WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

Master Thesis

Collaboration networks in open-source software development

Jozsef Csepanyi

Date of Birth: 06.09.1996
Student ID: 11927479

Subject Area: Information Systems

Studienkennzahl: h11927479

Supervisor: Johannes Wachs

Date of Submission: 02. April 2021

Department of Information Systems and Operations, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

DEPARTMENT FÜR INFORMATIONS-
VERARBEITUNG UND PROZESS-
MANAGEMENT DEPARTMENT
OF INFORMATION SYSTEMS AND
OPERATIONS

Contents

List of Figures

List of Tables

Abstract

Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus....

1 Introduction

In recent years open source software solutions have become widely popular and frequently used in both scientific and enterprise use, which can be attributed to a number of factors, most importantly the ease of development and deployment of IT projects, improved cybersecurity and enhanced scalability [19]. This increases the contribution to open source projects from enterprises and individuals alike. Due to its nature, open source software projects are driven by community contributions, and depend heavily on active participation in all phases of the project.

Software development in a corporate environment usually follows a strict hierarchial structure, where each participant is given a precise position and responsibility, like project manager, scrum master, senior or junior developer, and employees do not tend to work outside of their assigned tasks and territories. The main purpose of maintaining software development structures is for the company to ensure that the outcome of the project is in accordance with the business objective, adheres to the pre-set quality criteria and it is completed in a given timeframe; in other words to asses the risks associated with the business objective of the software project [21]. This is achieved by breaking down the developed software into smaller, less complex components, and grouping the developers into managable teams, where the communication is moderated between teams [5].

As opposed to commercial software development, Free/Libre Open Source Software (FLOSS) projects usually do not follow an organizational hierarchy, and are usually self-organizing and dynamic [5]. Issues, bugs and progress are tracked openly, and everyone is encouraged to contribute based on the current topics and expertise, but purely on a volunteering basis. The lack of access restriction to certain modules allows for much more spontaneous interaction between developers, which generate large, complex networks [15]. These complex networks can be seen as large social networks of developers based on collaboration.

Because contribution to FLOSS projects are voluntary, participants have a different motivation for taking part than in commercial software development. According to El Asir et al. [8], FLOSS participation can be motivated by internal and external factors. Internal factors include self-improvement,

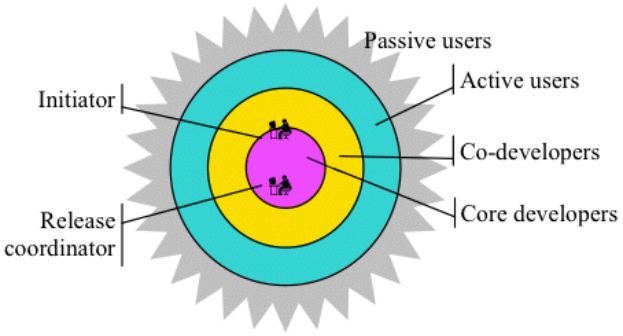

Figure 1: Onion model of collaboration types in FLOSS projects [6].

learning and contribution as a hobby or pass-time activity [1, 23], whereas external factors are motivated by marketing and demonstrating certain skills, thus increasing and improving employability [1].

2 Background and rationale

2.1 Collaboration in FLOSS projects

Collaboration networks of open source software (OSS) have been a subject of many academic research. Raymond [6] has defined collaboration based on bug report interaction, and observed the collaboration network of 124 large-scale SourceForge projects. The generated networks have widely different centralization properties, but it was observed that larger sized projects tend to be more decentralized. The broad community roles contributors tend to take have been also identified in [6], which have been coined as the *onion model* in [15] (Figure 1).

The onion model describes the types of participants in an OSS project as layers. The center represents the small group of core developers, who are responsible for the majority of contributions to the software. They are surrounded by a larger group of co-developers, whose main contributions are usually bug fixes reported by the active users. The passive users are usually the largest in numbers, who do not contribute or report any bugs. In a healthy FLOSS project, each layer of contributors are about one magnitude larger in numbers than the preceeding inner layer [18].

El Asir et al. [8] used a K-means classification to categorise project par-

ticipants into a similar core-periphery structure (core, gray in-between area, and periphery) based on SNA metrics with a montly timeframe, and analysed how and why contributors transition between groups. They found that technical contributions like code commits and lines added have a much heavier impact on becoming a core developer as opposed to other activities, such as testing, reviewing and commenting.

A literature review conducted by McClean et al. [17] systematically analysed the state-of-the-art research of 46 scientific papers in the field of FLOSS social networks, and categorised them into three groups based on topic: structure, lifecycle and communication. They conclude, that the existence of core-periphery structure in OSS projects is well established in the field, which is also an indicator of a healthy FLOSS software. Regarding the lifecycle, generally the core development team does not change significantly over time, however, the project becomes more decentralised and distributed as it matures. A lack of research regarding temporal analyses were identified in the most current knowledge, which was suggested as a future research area in this field.

2.2 Social network of Open-source projects

In a larger FLOSS project, developers usually cannot understand every part of the project, therefore collaboration is required with eachother. The network created by the collaborators can be considered as a social network, because collaboration requires some kind of social interaction with eachother. Social network theory describes how social interaction patterns affect the individual behaviour [16]. We can model an OSS project's social network as a graph, where nodes represent collaborators (developers, bug reporters, etc...) and edges represent the social interaction between them. In mathematical terms...

The type of the social interaction determines the created network, therefore choosing the basis of collaboration can have a significant impact on the network structure. The common types of developer social networks (DSNs) are Version Control System-based (VCS-DSN), Bug Tracking System-based (BTS-DSN) networks and DNSs, that are purely based on social elements [2]. The VCS-DSN take the version control application as a source for network generation by recording collaboration based on co-edits of the same module, file or code section. Choosing the granularity can impact the precision of true collaborations represented in the network. Co-edits by multiple

developers to a single module or file does not necessarily mean actual collaboration was required from the authors, as the parts edited could work functionally independent from eachother. By increasing the granularity to file sections (classes or functions within a single file) or even lines, we can be more certain, that coordination was required, but we risk leaving out semantically connected parts of the project [12]. In contrast to VCS-DSNs' purely technical approach, the BTS-DSNs use semi-technical bases for connecting participants, such as comments on issues, bugs or reviews [8]. These artifacts, although being tightly related to specific sections of the source code, allow for taking into account conversational elements as contribution. For example, participants, who do not contribute directly to the software source code, but actively review and comment, are also considered. Lastly, social networks of developers can be constructed on project participation, following, starring or through communication means like mailing lists. The technical aspect of collaboration is minimized in such DNSs, and they are more fit for project organization and communication analyses in FLOSS projects (mailing list vs file file edits vs line edits)

2.3 OSS project success

community maturity ([14] in [2])

"Successful projects will likely have modular structure from the start or after refactoring as the source code grows larger and more unwieldy" [3]

success factors: Average Time Efforts, Number of Developers, Comments, Total Code Lines, Comment Ratio, Number of Rater [22] Truck Factor [4]

3 Motivation of research problem and research questions

Because there is a high dependency on the community in open source software projects, by understanding how contributions are included and what patterns emerge we can gain valuable insight into the project's current state and its trajectory. As stated before, SNA analysis of OSS have been extensively studied, but there is a lack of research regarding temporal models analyzing the lifecycle of a FLOSS project.

The goal of this paper is to fill in this gap by examining OSS project collaboration networks over time using SNA metrics. More specifically, one part of the research will focus on the evolution of such collaboration networks and comparing and contrasting these networks with the software outcome. The second part will focus on events during a project, and how it affects the developer collaboration. The research questions, which are broken down into subquestions, are as follows:

1. **How does the temporal lifecycle information of a project influence its success?**

   (a) *Based on temporal models of collaboration, is it possible to predict the outcome of the project?* Since it has been proven that the core collaborators do not change much over the course of the OS software development, our assumption is that any sudden or long-term change, that is not consistent with the other observed projects, can have a significant impact on the outcome (negative or positive alike).

   (b) *Can stages of a FLOSS project with a maturity model be observed?* As most OS software starts with a small collaborator basis and grows over time, it can be assumed, that each project goes through the same steps of open source maturity levels. On the other hand, it is also possible that due to the uniqueness of each project, no such stages are observable.

2. **How do major events in the project lifecycle change the collaboration network of the project?**

   (a) *Do planned or foreseeable events change the collaboration structure?* Major software version releases can be considered foreseeable events of the project lifecycle, which could have an effect on the developer collaboration. For example, there might be a higher rate of interaction between contributors just before a new version is released to clear up the backlog of tasks. But it is also possible, that commit and change rates drop during this time, because the focus shifts to stability and testing instead of new features.

   (b) *How unforeseeable internal or external events affect FLOSS collaboration?* Sudden shocks to the project, such as an announcement of disinterest from major users of the software, discontinued enterprise support of the project, large-scale global events like the

pandemic, or sudden employee firings can have significant effect on the core and periphery collaborators alike. By analysing the collaboration network before, during and after such changes, we might be able to recognise patterns, that regularly occur around these events.

3.1 Research methodology

To find answers to the research questions above, first we build a repository analyzer tool, which mines collaboration data from FLOSS projects, generates static snapshot collaboration networks at each given time interval and calculates SNA metrics for each snapshot. Then these metrics can be aggregated over time, or plotted against time to discover changes in the network. The `git2net`[1] [9] Python library provides the necessary tools to mine any project repository that uses git version control. It also incorporates temporal network generation capability, which can be used as a source for creating static collaboration networks aggregated over a given period of time.

We apply a hybrid methodology of qualitative and quantitative research. First, as part of the qualitative research, we choose a small number of repositories to be analyzed. We observe the number of connected components, centrality, number of nodes and mean degree SNA metrics in order to discover the core and peripherial collaborators over the project lifecycles. The basis of collaboration, due to the unavailability of other means of communication, is coediting files. Based on the state of the art research in this field, file coediting proves to be an effective and easy way to represent collaboration between developers.

After discovering the collaboration structure over time, we will match the breakpoints and unexpected spikes or troughs to events within the lifespan of the project. We expect that the key SNA metrics will show a periodicity around planned releases and other reoccurring events (e.g. holiday season). Outstanding values without reoccurrence, on the other hand, are more likely to be consequences of unexpected events. In these cases, it should be observed whether the network is capable of reorganizing itself, or does the event leave a permanent mark on the collaboration structure. A categorization of unexpected events and the level of impact each category has should be observed.

For the quantitative research to be conducted, we will gather a large set

[1] `https://github.com/gotec/git2net`

of repositories along with major events in its lifecycles. We will then run the miner for all repositories, and with the findings of the qualitative research, we will try to detect all major events and their type (planned or unexpected). We will utilize the `ruptures` [2] library to detect changes in the continuous SNA metrics. If the model is capable to accurately recognise events, then we can also apply it on any repository to detect changes, which will allow us to discover changes in the collaboration network that are not related to publicly known events or releases.

4 Gitminer implementation

To find answers to the research questions, we implement an analysis tool to mine and analyze project repositories, which allows us to generate collaboration networks and network metrics for the analysed projects.

4.1 `git2net` miner

The process begins with the project mining. After cloning the repository, the `git2net` [9] library is used to collect data related to commits. Specifically, who is the author of each commit, which files were modified (created, edited, deleted) with the commit, and when was the commit created. Additionally, the lines edited by the author within each commit are collected separately, allowing for a more fine-grained collaboration network generation if necessary. The results are collected into an `sqlite` [3] database file's *commits* and *edits* tables.

The `git2net` mining process by default collects all the commits throughout the project's lifecycle. However, the processing time of each commit differs based on the number of edits, the affected number of files and the file types as well, which makes collecting certain commits very resource-intense and time-costly. Therefore, we exclude every commit, which contains more than 100 file modifications, during each repository mining using the *max_modifications* parameter. As observed by Gote et al [9], this exclusion criteria does not affect significanly the generated network, because they are mostly merge commits or project restructurings, which do not mark any

[2] `https://github.com/deepcharles/ruptures`
[3] https://www.sqlite.org/index.html

true collaboration effort between developers. During the data mining in certain repositories, we encountered commits, that were not mineable with this method and the mining process halted, presumably due to processing error because of binary file changes in these commits. We also excluded these commits from our data mining process.

This exclusion criteria resulted in an average of 3% of commits excluded in all repositories subject to our analyses, with the highest excluded commit rate being 20%.

4.2 `repo_tools` miner

We use the `repo_tools` [4] Python library to query the Github API for additional repository data extraction, such as:

- Releases
- Tags
- Issues
- Stars and followers

The mining output is also stored in a `sqlite` relational database, which is queried later on during the analysis.

4.3 Data preprocessing

The collaboration networks with the `git2net` library connect the authors to their edited files using only file and author names instead of IDs. This creates an issue when generating the networks, because authors with the same name will show up as one node, and they will be connected to the files they touched combined. Furthermore, authors that change their displayed name ('author_name' field in the mining database) or log in from diferent accounts, where they have different names, will show up as multiple nodes instead of a single vertex.

We utilize the `gambit` [10] rule-based disambiguation tool to resolve the author names. Furthermore, the created networks have issues when the node names contain special characters or spaces. Therefore, after disambiguation,

[4] https://github.com/wschuell/repo_tools/

we replace every unique author name with its ID number.

As the files are also labelled by their filename property in the network outputs, the same filenames but in different folders are also displayed as single nodes. In order not to create false collaborations, we simply remove the files from the network with filenames, that occur more than once in all the repository subdirectories. We argue that this does not remove any significant collaboration data, since most files sharing their name with other files are technicaly files, like `__init__.py` for a Python project.

4.4 Collaboration networks

When creating a DSN from the mined data, we have multiple methods at hand. The `git2net` library provides its own co-editing network function, which returns a temporal network of collaborators. This uses the co-authorship algorithm developed by Gote et. al. [9], however, we would like to have more control over the network generation method, such as simple file-based co-authorship in order to customize the network for our needs, like weighing each relation or generating undirected graphs.

4.4.1 Temporal bipartite network

As a first step, we generate a temporal bipartite network of authors and their edited files with the `git2net` built-in *get_ bipartite_ network* method. A temporal network is a `pathpy` [5] graph object, which contains a collection of timestamped graphs of a single network at each point in time within the observed timeframe. Such a snapshot $S_t = (U, V, E_t)$, where U is the set of authors, V is the set of files and E_t is the set of file edits as edges at t timestamp. By connecting the authors, who touched the same files, and removing the nodes representing the edited files (converting the bipartite network to a regular network), we can observe the evolution of the collaboration over time, represented in Figure 2.

Although a temporal network preserves the time aspect of the graph by the edges being tied to the time dimension of the graph, calculating network metrics like centrality on such networks is infeasible. Visualization also proves to be difficult in representations where animation is not possible.

[5] https://www.pathpy.net/

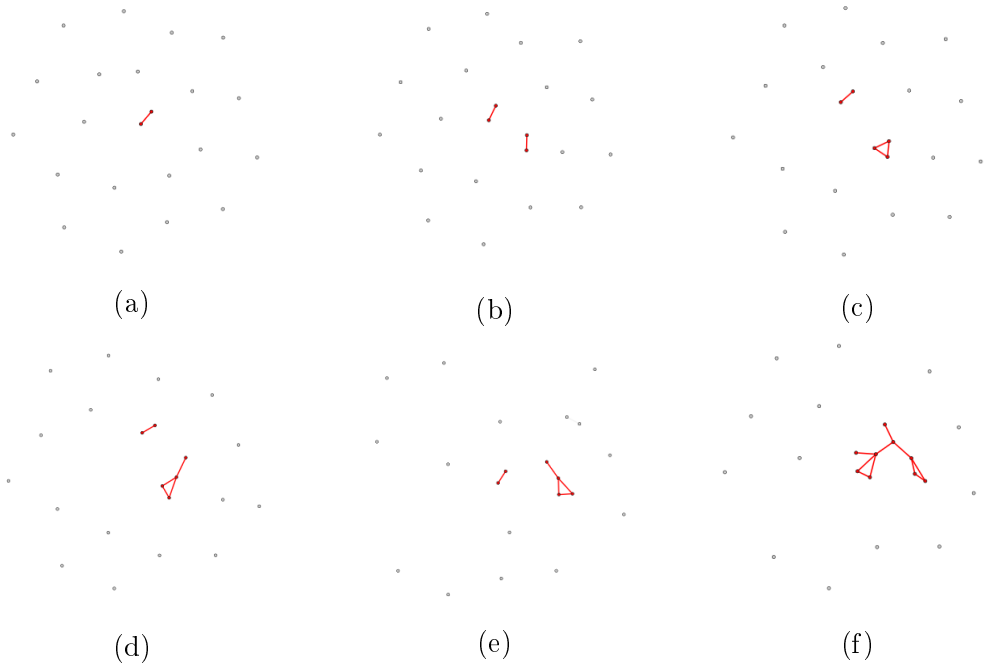

Figure 2: Sequential snapshots of the `networkx` collaboration network with a moving time-window of 30 days and 7-day steps.

Therefore, we aggregate the bipartite network over a given timeframe into a static network. All nodes within the temporal net are preserved, and all directed edges are added to the network with the edge weight representing how many times that author edited the file.

4.4.2 Static networks

The generated static weighed bipartite network looses its time-varying component, but now we are able to manipulate and calculate complex statistics over it. Figure 3 is an example of such a network. As a next step, we convert the the bipartite network into an authors' network by removing the nodes representing files.

We have multiple methods to convert the directed and weighted bipartite network into a projection of authors. We could simply remove the files and connect each author, that worked on the same file, however, the end result would be an unweighted graph. This would falsely show, that all collaborations are weighed equally, which is clearly not the case, as multiple continuous

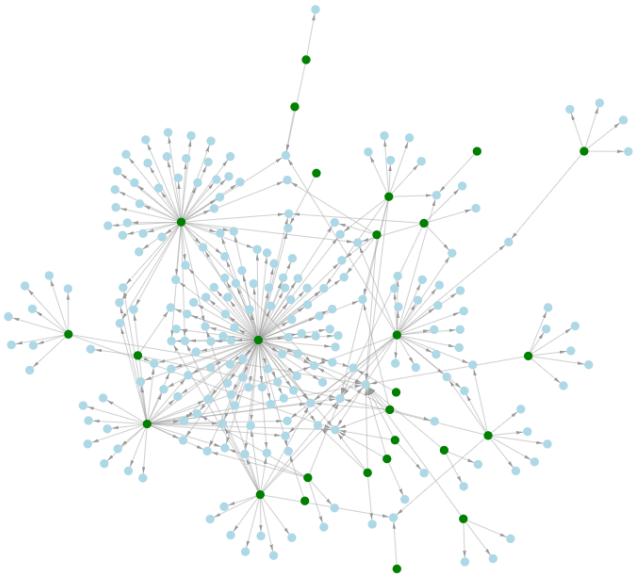

Figure 3: A bipartite network of authors (green) and edited files (light blue) in the `pandas` project within the timeframe 31/01/2021 and 15/02/2021

edits on the same file from both parties should represent a stronger collaborative connection. Therefore, firstly we implement the Weighted One-Mode Projection (WOMP) method [20], The WOMP method converts the bipartite network $G(A, F, E)$, where E is the edge list containing tuples (a_i, f_i, w_{ij}), and $w_{ij} \in E$ is the weight between author $a_i \in A$ and file $f_i \in F$. With this notation, a weighted directed edge can be calculated for any $a_a, a_b \in A$ as follows:

$$w_{ab}^{A \to A} = \sum_{j=1}^{m} \frac{w_{aj}}{W_a^F}$$

where W_a^F is the sum of all outgoing edge weights from author a to all files F denoted as $W_a^F = \sum_{i=1}^{n} w_{ai}$. This creates a bidirectional weighted collaboration network between authors a_1 and a_2, where the weight w_{12} represents the relative collaboration effort of a_1 towards a_2 compared to all the other developers a_1 has collaborated with. Conseqently, every edge is in the rage $[0, 1]$ in the resulting WOMP network.

A disadvantage of the WOMP method is, that the generated collaboration network is bidirectional, meaning if there was any common authored files between a_1 and a_2, then there will be both w_{12} and w_{21} connecting them.

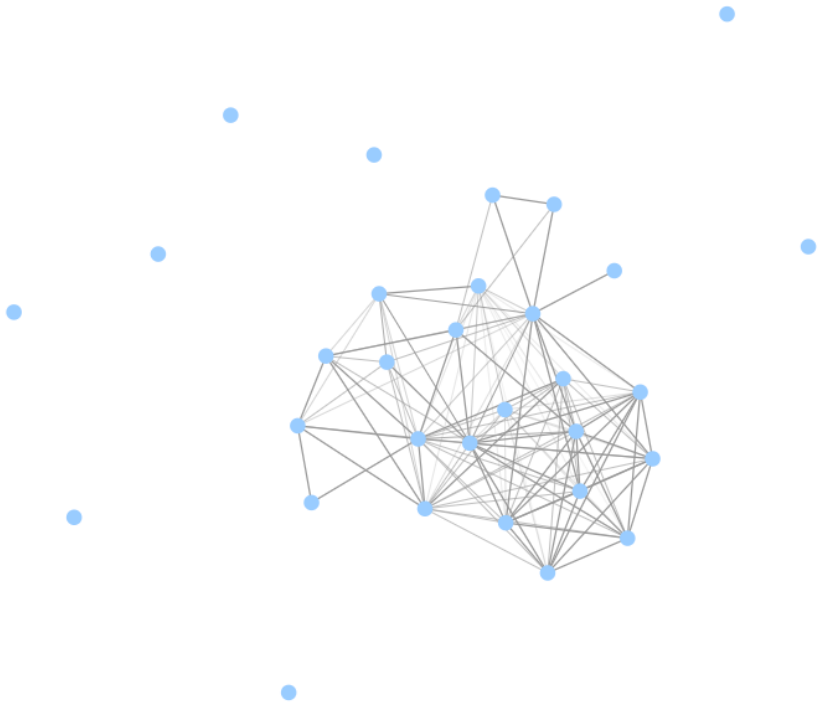

Figure 4: Weighted Jaccard similarity collaboration network of `pandas` generated from the bipartite network in Figure 3.

To simplify the network, we want to generate an authors network, where the edges are undirected. For this, we are using the weighted Jaccard method on the files-authors bipartite network:

$$w_{ab} = \frac{\sum_{f \in F} min(f_a, f_b)}{\sum_{f \in F} max(f_a, f_b)}$$

For each file f that a_1 and a_2 authors touch, we sum up the minimum and maximum weights the authors have towards each file, then we divide the sum of minimums with the sum of maximums. This results in the undirected author-author network with edge weights in range $[0, 1]$. By default, this method removes isolated contributors, who do not collaborate with eachother, but are actively editing the files. We add these nodes manually. Figure 4 shows the final author network.

4.5 Core and periphery, centralization

A critical part of the OSS software projects is the existence of core and periphery developers. It has been observed, that in each FLOSS project there are a small number of developers, who provide the vast majority of development effort into the project. It has been also established, that the members of core developers do not change substantially during the project's lifecycle. However, there was no effort on whether there is a change in the collaboration pattern, especially before, during or after a major lifecycle event. Therefore, we make efforts identifying the core developer network to observe these changes.

We use the degree centrality of each node (i.e. developer) to identify the core members. Degree centrality of a node is the fraction of all possible nodes it is connected to. We can calculate it by dividing the degree with $n-1$, where $n = |G|$ the number of nodes within the network. Since the core developers contribute the majority of commits and edits of the project, they are expected to be connected with more nodes. Joblin et. al. [11, 12] have also identified degree centrality as the best predictor of core developers. In cases, where binary classification of core or periphery is needed, we assign developers to the core network if their degree centrality score is in the top 20th percentile, otherwise they are considered as periphery. We also take note, that this method does not consider the weighted edges, only the number of edges (degree) a node has. Although this method could be refined to consider the node degree weighted with the edges, we argue that this could lead to invalidity. In case of two developers, who only contributed to one file, they will be represented with a strong connection and would receive a high weighted degree value, whereas a core contributor, who edits many files, can have many weak connections but these might not add up to one strong connection of the two isolated developers when weighted with the edge weights. It is clear, that a developer with many connections, regardless of the strength of the collaboration, should be considered core.

The degree *centrality* can be calculated for every node, however, through our analysis we would also like to measue a global *centralization* metric, which is applicable to the whole network. As suggested by Crowston and Howison [7], we calculate the degree *centralization* by summing the differences between the maximum and each node's degree *centrality*.

$$C_D(A) = \frac{\sum_{i=1}^{n}(C_d(a*) - C_d(a_i))}{H}$$

where $C_d(a)$ is the degree *centrality* of an author a, $a*$ is the author with

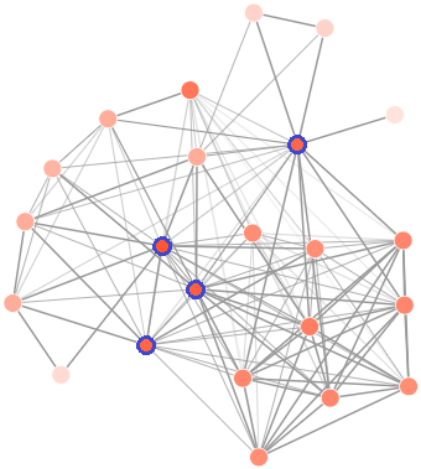

Figure 5: Degree centrality represented on the `pandas` collaboration network. Darker colors represent a higher degree centrality value. The four highlighted nodes are in the highest 20th percentile of degree centrality, classifying as members of the core developers.

the highest degree *centrality* value, and n is the number of authors in the collaboration network A. Since the *centrality* values are already in the range $[0, 1]$, the

4.5.1 Centralization

"Specifically, the network centralization is calculated as the sum of the differences between the maximum and each individual's centrality score, normalized to range from 0 to 1 by dividing by the theoretical maximum centralization. [7]"
[13]

4.6 Network statistics

4.6.1 Hierarchy

5 Collaboration pattern analysis

5.1 Observed projects and events

5.2 SNA metrics analysis

5.3 Results

6 Quantitative analysis of projects during crunch time

6.1 Collaboration network changes

6.2 Prediction of outcome based on collaboration changes

7 Discussion and results

8 Conclusion and future work

References

[1] Shaosong Ou Alexander Hars. Working for Free? Motivations for Participating in Open-Source Projects. *International Journal of Electronic Commerce*, 6(3):25–39, April 2002.

[2] Mohammed Aljemabi and Zhongjie Wang. Empirical Study on the Evolution of Developer Social Networks. *IEEE Access*, PP:1–1, September 2018.

[3] M. Antwerp. Evolution of open source software networks. pages 25–39, January 2010.

[4] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. A Novel Approach for Estimating Truck Factors. *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10, May 2016.

[5] 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*, SIGSOFT '08/FSE-16, pages 24–35, New York, NY, USA, November 2008. Association for Computing Machinery.

[6] Kevin Crowston and James Howison. The social structure of free and open source software development. https://firstmonday.org/ojs/index.php/fm/article/download/1478/1393?inline=1, February 2005.

[7] Kevin Crowston and James Howison. Hierarchy and centralization in free and open source software team communications. *Knowledge, Technology & Policy*, 18(4):65–85, December 2006.

[8] Ikram El Asri, Noureddine Kerzazi, Lamia Benhiba, and Mohammed Janati. From Periphery to Core: A Temporal Analysis of GitHub Contributors' Collaboration Network. In Luis M. Camarinha-Matos, Hamideh Afsarmanesh, and Rosanna Fornasiero, editors, *Collaboration in a Data-Rich World*, IFIP Advances in Information and Communication Technology, pages 217–229, Cham, 2017. Springer International Publishing.

[9] Christoph Gote, Ingo Scholtes, and Frank Schweitzer. Analysing Time-Stamped Co-Editing Networks in Software Development Teams using git2net. *arXiv:1911.09484 [physics]*, November 2019.

[10] Christoph Gote and Christian Zingg. Gambit – An Open Source Name Disambiguation Tool for Version Control Systems. *arXiv:2103.05666 [physics]*, March 2021.

[11] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics. *arXiv:1604.00830 [cs]*, April 2016.

[12] Mitchell Joblin, Sven Apel, and Wolfgang Mauerer. Evolutionary trends of developer coordination: A network approach. *Empirical Software Engineering*, 22(4):2050–2094, August 2017.

[13] Michael Klug and James P. Bagrow. Understanding the group dynamics and success of teams. *Royal Society Open Science*, 3(4):160007.

[14] Yu-Ru Lin, Hari Sundaram, Yun Chi, Junichi Tatemura, and Belle Tseng. Blog Community Discovery and Evolution Based on Mutual

Awareness Expansion. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 48–56, November 2007.

[15] Juan Martinez-Romo, Gregorio Robles, Jesus M. Gonzalez-Barahona, and Miguel Ortuño-Perez. Using Social Network Analysis Techniques to Study Collaboration between a FLOSS Community and a Company. In Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, and Giancarlo Succi, editors, *Open Source Development, Communities and Quality*, volume 275, pages 171–186. Springer US, Boston, MA, 2008.

[16] M. R. Martínez-Torres. A genetic search of patterns of behaviour in OSS communities. *Expert Systems with Applications*, 39(18):13182–13192, December 2012.

[17] Kelvin McClean, Des Greer, and Anna Jurek-Loughrey. Social network analysis of open source software: A review and categorisation. *Information and Software Technology*, 130:106442, February 2021.

[18] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, July 2002.

[19] PwC. Leading benefits of open-source software among enterprises worldwide as of 2016. *Statista*, 2016.

[20] Rotem Stram, Pascal Reuss, and Klaus-Dieter Althoff. Weighted One Mode Projection of a Bipartite Graph as a Local Similarity Measure. pages 375–389, June 2017.

[21] Ashish Sureka, Atul Goyal, and Ayushi Rastogi. Using social network analysis for mining collaboration data in a defect tracking system for risk and vulnerability analysis. In *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, pages 195–204, New York, NY, USA, February 2011. Association for Computing Machinery.

[22] Xuan Yang, Daning Hu, and Davison M. Robert. How Microblogging Networks Affect Project Success of Open Source Software Development. In *2013 46th Hawaii International Conference on System Sciences*, pages 3178–3186, January 2013.

[23] Yunwen Ye and K. Kishida. Toward an understanding of the motivation of open source software developers. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 419–429, May 2003.