projects using social network analysis [16]. Their analysis was based on projects hosted on SourceForge, which is a free web-based system that provides tools for Open-source development. The focus of the study were on the bug report part of the development process to study interactions between the developers. The structure that was used for the network analysis in the aforementioned study is very similar to the structure we have planned to use in our study, the edges were defined by interactions between different developers. In their case an interaction is from a sender of a message in a bug report to the sender of the preceding message or to the original bug reporter. Crowston chose to study the centrality of the projects on SourceForge using the degree centrality metric. Their results showed that the centrality differed between the projects, contrary to what they expected.

Madey, Freeh, and Tynan did a similar study on SourceForge-projects where they studied developer collaboration in and between different projects [32]. The research results show that there are individuals that serve as links between many projects. Lopez-Fernández et al. did something similar but instead looked at developers working on the same files [31].

A study done by Orucevic-Alagic and Höst looked at the committers' network in the android open source project [39]. The aim for the study was to find out how one can utilize network analysis to study a development community. Several social network metrics were calculated on the committers' network for android. The results showed that the approach proposed in the study can be used to study developer collaboration in software development communities.

Bird et al. studied the reply-to relationship on mailing lists and developers working on the same files [13]. The study were done on five different mature open-source projects using social network analysis. They found that sub-groups could be found in the networks i.e the networks were modular. They also found that the developers that did changes to the same files also likely interacted with each other on the mailing lists.

Godfrey and Tu did a study on the growth of a huge open-source project over time [23]. The results showed that the project kept a linear growth pattern even after reaching a huge size, several millions lines of code. This contradicts the commonly accepted belief that with an increase in size the growth declines.

Mockus, Fielding, and Herbsleb studied the Apache Web Server which showed that the developer network in Apache projects may have interesting properties, such as that the 15 most active developers stood for more than 85% of the lines of codes produced in the Apache web server [34].

This thesis aims to expand on earlier studies on open-source projects. Previous studies

have often focused on some specific metrics on a small amount of projects where the results sometimes have been inconclusive. This study will be more exploratory and will be performed on a wide variety of mature projects widely used in the industry with the objective to understand how developers collaborate in successful projects. The goal is to find some common property for these projects that can be used as benchmark for open-source development.

Chapter 3

Methodology

This chapter presents the network metrics used and the network graph structure chosen for this thesis.

3.1 Research Questions

The research done in this thesis will be based on the following questions:

- What common collaboration metrics can be observed in the projects hosted on the Apache Software Foundation?
- What (if any) benchmarks for other projects can be created from those observed metrics?

3.2 Networks

In order to do social network analysis, networks representing developer collaboration need to be created. In these networks, developers will be represented as vertices and the edges between them indicates collaboration. Developers have collaborated if they have made edits to the same file, and the strength of that collaboration is relative to their number of edits compared to the total number of edits to the file they have worked on together. In the following example a file has received commits from three different developers and the total amount of commits to that specific file is ten. These three developers are called X, Y and Z respectively. Developer X has committed once to the file while developer Y has done three commits and developer Z has done six changes to the file. The weight on the outwards edges from a developer will then be the number of commits the developer has made divided by the total number of commits. For the edges originating from Developer Y the weight would then be 3/10 in this case. The graph will have two edges between each pair of vertices (developers) since all the developers has at least changed the file once. See Table 3.1 and Figure 3.1. This process would be repeated for all files in the projects and the resulting weights would be summed up.

Developer	Number of Commits
X	1
Y	3
Z	6

Table 3.1: Number of commits per developer

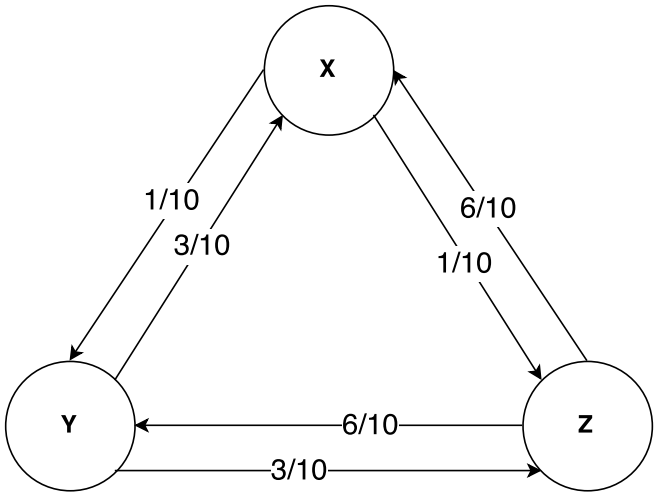


Figure 3.1: Network structure with 3 developers modifying the same file

3.3 Network Analysis Metrics

There are several metrics that are commonly used when analysing social networks. Below are descriptions and definitions of those that was applied to the Apache projects in this paper.

3.3.1 Vertex Strength

Vertex strength is a modified version of vertex degree where the weight of the edges are taken into account. The strength of a vertex v is defined as [11]

$$s_v = \sum_{i \in N(v)} w_{vi} \tag{3.1}$$

where $N(v)$ is the set of all neighbours to vertex v and w_{vi} is the weight of the edge from vertex v to i. In unweighted networks (i.e. all weights are equal to 1) vertex strength and degree are equal. In directed networks each vertex will have an in-degree and an out-degree. The in-degree is the sum of the weights of all edges coming into the vertex and the out-degree is the the sum for all outgoing edges from that vertex.

In this study, the strength of a vertex is a representation of that developers influence over the ones he/she has worked with. A high vertex strength indicates that the developer has a big influence in the social network.

3.3.2 Clustering Coefficient

The clustering coefficient was first introduced in 1998 by Watts and Strogatz [44] and describes how connected the neighborhood of a vertex is. It can be applied to both a single vertex or to a complete graph, where it is defined as the average clustering coefficient of all vertices.

The clustering coefficient for a directed graph is defined as

$$C_v = \frac{n_v}{k_v(k_v - 1)} \tag{3.2}$$

where n_v is the number of edges between the neighbors of vertex v and k_v is the degree of v. In other words, the clustering coefficient is the number of edges between a vertex's neighbors divided by the maximum possible number of edges between its neighbors.

The average clustering coefficient is simply defined as the arithmetic mean of the clustering coefficients:

$$C = \frac{1}{|V|} \sum_{v \in V} C_v \tag{3.3}$$

where V is the set of all vertices in the graph and C_v is the clustering coefficient of vertex v.

For weighted networks, Barrat et al. [11] presented the following definition

$$C_v^w = \frac{1}{s_v * (k_v - 1)} \sum_{i,j} \frac{w_{vi} + w_{vj}}{2} a_{vi} a_{vj} a_{ij} \tag{3.4}$$

where k_v once again is the degree of vertex v, $a_{vi} = 1$ if there exists an edge from v to i and 0 otherwise, w_{vi} is the weight of the edge from v to i and s_v is the strength of vertex v as defined in Equation 3.1.

For directed networks, the standard definition of the global clustering coefficient is not applicable. Instead, a modified version of the global clustering coefficient that uses transitivity may be used.

$$C_G = \frac{T_C}{T} \tag{3.5}$$

Equation 3.5 shows the definition of the global clustering coefficient, where T_C is the number of triangles in the graph and T is the number of connected triples. When transitivity

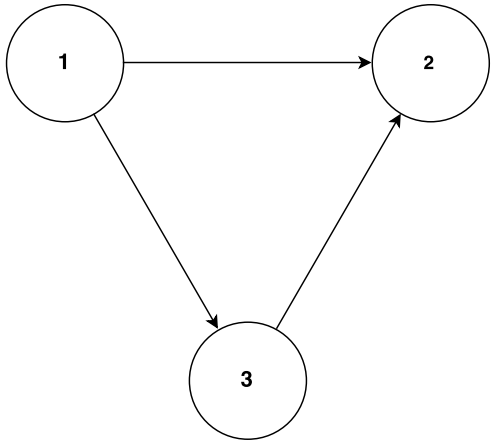


Figure 3.2: A transitive triangle when centered around vertex 3, but not when centered around vertex 1 or 2

is taken into account, only triples where one of the included vertices has both an inwards edge and one outwards. In order for the triple to be included in the connected triangles, there has to be an edge from the start of the chain to the last (third) vertex in the chain. Opsahl and Panzarasa builds upon this version of the global clustering coefficient to adapt it to weighted networks [38].

Note that each triangle is counted three times (once for each vertex).

Gephi does not include calculations of metrics on weighted graphs by default, only supporting calculations of the the unweighted version by Watts and Strogatz. On Gephi's plugin marketplace Griffo has developed a plugin [24] that calculates the weighted clustering coefficient as defined by Barrat et al.

3.3.3 Centrality

There are several different measures of centrality in a network. The centrality measures uses different forms of criteria to indicate the importance of each respective vertex in a network. The importance of a vertex is not an individual attribute but is instead a measure of how much influence the vertex has over other vertices [25]. There is no guarantee that a vertex that is considered important by one criteria is equally important for another criteria.

3.3.4 Degree Centrality

Degree centrality uses the number of adjacent edges as criteria to rank the importance of the vertices. A high number of adjacent edges indicates that a lot of information passes through the vertex. The degree centrality for directed graphs can be expanded to an in-degree and an out-degree, where the in-degree is the sum of weights on the edges coming into the vertex and the out-degree is the total weight of edges leaving the vertex.

$$C_d(v) = d(v) \tag{3.6}$$

where $d(v)$ is the numbers of edges to or from vertex v.

3.3.5 Betweenness Centrality

Betweenness centrality is a measure of an individual vertex's centrality in a network, i.e how many of all the shortest paths in the network that passes through that vertex. A high betweenness centrality indicates that the vertex has a big influence in the flow of data in the network. A vertex with very high betweenness centrality can be a risk since if that vertex for some reason disappears from the network then the communication in the network has to change drastically.

A proposed algorithm for calculating betweenness centrality on weighted networks was presented by Brandes [15]. Brandes suggests that communication might be quicker along a path that is a little longer than the shortest but where the weights of the edges are larger. Larger weight indicates more frequent communication which can be beneficial.

Brandes' algorithm is the one used by gephi for calculating all three centrality indices. He defines betweenness centrality as

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{3.7}$$

where $C_B(v)$ is the betweenness index for vertex v, $\sigma_{st}(v)$ is the number of shortest paths from s to t that passes through v and σ_{st} is the number of shortest paths from s to t. This definition is based on Freeman's article [20].

3.3.6 Closeness Centrality

The sum of the distance of the shortest paths from a vertex to every other vertex in the network is measured as closeness centrality. The idea is that a vertex with a large closeness centrality index has an easier time spreading information to the rest of the network than a vertex with a small index. The definition of the closeness centrality for vertex v is as follows

$$C_C(v) = \frac{1}{\sum\limits_{s \neq v \in V} d_G(v,t)} \tag{3.8}$$

where V is the set of all vertices in the network and $d_G(v,t)$ is the length of the shortest path from vertex v to vertex t [21].

Chapter 4

Approach

In this chapter the work that was performed in order to gather and analyse the needed data is detailed. The basic work process is described in Section 4.1, with each section in this chapter explaining each step respectively.

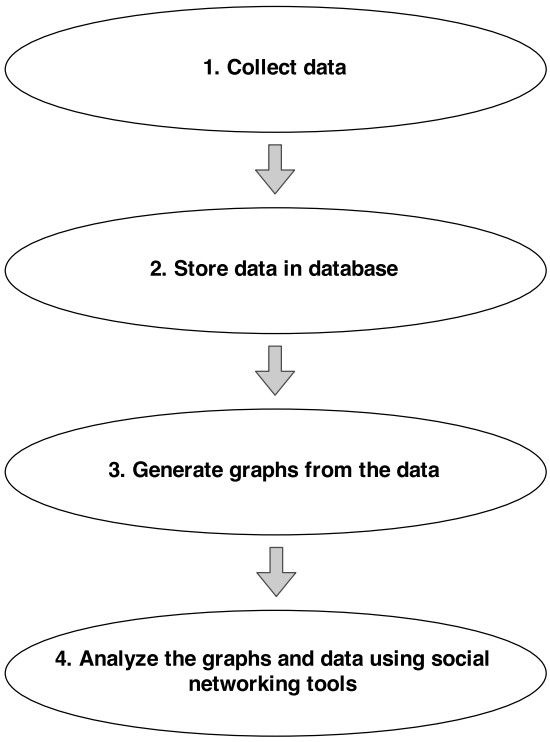


Figure 4.1: Overview over the work process in this project

4.1 Collecting data

Before the analysis of the projects hosted on the Apache Software Foundation could begin the commit histories for the projects had to be retrieved and to do that the location of their repositories had to be collected. Apache hosts an alphabetical list of all their projects [9] with links to pages with information about the projects, including the location of the repositories and which categories the projects belong to. Using the library Curl [17] for fetching data over HTTP, a program to collect the relevant information was written. The commit data was obtained in late February and early March 2015, Apache hosted 249 different projects during this time.

The direct links to git projects were often wrong and needed to be manually corrected. A small part of them lead to repositories hosted on other services, primarily GitHub.

Obtaining the commit-logs for the projects differs depending on if the project uses SVN or Git for revision control. Both systems have built in support for extracting commit logs ('svn log' and 'git log' respectively). SVN's log command is easy to use as it is usable on a remote repository and has an option for formatting the output as XML (–xml), all that is needed is to supply it with the URL for the repository on Apache Software Foundation. In order to also get which files that was modified in a commit, the option '-v' is available. It is fairly slow (downloading the 1.8 GB of logs took approximately two hours) but not too slow to be manageable. By writing a program with concurrent calls to svn's log command the time was cut down significantly.

With Git it is impossible to use the log command on a remote repository, leaving two options for getting the commit history. Either clone all repositories or write a program for parsing the commits viewable on ASF's website. A quick test showed that the latter approach was infeasible as downloading the thousands of commits for each project one by one would take days. On the other hand, cloning all the Git projects was no problem as code repositories tend to be small (< 100 MB). This solution was actually faster than using 'svn log' on a remote repository. Sadly, Git does not have an option for formatting the logs as XML, but does support custom formatting using '-pretty=format:"<format>"'. A short shell script that took advantage of this formatting in order to print xml files similar to those created by SVN was written.

```
git rev-list HEAD
```

gave us all the available revisions and then the following was called for each revision:

```
git log -1 --pretty=format:"<author>%ae</author>%n
<date>%ad</date>%n<msg>%s</msg>%n" <rev>
```

(%ae is the email of the author, %ad is the date of the authoring, %s is the commit message and %n is a newline character.)

```
git log -1 --pretty=format:"" --name-only <rev>
```

In this way, XML output fairly similar to SVN's '–xml' was achieved. The only major difference between the output from SVN and Git was the way they formatted the date, but that was easily solved with some regular expressions. The timestamp in the commits in Git differs from SVN by saving the local time the commit was made and information about in

which timezone it was committed. So this information had to be used to convert all times to a unified time.

The projects that were analyzed were receiving updates constantly which meant that we had to find a way to keep the commit histories up-to-date. It would be unnecessary to retrieve the commit logs already obtained again so to find a way to only collect the commit logs that had been added since our latest retrieval of data were preferred. The 'svn log' command could take individual dates as arguments or take an ID as argument so that only commits with a higher ID than the argument would be obtained. Git also has an option to add a date as an argument, the command.

```
git log '--since<date1>'
```

where '<date1>' is the earliest date we are interested in, all the commits after this date are retrieved. There is also the possibility to add a time together with the date to further specify exactly which revisions one are interested in.

The identification used when committing to a SVN-project on the Apache Software Foundation is a unique username that is chosen upon acquiring committer status. This means that the username can be used to identify who has made the commit. It does not work the same way for the Git-Projects though. A committer in a Git-project is identified by an e-mail address. We discovered that there existed commits where the name of the committer was the same but the e-mail addresses used for committing differed. We assumed that the name of a committer in a project were unique since the probability of two individuals, working on the same project, having the exact same name are next to none and it would not affect our results substantially if it turned out that our assumption were incorrect.

4.2 Database

All data collected in the previous step needed to be stored in a database. A database would make it easy to add new commits fetched from Apache as well as find all projects belonging to a certain category. For this purpose, a MySQL database was created to store data about the various projects including their commit histories. MySQL was chosen due to its availability, robustness and ease of use. It was decided on four relations, one storing the projects, one storing commits, one storing files edited in commits and one storing the categories of projects. A more detailed view can be seen in Figure 4.2.

A program was written in order to parse the XML-files containing the commit histories and insert the data into the database. A few things were noticed when examining the data parsed. Several commits lacked a name of the author and some smaller projects only had a single author/committer, making them irrelevant for this study. After this step 228 projects remained from the original 249. Some project repositories had been unreachable at the time of data collection and some were discarded because of too few committers.

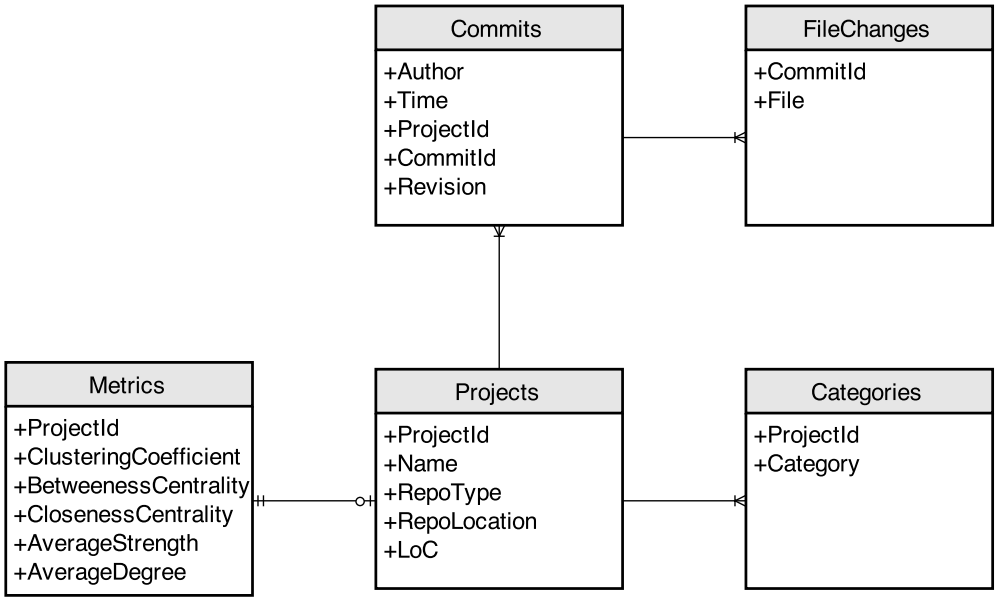


Figure 4.2: Database design

4.3 Graph Generation

The last step before the analysis could start was to generate the developer collaboration graphs. For this to be done, the edges needed to be defined. The most basic edges are between all developers who had worked on the same file in a project. The weight for an edge from developer d_i to developer d_j was defined as

$$\frac{\sum_{f \in F_{ij}} e_{if}}{\sum_{f \in F_{ij}} \sum_{k} e_{kf}} \tag{4.1}$$

where F_{ij} is the set of all files modified by both developer i and j and e_{if} is the number of commits modifying the file f by developer i.

Calculating the edges was trivially done by joining the tables Commits and FileChanges. The calculated edges and their weights could then be stored in a *.gdf* file, as seen in Figure 4.3 that could then be imported into Gephi. Initially, one *.gdf* file was generated for each project and one for all of the projects combined.

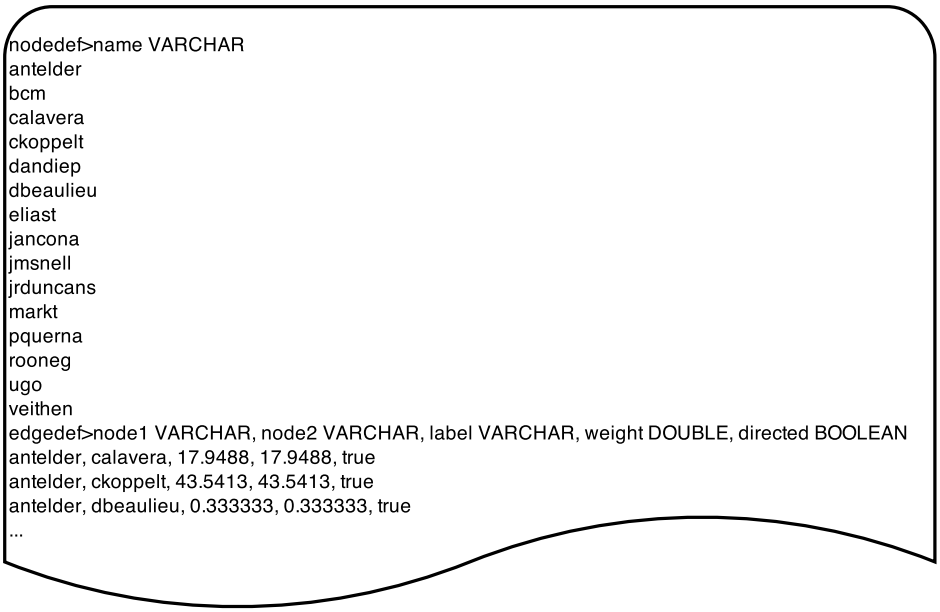


Figure 4.3: The GDF file format

4.4 Analysis

When studying the different calculations provided by Gephi, it was discovered that Gephi ignored edge weights when calculating several network metrics. For some (clustering coefficient) there existed plugins that added the required functionality, but for some metrics (e.g. various centrality indices) there existed no such solution. Instead, it was decided that the tnet package for R was to be used for the calculation of the network metrics. tnet supports weighted calculations for the clustering coefficient and the centrality indices. Modifying the program that generated input for Gephi to create valid input files to R/tnet was a small task, as tnet can import files on the form

```
source \t destination \t weight
```

which is very similar to the core of the Gephi input format. The only real difference between the two is that tnet needed numeric IDs for the source and destination.

Although Gephi does not take edge weights into account when calculating metrics it was still useful for visualizing the networks during the analysis.

Some properties that could affect the results were discovered when analyzing the projects. One example is a committer called 'dev-null' that had made a total of over 30000 commits to only five different projects. Investigating this further it was found that the commits made by 'dev-null' were actually transfer of data from an SVN-repository to a GIT-repository. So the commits done by 'dev-null' were not actual commits and were thereby discarded for the analysis since the amount of commits were significant which could have an impact on the final results.

Various metrics were then extracted from the database and plotted using the tool gnuplot[22]. If any correlation was found, it was then transformed into a linear correlation.

For determining linear correlation between metrics, R's ability to calculate the correlation coefficient was used.

Finally, after noticing the trends in the distribution of the centrality indices, a simple application which calculated the average percentage of most active developers in the projects that had contributed 75% or more of the total number of commits. This was done by continuously picking the most active developer in a project and adding his/her number of commits to a sum until that sum reached 75% or more of the total number of commits in that project.

Chapter 5

Results

In this chapter the results from our analysis is presented.

5.1 Network Metrics in relation to number of committers

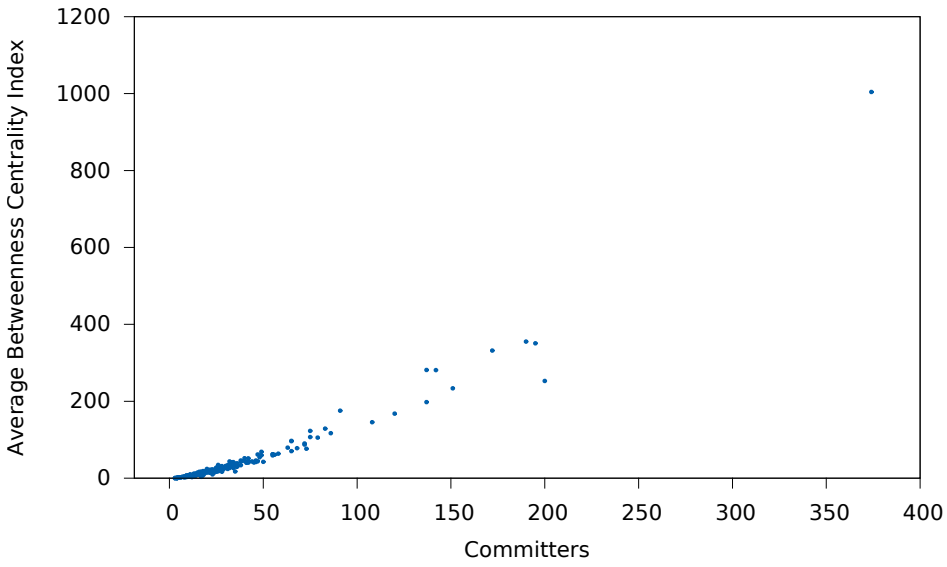


Figure 5.1: Average betweenness centrality over number of committers per project. Each data point represents one project

The correlation coefficients for the graphs showing betweenness (Fig. 5.1), closeness (when both variables are transformed with logarithms) (Fig. 5.2), degree (Fig. 5.3) and strength (Fig. 5.4) are 0.958, -0.960, 0.811 and 0.512 respectively. That suggests that there's no linear correlation between strength and the number of committers.

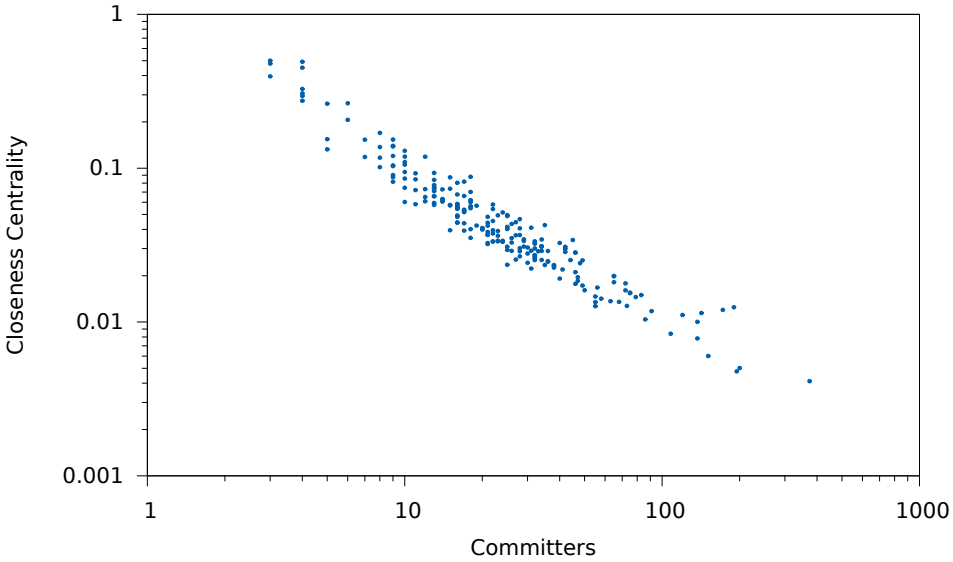


Figure 5.2: Average closeness centrality over number of committers per project. Note the logarithmic scale. Each data point represents one project

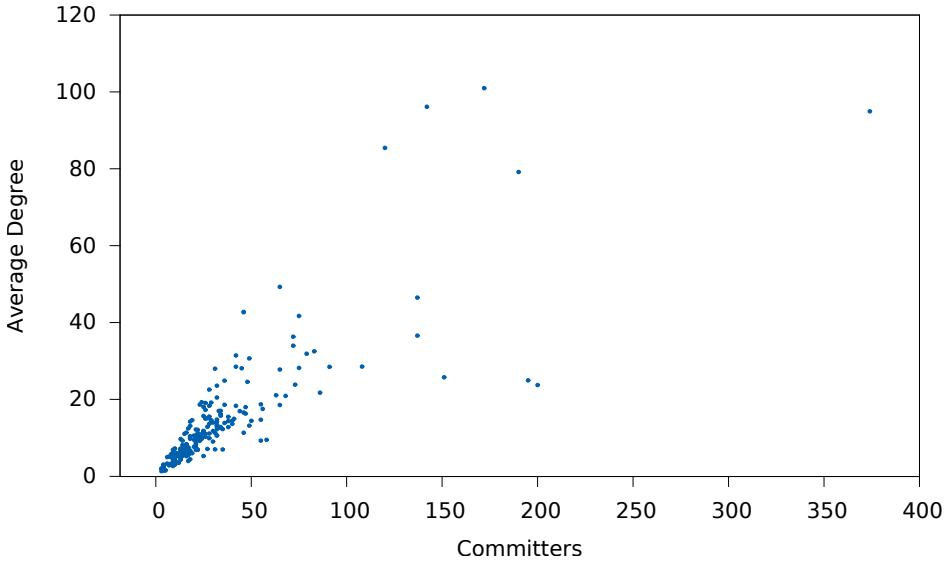


Figure 5.3: Average degree over number of committers per project. Each data point represents one project

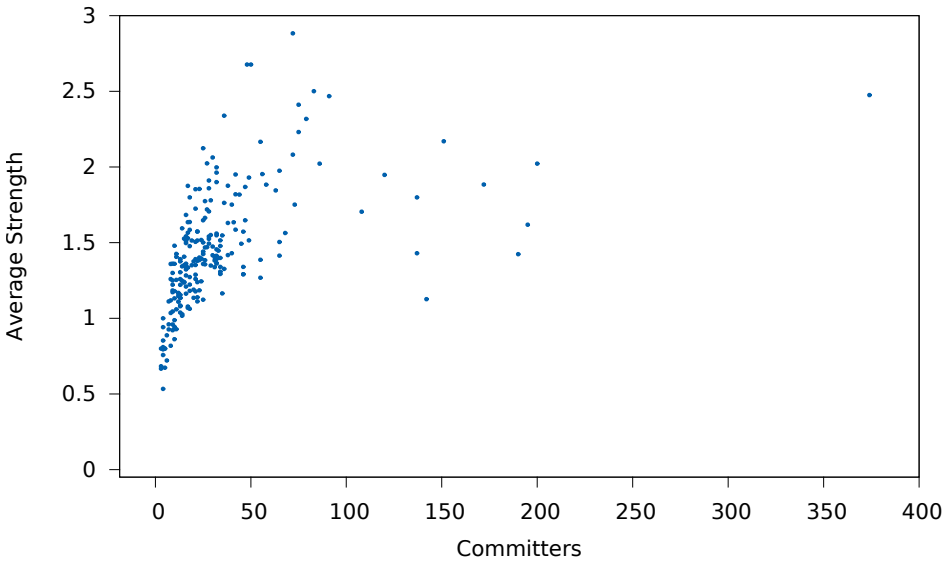


Figure 5.4: Average strength over number of committers per project. Each data point represents one project

5.1.1 Distribution of centrality indices

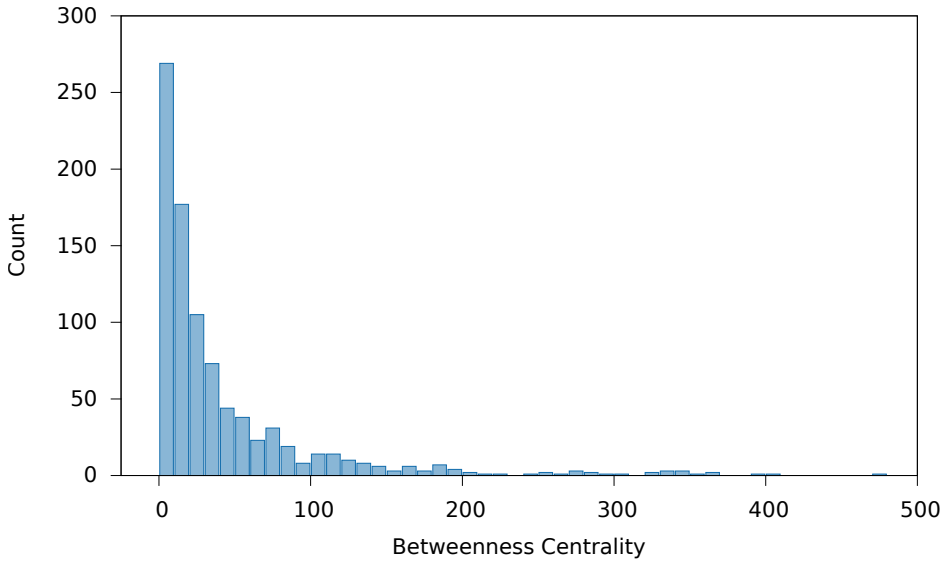


Figure 5.5: Projects with 30 or less committers

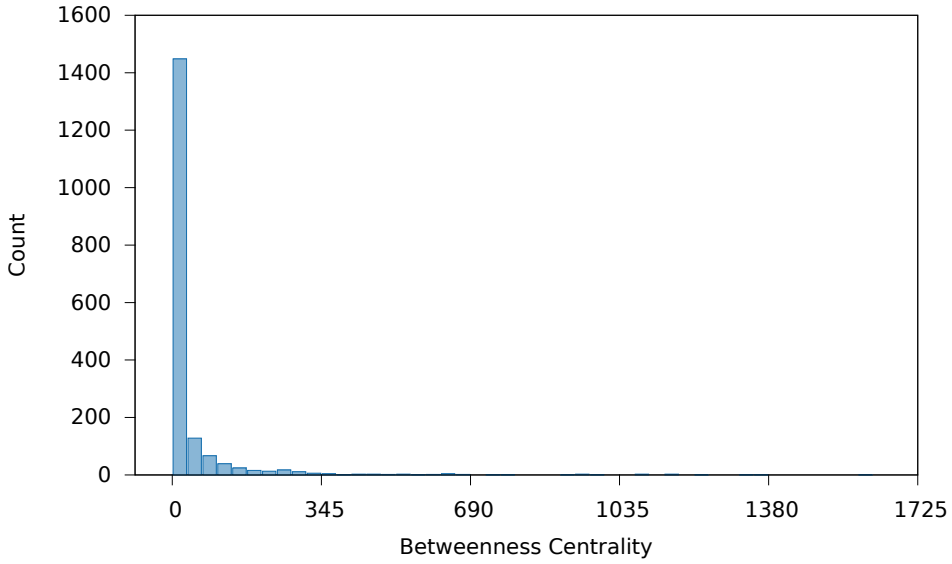


Figure 5.6: Projects with 31 to 50 committers

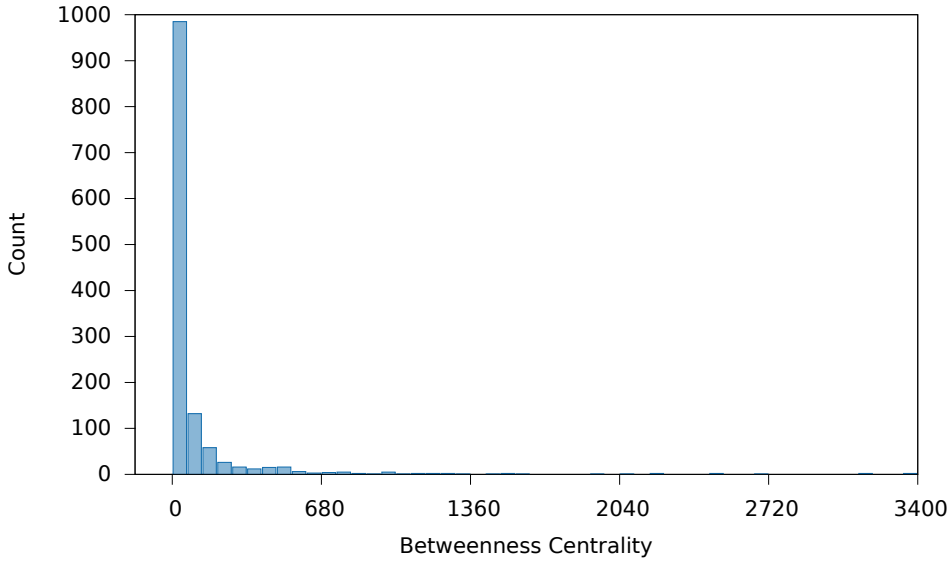


Figure 5.7: Projects with 51 to 100 committers

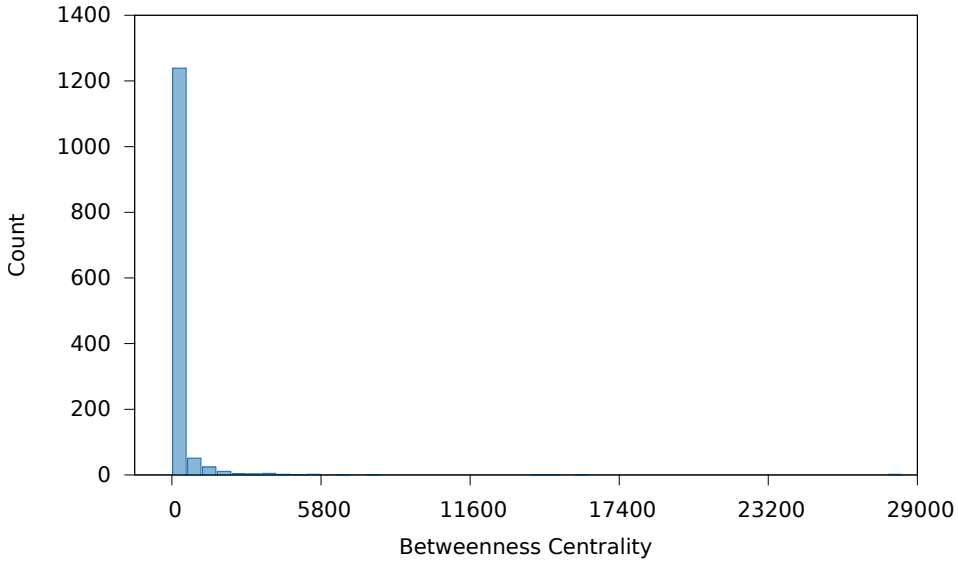


Figure 5.8: Projects with 101 to 200 committers

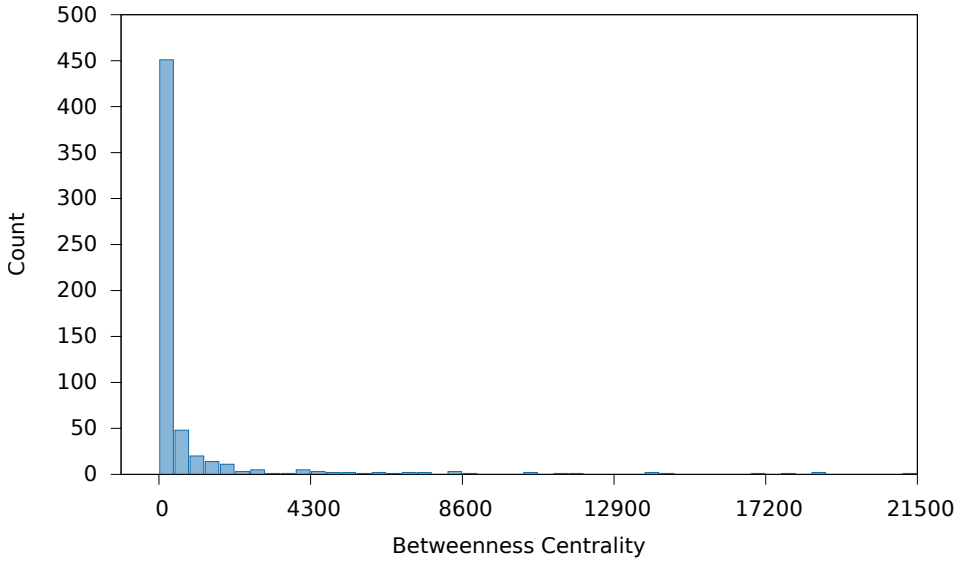


Figure 5.9: Projects with more than 200 committers

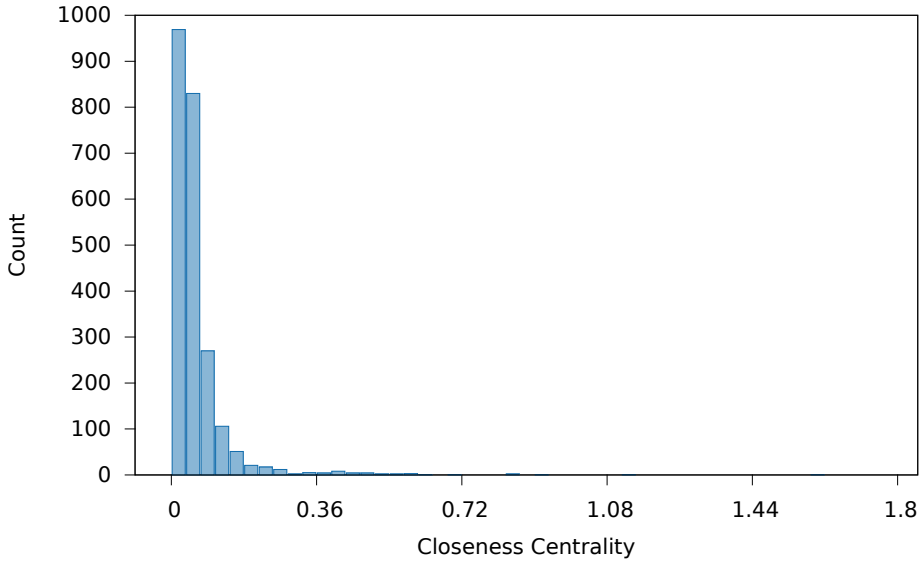


Figure 5.10: Projects with 30 or less committers

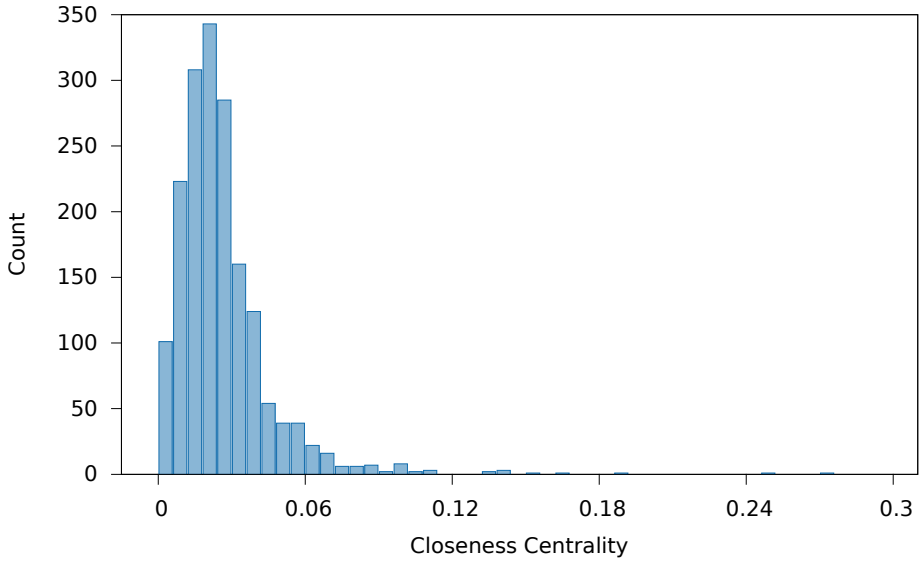


Figure 5.11: Projects with 31 to 50 committers

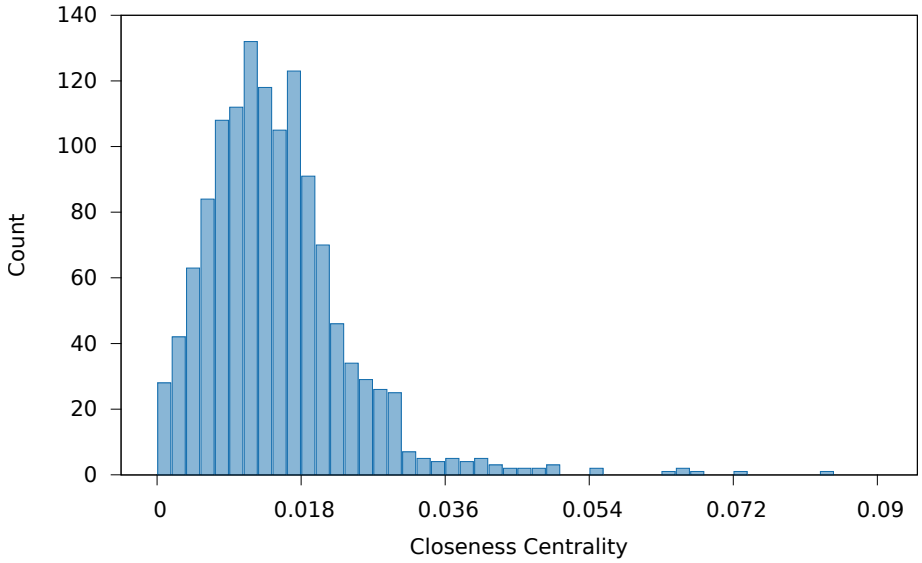


Figure 5.12: Projects with 51 to 100 committers

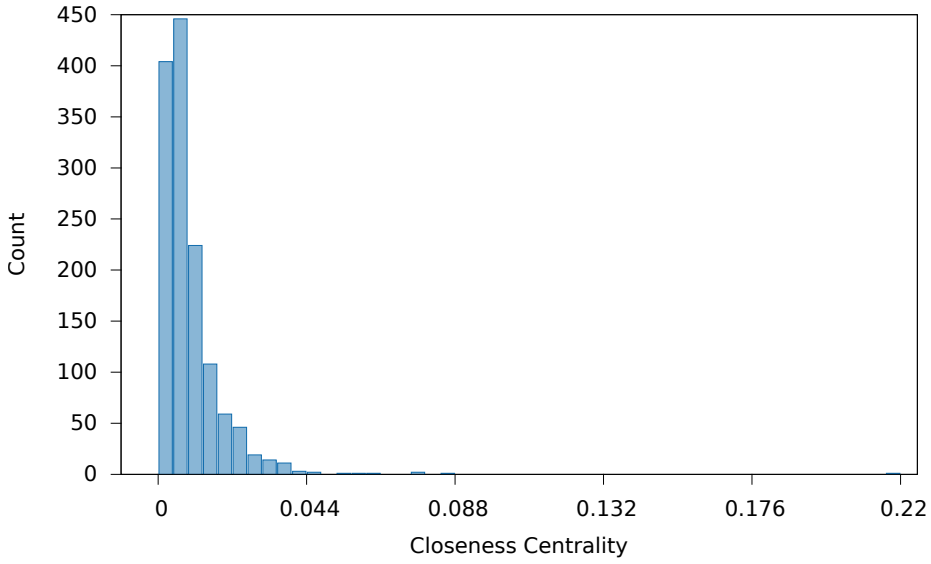


Figure 5.13: Projects with 101 to 200 committers

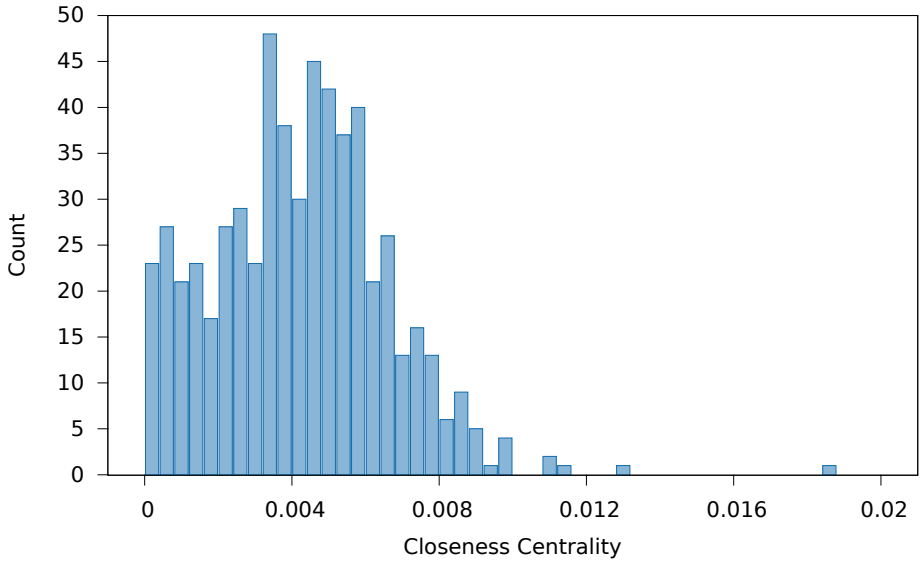


Figure 5.14: Projects with more than 200 committers

5.2 Clustering Coefficient

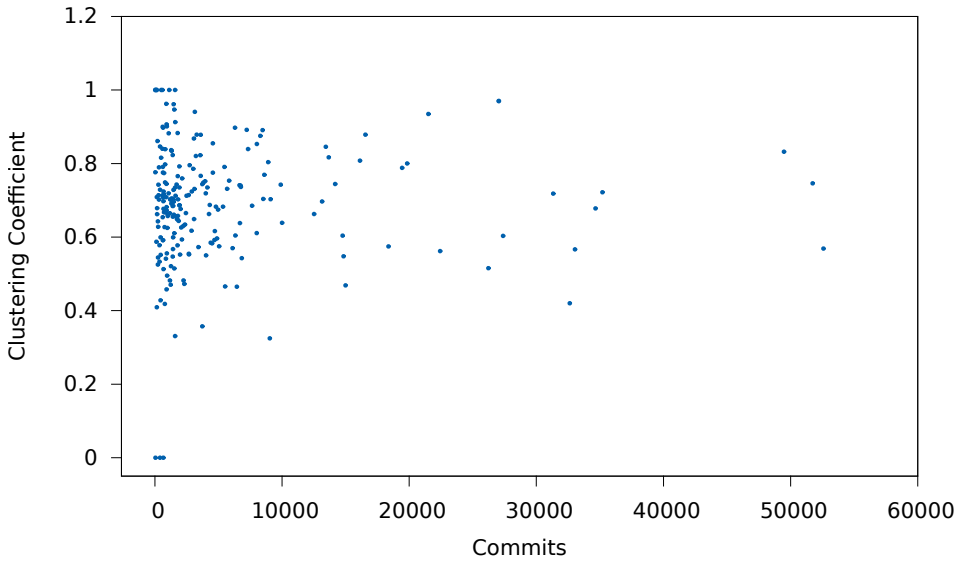


Figure 5.15: Clustering coefficient over number of commits per project. Each data point represents one project

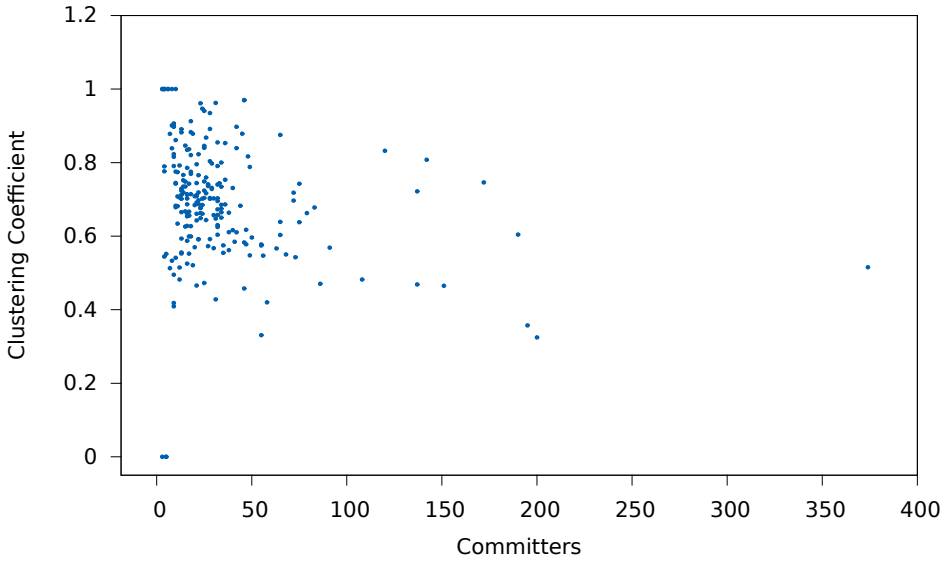


Figure 5.16: Clustering coefficient over number of committers per project. Each data point represents one project

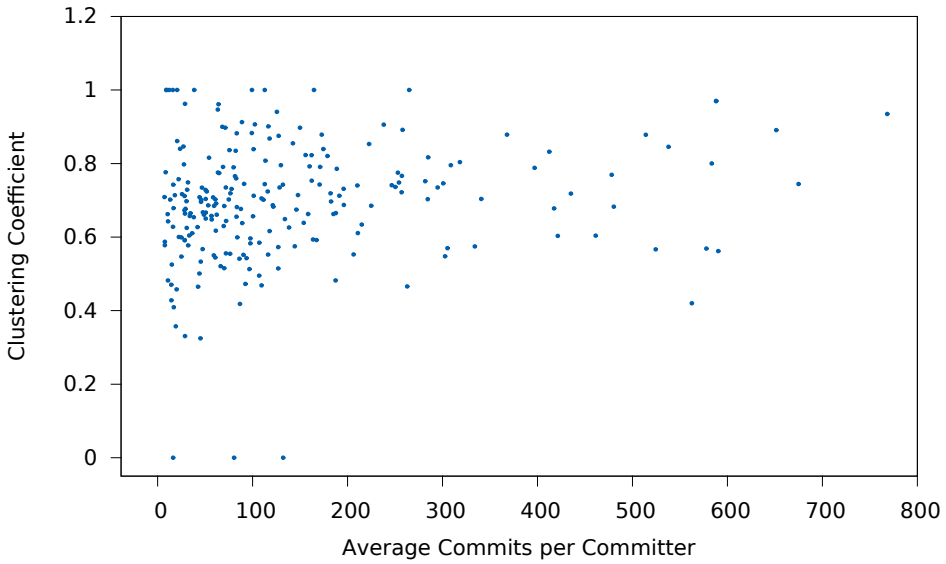


Figure 5.17: Clustering coefficient over average number of commits per developer. Each data point represents one project

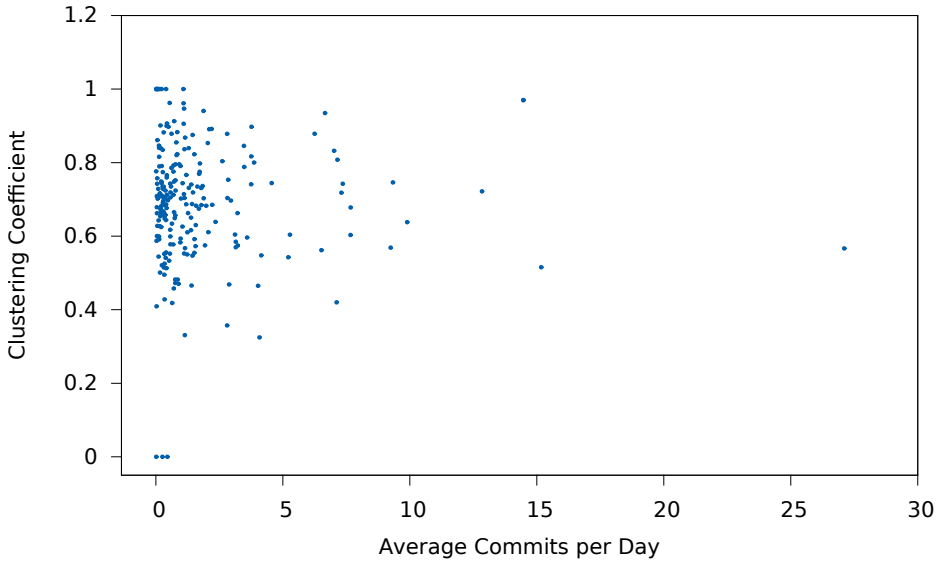


Figure 5.18: Clustering coefficient over average number of commits per day. Each data point represents one project

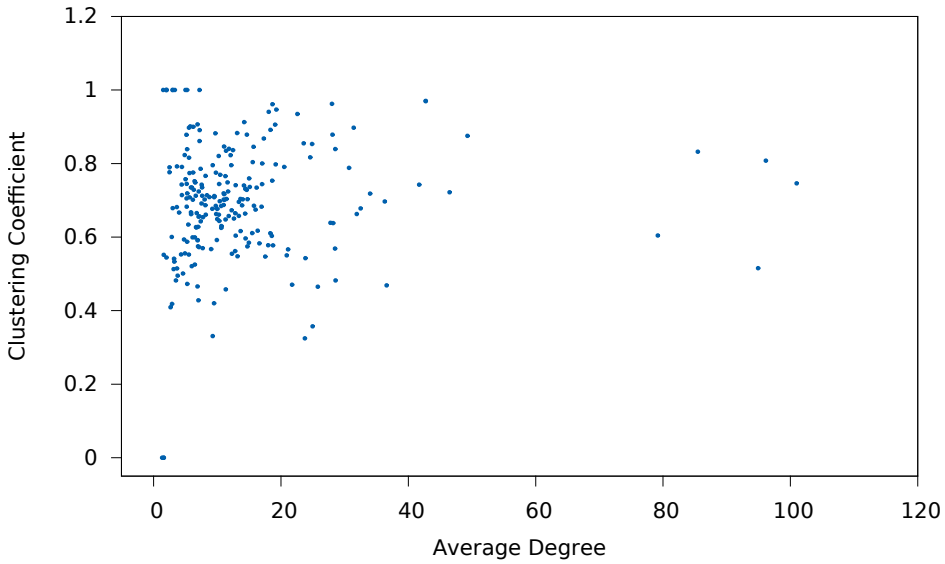


Figure 5.19: Clustering coefficient over average degree. Each data point represents one project

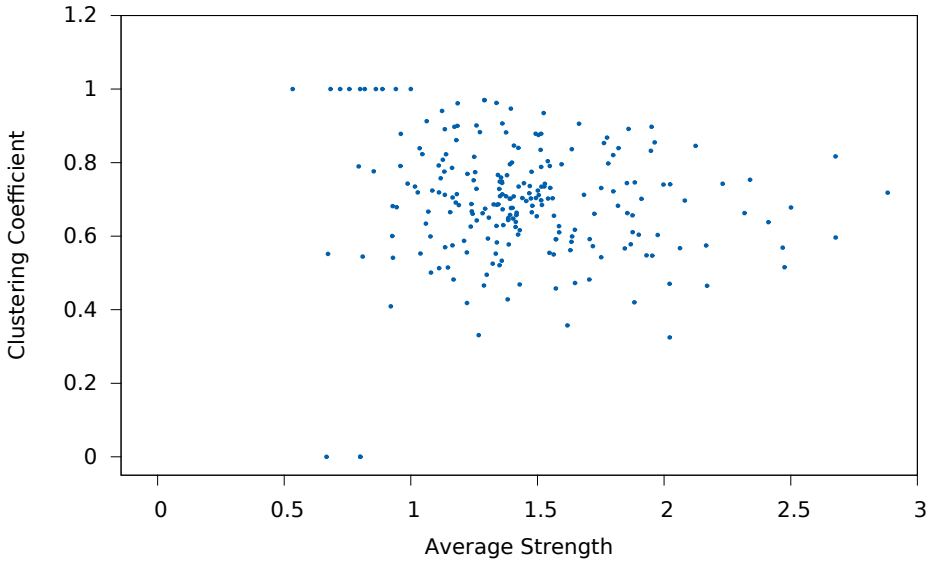


Figure 5.20: Clustering coefficient over average strength. Each data point represents one project

5.3 Metrics for different project-categories

Category	Count	Average Degree	Average Out-Strength
all	228	13.8456	1.4385
big-data	27	15.7087	1.4860
build-management	17	16.9502	1.1942
cloud	10	22.4263	1.5872
content	14	9.8305	1.5362
database	22	23.6451	1.5136
graphics	5	21.0555	1.6763
http	14	29.7259	1.7260
httpd-module	4	24.2663	1.2070
javaee	9	19.0324	1.8808
library	82	10.0269	1.3700
network-client	18	13.9835	1.5277
network-server	35	16.6735	1.5966
retired	9	9.4677	1.5785
testing	4	13.8425	1.2042
web-framework	25	14.8778	1.5341
xml	28	13.7537	1.5609

Table 5.1: Average Degree and Strength for different project categories

Category	Count	Clustering	Closeness	Betweenness
all	228	0.6949	0.0659	39.6982
big-data	27	0.6794	0.0472	54.8562
build-management	17	0.6522	0.1318	49.1389
cloud	10	0.6261	0.0487	156.5869
content	14	0.6493	0.0515	21.3229
database	22	0.7433	0.0379	57.1528
graphics	5	0.7929	0.0419	37.8886
http	14	0.6872	0.0288	89.6320
httpd-module	4	0.7726	0.1703	46.5317
javaee	9	0.6850	0.0304	63.0450
library	82	0.6851	0.0578	24.0122
network-client	18	0.6627	0.0443	38.3626
network-server	35	0.6707	0.0494	42.5121
retired	9	0.7191	0.0575	16.3592
testing	4	0.7400	0.0829	53.5187
web-framework	25	0.7338	0.0734	33.7100
xml	28	0.7430	0.0705	26.8053

Table 5.2: Average centrality metrics and clustering coefficients for different project categories

In Table 5.1 and Table 5.2 we can see that there are no clear correlation between the category of a project and the different network metrics

Chapter 6

Discussion

6.1 Centrality

Both centrality indices has a clear correlation to the number of committers in a project, with their correlation coefficients (0.96 for betweenness centrality and -0.96 for closeness centrality) hinting at a very high probability of linear correlation. It's an odd coincidence that the correlation coefficients are as strong for both centrality indices as they have quite different definitions, but keep in mind that the coefficient for the closeness centrality is calculated after logarithmic transforms. This might be explained by the fact that projects often have a small core of developers that are responsible for most of the commits in a project. In fact, in the projects analyzed in this thesis, on average the 20.3% most active developers contributed more than 75% of the commits. These core developers will have made changes to almost every file in the project while the non-core developers that only make commits occasionally will only have touched a small amount of files. This creates a network structure with a small amount of developers in the centre and the rest in the periphery. The effect of this is that almost all the shortest paths in the developer network passes through a small amount of vertices (developers) giving these vertices a very high betweenness centrality. If we have a network with N committers and a new committer is added to the developer network then there will be N new shortest paths since there is a shortest path from all the ”old” vertices to the ”new” vertex. Almost all of these shortest paths will go through the core developers increasing their betweenness centrality greatly while the ”new” vertex will have a very low betweenness. Thus several vertices will have an increased betweenness centrality, increasing the average for the whole network. The same reasoning can also be used to explain the decrease in closeness centrality with an increase in number of committers. The number of developers in the periphery compared to the number of developers in the core increases with the total number of developers giving the network a lower average closeness centrality overall.

6.1.1 Distribution of centrality

While looking at the distributions of betweenness and closeness centrality for projects with different amount of developers it can be concluded that a very large amount of the developers have a betweenness index relatively small. It is only a very small amount of developers that have a high betweenness centrality but on the other hand their betweenness centrality are in some cases extremely large. This further implies a network structure where a few developers in the core are responsible for a large number of commits while most developers in the periphery of the developer networks only does relatively few commits. Closeness centrality has a similar tendency where the majority of the developers are in the lower half of the observed interval.

6.2 Clustering

No correlation was found between the clustering coefficient and any other metric, such as project age, lines of code, number of committers, average degree or average strength. The clustering coefficients found in the Apache projects also vary widely, from below 0.3 to above 0.9, making it hard to draw any conclusions.

What can be observed in the graphs is that some extreme values for the clustering coefficient can be found in smaller projects (projects with a small amount of committers) where the clustering coefficient is either 1, 0 or very close to them. This can not be seen in projects with more than 20 committers.

6.3 Evaluation

The structure used by Apache where there are several different roles that a person can have might influence the results. It can not be guaranteed that the committer that performs a commit has actually made the changes in the commit. The initiator of a change in the code could be a developer that does not have write access to the code repository, this on the other hand requires that the developer proposes the changes to a committer which then approves if it is considered a valuable improvement to the project. So there is a possibility that a committer has performed several commits that he/she is not the author of. On the other hand, the committer has to critically review the changes and probably discuss with the developer the purpose of the changes. This will make the committer really understand why the changes are needed and because of that he/she will be able to explain to other developers why the changes were made and what purpose they have.

6.4 Average Degree

As can be seen in Figure 5.3 the average degree is very high in many projects. That means the networks are not really sparse and does not really fit into the small-world phenomenon. It could be argued that the method used to create the edges between is too generous in creating connections between developers. It might have been better to limit the connections

(edges) between developers based on e.g. the time between the developers commits or the amount of commits that has been submitted in between.

Chapter 7

Conclusion

This thesis work investigated the developer collaboration in open-source networks with the goal to find common collaboration metrics among the projects. Some common trends were found in the data. The results show that the average betweenness centrality and average closeness centrality is correlated to the number of committers, which might be attributed to the structure of the developer networks where the core consists of a few developers that are responsible for a large proportion of the total number of commits. The distribution of centrality indices in the projects also seems to support this network structure.

It was not possible to draw any conclusions regarding the clustering coefficient. This may be related to the generous amount of edges created, which might create clusters of developers that never really worked much together. Very low clustering coefficients were rare however.

The results of this study has shown that it is possible to find common collaboration metrics in open-source projects, and some of those findings might be useful for benchmarks, such as the distribution of centrality indices within the projects. A large project that have a very low average betweenness centrality compared to the Apache-projects may have an ineffective development process because a low betweenness centrality indicates that most developers seem to do changes on almost all parts of the projects. It seems like the most effective way of developing good quality software, at least in open-source projects, is to have the majority of the developers specialized on a specific part of the project and only a small amount of the developers binding these parts together to the final product.

7.1 External Validity

The projects analyzed in this thesis has many similarities to both open and closed-source development thanks to the Apache Software Foundation's well organized development and thanks to contributions from various companies share developers with companies' closed-source projects, and consists of projects of many types and sizes and the data gathered should therefore be a fairly good representation of the software development industry. Still, they all follow the guidelines set by Apache, and it might have been advantageous to include projects from other sources in order to verify the results external validity.

7.2 Future Work

During this thesis several realizations have been made that could be considered for future studies. The networks created in this study did not take the difference in time between commits on the same files into account when creating the edges. Changes that are close in time are probably more relevant than changes that are several years apart. So to only create edges between developers that have made changes to the same files close in time could be a consideration for future studies.

It could also be interesting to study how the metrics change during a projects lifetime. The collaboration may be different in a new project compared to a project that has been worked on for several years.

There were an intention to include project size in terms of lines of code as a metric in this study. But it was concluded that the method used for extracting lines of code for each project gave questionable results, so the decision was made to skip lines of code as a metric. Future studies in the subject may want to use lines of code as a metric since it is a good representation of how large and complex a project is.

Bibliography

[1] *About Git*. March 2015. URL: http://git-scm.com/about.

[2] *Apache Hadoop*. May 2015. URL: https://hadoop.apache.org/.

[3] *Apache HTTP Server*. May 2015. URL: http://httpd.apache.org/.

[4] *Apache Incubator*. April 2015. URL: http://incubator.apache.org/.

[5] *Apache Incubator Graduation Requirements*. April 2015. URL: http://incubator.apache.org/incubation/Incubation_Policy.html#Graduating+from+the+Incubator.

[6] *Apache license*. March 2015. URL: http://www.apache.org/licenses/LICENSE-2.0.

[7] *Apache OpenOffice*. May 2015. URL: https://www.openoffice.org/.

[8] *Apache software foundation*. February 2015. URL: http://www.apache.org/foundation/.

[9] *Apache Software Foundation index*. February 2015. URL: http://projects.apache.org/indexes/alpha.html.

[10] *Apache Subversion Features*. April 2015. URL: https://subversion.apache.org/features.html.

[11] A. Barrat, M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. "The architecture of complex weighted networks". In: *Proceedings of the National Academy of Sciences of the United States of America* 101.11 (2004), pp. 3747–3752. DOI: 10.1073/pnas.0400087101.

[12] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: An Open Source Software for Exploring and Manipulating Networks". In: *International AAAI Conference on Weblogs and Social Media* (2009), pp. 361–362.

[13] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. "Latent social structure in open source projects". In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM. 2008, pp. 24–35.

[14] Michael Bloch, Sven Blumberg, and Jurgen Laartz. "Delivering large-scale IT projects on time, on budget, and on value." In: *Mckinsey Quarterly* (Oct. 2012).

[15] Ulrik Brandes. “A faster algorithm for betweenness centrality”. In: *Journal of Mathematical Sociology* 25.2 (2001), pp. 163–177.

[16] Kevin Crowston. “The social structure of open source software development teams”. PhD thesis. Syracuse University, 2003.

[17] *Curl*. March 2015. URL: http://curl.haxx.se/.

[18] Boru Douthwaite, Andrea Carvajal, Sophie Alvarez, Elías Claros, and LA Hernández. “Building farmers’ capacities for networking (Part I): Strengthening rural groups in Colombia through network analysis”. In: *KM4D Journal* 2.2 (2006), pp. 4–18.

[19] *Eclipse Community Survey 2014*. February 2015. URL: https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/.

[20] Linton C. Freeman. “A set of measures of centrality based on betweenness”. In: *Sociometry* (1977), pp. 35–41.

[21] Linton C. Freeman. “Centrality in social networks conceptual clarification”. In: *Social networks* 1.3 (1979), pp. 215–239.

[22] *gnuplot*. May 2015. URL: http://www.gnuplot.info/.

[23] Michael W. Godfrey and Qiang Tu. “Evolution in open source software: A case study”. In: *Proceedings of the International Conference on Software Maintenance, 2000*. IEEE. 2000, pp. 131–142.

[24] Umberto Griffo. *Weighted Cluster Coefficient - Gephi Marketplace*. March 2015. URL: https://marketplace.gephi.org/plugin/weighted-cluster-coefficient/.

[25] Robert A. Hanneman and Mark Riddle. *Introduction to social network methods*. University of California Riverside, 2005.

[26] *How the ASF works*. March 2015. URL: https://www.apache.org/foundation/how-it-works.html.

[27] Matthew O. Jackson. *An Overview of Social Networks and Economic Applications*. 2009.

[28] Alden S Klovdahl. “Social networks and the spread of infectious diseases: the AIDS example”. In: *Social science & medicine* 21.11 (1985), pp. 1203–1216.

[29] Vito Latora and Massimo Marchiori. “Economic small-world behavior in weighted networks”. In: *The European Physical Journal B-Condensed Matter and Complex Systems* 32.2 (2003), pp. 249–263.

[30] Josh Lerner and Jean Triole. *The Simple Economics of Open Source*. Working Paper 7600. National Bureau of Economic Research, Mar. 2000.

[31] L. Lopez-Fernández, G. Robles, J. Gonzalez-Barahona, and I. Herraiz. “Applying Social Network Analysis Techniques to Community-Driven Libre Software Projects”. In: *International Journal of Information Technology and Web Engineering* 1 (2008), pp. 28–50.

[32] Gregory Madey, Vincent Freeh, and Renee Tynan. “The open source software development phenomenon: An analysis based on social network theory”. In: *Proceedings of the American Conference of Information Systems*. 2002, p. 247.

[33] Stanley Milgram. “The small world problem”. In: *Psychology today* 2.1 (1967), pp. 60–67.

[34] Audris Mockus, Roy T. Fielding, and James Herbsleb. “A case study of open source software development: the Apache server”. In: *Proceedings of the 22nd international conference on Software engineering*. ACM. 2000, pp. 263–272.

[35] *Netcraft survey*. April 2015. URL: `http://news.netcraft.com/archives/category/web-server-survey/`.

[36] Mark EJ. Newman. “Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality”. In: *Physical review E* 64.1 (2001), p. 016132.

[37] Christopher Oezbek, Lutz Prechelt, and Florian Thiel. “The onion has cancer: Some social network analysis visualizations of open source project communication”. In: *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. ACM. 2010, pp. 5–10.

[38] Tore Opsahl and Pietro Panzarasa. “Clustering in weighted networks”. In: *Social networks* 31.2 (2009), pp. 155–163.

[39] Alma Orucevic-Alagic and Martin Höst. “Network analysis of a large scale open source project”. In: *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE. 2014, pp. 25–29.

[40] *R project*. April 2015. URL: `http://www.r-project.org/`.

[41] Eric Raymond. “The cathedral and the bazaar”. In: *Knowledge, Technology & Policy* 12.3 (1999), pp. 23–49.

[42] Charles Severance. “The Apache Software Foundation: Brian Behlendorf”. In: *Computer* 45.10 (2012), pp. 8–9.

[43] *Tnet*. April 2015. URL: `http://toreopsahl.com/tnet/software/`.

[44] Duncan J. Watts and Steven H. Strogatz. “Collective dynamics of ‘small-world’networks”. In: *nature* 393.6684 (1998), pp. 440–442.

Appendices

Appendix A

Script for calculating metrics with tnet

```
library("tnet")

files <- list.files("graphs", full.names=TRUE)

calc_metrics <- function(file) {
        print(file)
        data <- read.table(file)
        degree = degree_w(data)
        betweenness = mean(betweenness_w(data
            )[,'betweenness'])
        closeness = mean(closeness_w(data,
            directed=TRUE)[,'closeness'])
        clustering = clustering_w(data)
        return(t(c(basename(file),
                   mean(degree[,'degree']),
                   mean(degree[, 'output']),
                   betweenness,
                   closeness,
                   clustering)))
}

if (file.exists("output")) {
        file.remove("output")
}
colnames <- c("Name", "Degree", "Strength",
              "Average Betweenness Centrality Index",
              "Average Closeness Centrality Index",
              "Clustering Coefficient")
metrics <- lapply(files, calc_metrics)
y <- do.call(rbind, metrics)
```

```
write.table(y, "output", sep='\t',
    col.names=colnames, quote=FALSE, row.names=FALSE)
```

A.1 Excluded Projects

We were unable to retrieve the commit logs for the following projects:

- Apache Lucene.Net
- Apache Camel
- Apache Wink
- Apache Struts
- Apache log4cxx
- Apache Tuscany
- Apache Olingo
- Apache Empire-db
- Apache Taverna
- Apache Marmotta
- Apache Spark
- Apache Syncope
- Apache Cordova
- Apache Axiom
- Apache Jena
- Apache Spatial Information System
- Apache ActiveMQ
- Apache Allura

The following projects were too small to create any meaningful networks:

- Apache .net Ant Library
- Apache Compress Ant Library
- Apache ORO

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2015-06-04

EXAMENSARBETE Social Network Analysis of Open Source Projects
STUDENTER Nicklas Johansson, Christian Tenggren
HANDLEDARE Alma Orucevic-Alagic (LTH)
EXAMINATOR Martin Höst (LTH)

Social network analysis of open source projects

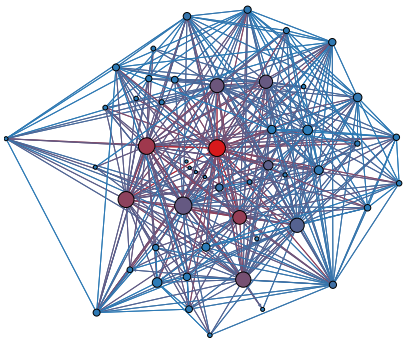
POPULÄRVETENSKAPLIG SAMMANFATTNING **Nicklas Johansson, Christian Tenggren**

By applying social network analysis on the collaboration of open-source developers for a wide variety of projects a few observations can be made that can give some valuable insight in the development process of open-source projects.

Motivation
In software development, open source projects are projects where the code is freely available for anyone to read and copy. These projects are developed through a decentralized structure, where anyone capable is able to submit changes and improvements to the codebase. This development process differs heavily from the way most companies do their development, yet it is still many times very successful and can produce products of considerable quality. As open source software is commonly used in many closed projects, they also get contributions from many companies. This makes them an interesting target for analysis of the development process as there's a large amount of data openly available, contrary to projects developed in closed environments.

This led us to perform a study on a large amount of open-source projects hosted by a well-known open-source community called Apache Software Foundation. The foundation hosts many projects with a wide variety of sizes, including several high profile projects such as OpenOffice.

Social network analysis
To analyze developer collaboration we have used the concept of social network analysis. This is done by studying networks representing developers and their collaboration. Performing a social network analysis means applying different metrics to the networks, where each metric looks at a specific property of the networks. Examples of metrics are the number of other developers a specific developer has collaborated with and centrality measures that shows the influence developers have over one another. The clustering coefficient is another metric that calculates if there exists subgroups in the developer networks. A subgroup is a set of vertices that are well connected to each other. Well connected means that each vertex in the set has an edge to a majority of all the other vertices in the set.



The developer network for Apache OpenOffice

Result
A few conclusions can be drawn from the data gathered from the various projects. The more developers a projects has, the higher the average centrality is. The average centrality follows a line very closely, suggesting that the networks have a similar structure. The majority of the developers have a small centrality index while only a few developers are very central in the projects. On the other hand, we found no real correlation between the clustering coefficient and any other data gathered throughout the study.