Who Can Help to Review This Piece of Code?

Noureddine Kerzazi, Ikram El Asri

Who Can Help to Review this Piece of Code?

Noureddine Kerzazi, Ikram El Asri

National Higher School for Computer Science
and System Analysis (ENSIAS)
in Rabat, Morocco
[n.kerzazi, ikram.asri]@um5s.net.ma

Abstract. Successful software projects require collaboration between team members. Efficient collaboration relies on both technical and social linkages. In this paper, we investigate whether a socio-technical analysis can support software contributors in identifying experts helping to review their source code. We mined the histories of five open source projects (OSS) from GitHub and examined both technical and socio-technical interactions based on Social Network Analysis (SNA). Mapping communication network to files co-edition network shows the existence of collaboration patterns between core teams and peripherals in the studied OSS projects. Our main contribution is the construction and mapping of three sources of social networks, in which contributors interact by co-editing, commenting or reviewing. We were able to identify behavioral patterns between core teams and peripherals related to the activity of code review. Our findings have implications on improving collaboration between contributors within virtual OSS communities witch drive teams' performance and software products quality.

Keywords: Collaboration, Coordination, Socio-Technical Relationships, Distributed development, Modern Code Review.

1. Introduction

Software development is performed by development teams collaborating and interacting together on technical artifacts [1]. Lack of coordination, team cohesion, awareness, or misguided effort can all result in software failures or team performance decrease [2, 3]. To avoid collaboration breakdowns occurring at technical level, team members should communicate on dependencies between technical artifacts (e.g., source code files). Finding the right person to ask for help is a difficult task within a large organization [4]. Furthermore, prior studies showed that selecting unsuitable reviewers may delay the review process by an average of 12 days [5, 6].

Interactions between team members, through different channels, constitute the backbone of socio-technical[1] perspective which has gain increased attention over the past decade [7]. Social Network Analysis (SNA) has been used to capture and understand

[1] "Socio-technical" term refers to the interactions between individuals involved in a software project. It occurs because of collaboration needs due to technical dependencies.

such information about relations among people [8] with the aim to enhance team performance, team members congruence, and software product quality.

Most software development projects, including open source software (OSS), have adopted the practice of code review as a collaborative mechanism aiming at inspecting new code changes made by contributors to evaluate their quality and identify potential defects [6]. For instance, experience indicates that proactive software defect issues are best found by experts when reviewing the code [9, 10]. However, co-workers need to identify right persons to review their code, especially in virtual environments such as for open source projects. If we could support developers identifying the right person likely to perform good review, we can enhance the quality of the code [10] and improve the signal to noise ratio of comments on commits, witch decrease the team's performance. One way of localizing reputed domain expert, to ask for reviewing a piece of code, is to build contributors socio-technical networks and analyze them. Historically, SNA has been known to be effective in many areas [4].

In this paper, we examine the socio-technical interactions of five OSS using version control system data history. We construct contributors' networks based upon which files are commonly modified by contributors, commented, and previously reviewed by contributors. Using network analysis, we can uncover details of required collaboration/coordination and recommend a list of experts for assigning suitable reviewers to a given software change. The contribution of this paper is threefold: *(1)* the empirical evidence that we need to improve the delay of collaboration through code review within OSS communities; *(2)* a new approach SNA-base for identifying suitable list of persons, for source code review assignment, based on SNA metrics; and *(3)* a large-scale experiment on five OSS projects, namely: AngularJs, Docker, Ruby, Symphony, and JQuery, involving more than 1000 (JQuery 250) of virtual contributors. The main contributions of this work are:

- We empirically explore the source code review activity in five OSS aiming at uncovering unseen social interactions within a core-periphery structure;
- An approach to overlap different networks (files co-edition, reviews, commenters) aiming to localize contributors in one network on an another one;
- We demonstrated how contributors with a high centrality degree have more likely a shorter review interval than peripheral contributors.

The rest of the paper is organized as follows: Section 2 outlines our motivation and pinpoints our research questions. Section 3 surveys related work. Section 4 presents the methodology including projects description and data collection process. Section 5 provides our results. Section 6 discusses the threats of validity. Section 7 concludes and outlines future work.

2. Background & Motivation

2.1. Social Network Analysis

Social Network Analysis is a research domain studying the relationships between individuals [8]. These networks are typically modeled as graphs in which nodes represent individuals and edges represent relationships between these individuals. For instance, if two contributors have modified the same file, there is a co-edition relationship that can be represented as an edge. Similarly, if two developers comment on technical artifacts then we can represent this interaction (i.e., the list of comments) as an edge between those two nodes. These social interactions have an impact on group dynamics affecting collaboration and task efficiency performance. Therefore, previous research has shown that SNA techniques might help to understand different types of developer interactions in OSS projects. Bird et al.[1] mapped code commit interactions (co-edition files) to interactions in the mailing-list.

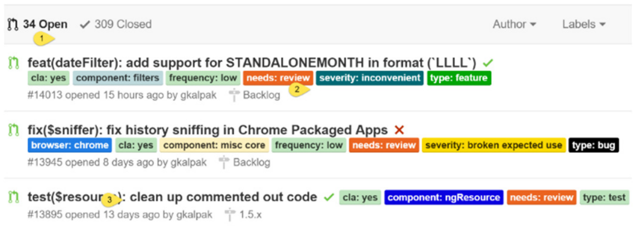
A wide range of SNA metrics can be applied for performing social analysis. Each metric looks at either specific properties of the networks or properties of the nodes. For our work we focused on centrality indices as detailed in Table1.

Table 1. Social Network Centrality Metrics

Metric	Definition	Meaning
Degree Centrality	Number of edges incident upon a node. The degree centrality of a vertex v, for a given graph G is: $C_D(\vartheta) = \deg(v)$	Indication of how many other contributors each contributor interacts with. Core developers have a higher degree centrality because they interact with a large number of other contributors.
Betweenness Centrality	The number of shortest paths traversing a given node.	Measure of the importance of a node within a graph. Contributors with a high betweenness centrality are broker nodes for communication with other contributors. Betweenness is useful as an index of the potential of a node for control of communication.
Closeness Centrality	Distance of an actor to all other nodes in the network.	Gives an indication of how quickly a developer can reach the entire community. Used to assess the speed of information spread from a given node to other reachable nodes in the network.
Eccentricity	Maximum distance from a node to all other nodes in the network	A high eccentricity means that the most distant vertex in the network is a long way away, and a low eccentricity means that the most distant vertex is actually quite close. A central developer should have lower eccentricity as they will have direct communication ties with many other developers.

2.2. Motivation

Software development is performed by teams collaborating and interacting together on technical artifacts. Code review has been widely used in both companies and open source software projects [11] mainly to ensure the inclusion of only high-quality code into the project codebase. Typically, a developer submits a code change to a code review system and recommends a set of developers to review his changes. Then, the reviewers would provide comments and suggest improvements to the changes. Next, the developer improves the change according to the comments. Once those improvements have been applied and approved by reviewers they can be integrated to the mainstream branch of the version control. Figure 1 shows a concrete example of a list of pending changes waiting for the review process.



34 Open ✓ 309 Closed Author ▾ Labels ▾

1

feat(dateFilter): add support for STANDALONEMONTH in format (`LLLL`) ✓
cla: yes | component: filters | frequency: low | needs: review | severity: inconvenient | type: feature
#14013 opened 15 hours ago by gkalpak Backlog

2

fix($sniffer): fix history sniffing in Chrome Packaged Apps ✗
browser: chrome | cla: yes | component: misc core | frequency: low | needs: review | severity: broken expected use | type: bug
#13945 opened 8 days ago by gkalpak Backlog

test($resour 3): clean up commented out code ✓ cla: yes | component: ngResource | needs: review | type: test
#13895 opened 13 days ago by gkalpak 1.5.x

Figure 1. A real example of requested needs for reviews (AngularJS project).

Preliminary analysis of data related to code review revealed a median delay ranging from 1 to 20 days. For instance, we found that for AngularJS project the review process takes, on average, 14 days (± std. dev 39.2) with a mean of 5 comments by review as illustrated in Figure 2. Finding appropriate code-reviewers in OSS projects can be a tedious and time consuming task.

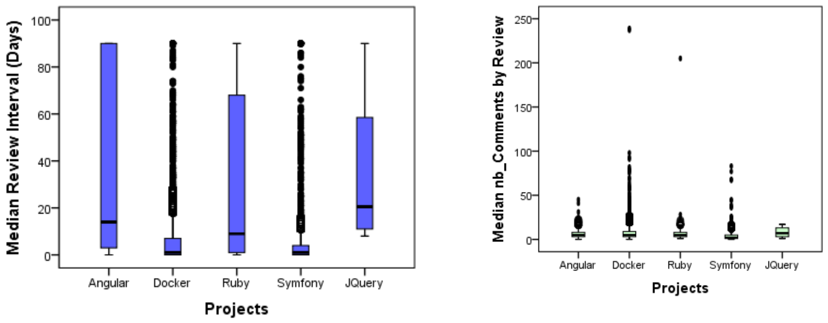


Figure 2. Review Intervals (left); nb_Comments by Review (right)

To speed up the code review process, contributors need to be supported with an automated approach seeking to find experts in the related source code. Thus, our exploration of code review using Social Network Analysis leads to the following research questions:

- ***RQ1.*** How does socio-technical interaction vary for various open source communities?
- ***RQ2.*** Does the socio-technical analysis make collaboration an actionable concept?
- ***RQ3.*** Contributors with a high centrality degree have shorter review interval than peripheral contributors?

3. Related Work

We address the related research areas with respect to socio-technical networks, source code review practice, and expert recommendation.

Socio-Technical Networks. It has been reported that higher socio-technical congruence usually correlates with higher developer productivity [2] and reduces integration failures [3]. Both researchers and OSS projects leads could use STC to diagnose project members' collaboration and improve team coordination. Prior work has shown that social networking contains plenty of information that can be leveraged for other purposes [8]. For instance, Bird et al. [1] studied code commit interactions and showed that developers who work on the same file are more likely to interact in the mailing-list. An SNA techniques can help an actionable graph of previous commit interactions providing new kind of metrics and patterns for a computational analysis.

Code Reviews Practice. Rigby and Storey [10] examined manually hundreds of reviews across five high profile OSS projects aiming to investigate the mechanisms and behaviors that developers use to find code changes they are competent to review. They found that the Apache project adopted a broadcast-based style of code review, meaning increasing the awareness of new changes, but annoying the community with a high amount of irrelevant notification. Prior work has analyzed the modern code review process used by the open source community. Baysal et al. [6] studied the factors that influence the outcome of the review process and found that review is positivity influenced by non-technical factors such as organization. Furthermore, a recent qualitative study at Microsoft [11] confirmed that identification of defects is not the only motivation for code review, but sharing knowledge among team members is also considered as an important motivation of modern code review.

Expert Recommendation. Xia et al. [12] proposed an approach, based on history of bug report and developer information, to recommend a developer for a new bug fix. Surian et al. [13] studied Sourgeforge.Net attempting a cross-projects developer recommendation using developers' collaboration network. Authors try to recommend a list of top developers that are most compatible for a specific task based on their past projects they have worked on before. In the area of Question & Answer community (i.e., StackOverflow) using topic modeling and collaborative voting scores, Tian et al. [14] introduced an expert recommendation system in order to identify user expertise relevant to the topics of a given question.

4. Research Methodology

Our goal in this paper is to discover how SNA metrics can help to understand the source code review practice within OSS projects. **Figure 3** illustrates our methodology regarding data extraction, processing and building different contributor's networks.

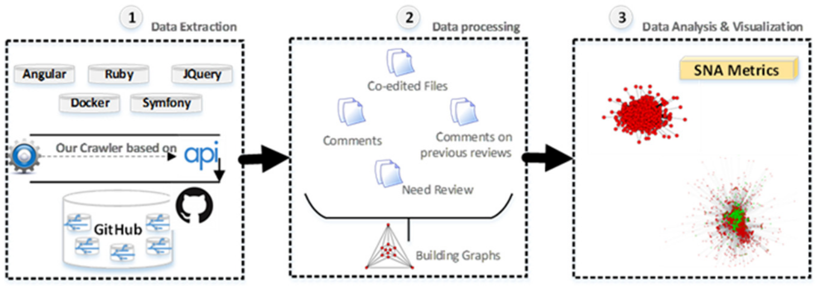


Figure 3. Overview of our data extraction and processing approach.

Phase1- **Data Extraction**. We focused our study on five large and rapidly-evolving open source systems. height stared projects from GitHub. Our choice of projects was based on the following criteria: *(i)* project should be among the 100 most stared projects; *(ii)* should be still under active development; and *(iii)* involving at least 250 contributors. Table 2 summarizes the characteristics of our selected projects including the programming language, the total number of developers, number of releases; number of lines of code; number of requested reviews; and the total of commits.

Table 2. Overview of the Studied Systems

	Overview				Commits	
	Language	Contributors	Releases	LOF	Request Review	Total
Angular.Js	JavaScript	1403	161	369.574	349	7534
Docker	Go	1301	129	670.722	2399	22318
Rails	Ruby	3000	283	387.862	750	56562
Symfony	PHP	1271	156	653.109	1193	25007
JQuery	JavaScript	250	134	62.566	10	6050

Phase2- **Data Processing:** For each project, we use GitHub API to extract detailed information about commits activity along with opened need reviews history. For each commit we were interested by the list of changed files (file name, status, author, churn) and comments (commenter, body, created at, etc.). When a collaborator opens a need review, conversation around the request is lunched. Thereby we were interested by contributors requesting reviews, carrying out a review, or simply commenting on it. We used these data to build three different graphs.

Phase3- **Data Analysis and Visualization**: For each project we constructed three networks: The first is the collaborators' network based on coedited files. We generated social network graphs as undirected, weighted graphs where nodes represent contributors and edge weights represent the quantity of interactions between those contributors. The second is reviews commenters based on discussion launched after a review request. For example, if contributors C_1 and C_2 commented together on 20 code review requests posted by any contributor then the weight of the edge between their nodes would be 20. In addition, we used mapping option in order to construct sub networks of contributors that requested, resolved and closed reviews requests.

5. Analysis and Results

RQ1. *How does socio-technical interaction vary for various open source communities?*

Motivation. We are interested in understanding whether or not the open source community has an efficient underpin collaboration mechanism. The exploration of the collaboration topologies between contributors in such a virtual environment will provide more insights on how the communication flows and what would be the main patterns for an effective collaboration related to source code reviewing.

Approach. For each project, we constructed three networks: *(1)* contributors' collaboration network based upon co-edition files where nodes are contributors and vertexes are undirected and weighted links between contributors having modified together same files; *(2)* Comment's network representing contributors interacting trough comments on technical artefacts; *(3)* Reviewer's network representing previous interactions on source code reviews.

Results. Figure 4 (first column) illustrates the structure and density of OSS contributors' network. A clear visual distinction appears through the five projects regarding the density of networks and concentration of core/peripheral developers. We were able to observe the hidden structure and organization of the communities in OSS development [15]. The Core developers are generally involved in an OSS project for a relatively long period, and make significant contributions to the development and evolution of the Open Source System. While Peripheral developers contribute occasionally to a given project.

Figure 4 also shows (two other columns) the mapping of other useful information on top of the contributors' network to provide respectively the network location of commenters on review and contributors performing reviews. We were able to discover that commenters (green spots) might be in the core as well as in peripheral areas. However, reviews are carried out by contributors with high degrees of centrality (bleu spots). We suspect that factors such as notoriety, awareness, and domain knowledge are of importance in this setting. A further topic analysis is required to get more insights regarding the density of core contributors.

RQ2. *Does the socio-technical analysis make collaboration an actionable concept?*

Motivation. Previous research examined the alignment between technical dependencies and social coordination within OSS projects [1]. Based on SNA, we try to confirm this correlation by examining connections between contributors in the context of work-related collaboration.

Approach. We used social network metrics to find patterns of collaboration needs. To do so, we manually analyzed interactions between collaborators through file co-edition, comments, and history of reviews. We then mapped our observations to our computed SNA metrics.

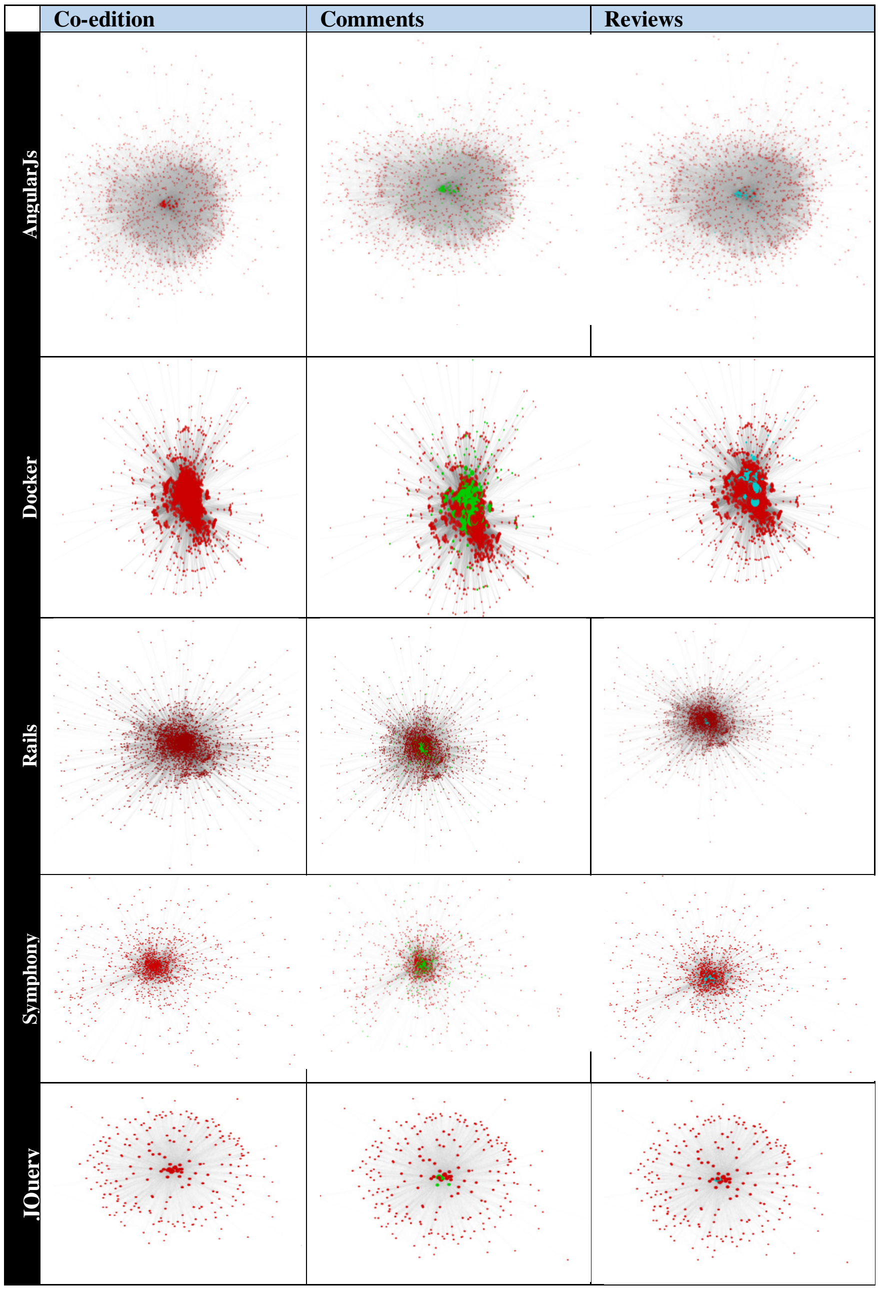


Figure 4. Structure and Density of OSS contributors' network

Table 3. SNA Metrics

	Clustering Coefficient	Network Centralization	Avg. # of neighbors	#Nodes	Network Density	Network Heterogeneity
Angular.Js	0.897	0.918	66.428	1393	0.048	1.674
Docker	0.874	0.794	81.385	1314	0.062	1.582
Ruby	0.855	0.876	207.242	3003	0.069	1.557
Symfony	0.866	0.952	63.425	1270	0.050	1.804
JQuery	0.834	0.725	61.197	244	0.252	0.837

Results. The core members across all projects play vital role of submitting reviews. For instance, we found that for AngularJs project, over 366 review requests (93%) of closed reviews were resolved by core developers.

We found that not all reviews are assigned, furthermore, assigned ones are most often closed within lower average days than others.

RQ3. *Contributors with a high centrality degree have shorter review interval than peripheral contributors?*

Motivation. Previous research [16] has shown that there are experts reviewing technical contributions involved in most OSS projects. However, this past research does not explain where are those expert located in the contributors' network and what is the mechanism for assigning reviews to a list of predefined experts categorized by domain knowledge. We aim to assess the influence of contributors' position within the network on how long it takes for a code change patch to be reviewed.

Approach. A recent study showed that core developers play a major role in the evolution of OSS projects [15]. We further extend these results with social network visualization on review and collaborators' networks with respect to the lead time for the review process.

Results.
Core contributors are source code review brokers. Based on SNA metrics and visualization, we were able to identify central core contributors (see density core in figure 4 – co-edition). We segregate contributors according to the degree of centrality they have and then we mapped these data to our network of reviews. Our analysis shows that we can go further with SNA metrics to study the position of contributors in the network and their review activities. Figure 5 shows a comparative of the amount of reviews, comments, and time interval regarding the position of the contributors within the network (centrality degree) as well as the distribution of degree for contributors. Because of the space constraint, we show only the comparison between AngularJs and Docker.

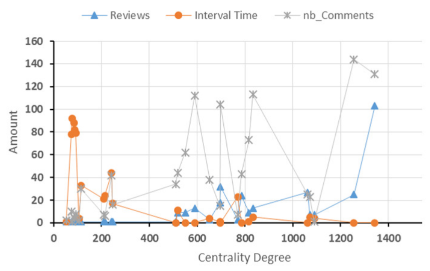


Fig.5.a1 **Angular** Review Metrics

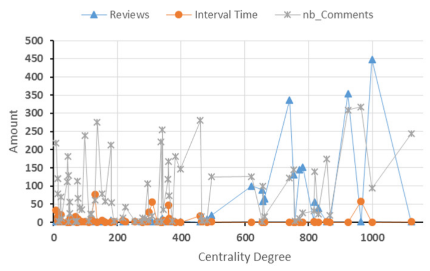


Fig.5.a2. **Docker** Review Metrics

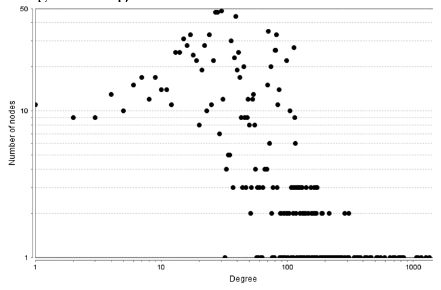


Fig.5.b1 **Angular** Degree Metric

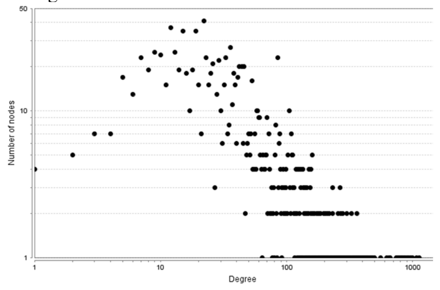


Fig.5.b2 **Docker** Degree Metric

Figure 5. Comparing Networks Metrics of Angular and Docker Projects

For AngularJs, reviews are requested by core team and carried out also by the core team. Surprisingly, we observed that for changes coming from peripherals the comments on reviews flows from the core to the periphery and went back to the core for final acceptance. However, for code changes coming from the core team we observed a less amount of comments and less discussions. Consequently, **more a contributor is central in the network of contributors likely he has a shortest time for code revision and acceptance**.

In addition, Figure 5b1 and 2 illustrates the distribution of centrality degree within AngularJS and Docker.

6. Threats to Validity

Construct validity. First, we assume that all communications occur with comments either on commits or on code reviews. Developers on Github might use an external social media or mailing list to communicate. We are actively expanding the scope of our research to interview Github contributors, via a survey, in order to get more insights on factors that might improve collaboration on code review. Second, there may be number of unmeasured confounding factors that could influence a review outcome such as the projects architecture, the code complexity, and the availability of contributors. There is no evidence that our partitioning approach or our measurements would be systematically biased based on any of these confounding factors. Furthermore, the fact that the results were similar across all eight projects provides additional confidence that any confounding factors did not significantly impact the study results.

Internal validity. We are aware that we might miss transitive dependencies between technical elements. For instance, changing the framework on which depends many files. Moreover, software development is dynamic, and as contributions are made over time, the nature of the socio-interactions changes. We mitigated this threat by studying multiple open source projects, using different languages, within the GitHub community.

External validity. We focus our evaluation on five open source systems, which threatens the generalizability of our case study results, as with all empirical studies. However, we studied a variety of systems from different domains, various languages, and sizes to alleviate potential bias. Nonetheless, additional replication studies are needed.

7. Conclusions

In this paper, we used a Social Network Analysis approach aiming at studying the collaboration through code review practice within five OSS projects. We extracted and analyzed historical data of five large and long-lived OSS projects. We described collaboration between OSS contributors based on socio-technical analysis regarding files co-edition, comments, and reviews. The first key contribution of this study relates to our ability to visualize how contributors from the core team development interact with peripheral ones. Another key contribution of this study is our socio-technical analysis to make collaboration an actionable concept. Using a longitudinal data set which connects the same contributors over time in different interactions, we have been able to examine the backbone of code review practice within virtual communities, which we think is crucial for software products quality. Finally, we demonstrated how contributors with a high centrality degree have more likely a shorter review interval than peripheral contributors. Our study has important practical implications. As understanding the nature and types of collaborations in a virtual environment which provided some insights on the main patterns for collaboration related to source code reviewing. Our results suggest that core contributors play vital roles in carrying out review requests.

As part of our future work we want to inspect the temporal dynamics by which an OSS community structure grows and fades over time. For example, we could study temporal dynamics to determine patterns of how newcomers to an OSS project progress from the periphery to the core teams, how they gain their notoriety and how they lead and impact the trajectory of OSS projects.

References

[1] C. Bird,D. Pattison,R. D'Souza*, et al.*, Latent Social Structure in Open Source Projects. in *Proc. of the 16th Int'l Symp. on Foundations of Soft. Eng. (FSE' 08)*, (Atlanta, Georgia), 24-35, 2008.

[2] M. Cataldo and J.D. Herbsleb Coordination Breakdowns and Their Impact on Development Productivity and Software Failures. *Transactions on Softw. Eng.*, *39* (3): 343-360, 2013.

[3] I. Kwan,A. Schroter and D. Damian Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project. *Transactions on Softw. Eng.*, *37* (3): 307-324, 2011.

[4] S. Yarosh,T. Matthews,M. Zhou*, et al.* I need someone to help!: a taxonomy of helper-finding activities in the enterprise *Proc. of the 27th Int'l Conf. on Computer Supported Cooperative Work (CSCW)*, Texas, 1375-1386, 2013.

[5] P. Thongtanunam,C. Tantithamthavorn,R.G. Kula*, et al.*, Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review. in *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, 141-150, 2015.

[6] O. Baysal,O. Kononenko,R. Holmes*, et al.*, The Influence of Non-Technical Factors on Code Review. in *In Proc. of the 20th Working Conference on Reverse Engineering*, (Koblenz, Germany), 122-131, 2013.

[7] A. Begel,R. DeLine and T. Zimmermann. Social media for software engineering *FSE/SDP workshop on Future of software engineering research*, Santa Fe, New Mexico, USA, 33-38, 2010.

[8] C. Kadushin *Understanding Social Networks: Theories, Concepts, and Findings*. Oxford University Press, 2011.

[9] A. Meneely and L. Williams, Socio-technical developer networks: should we trust our measurements? in *Proc. of the 33rd Int'l Conf. on Software Engineering (ICSE '11)*, (Waikiki, Honolulu, USA), 281-290, 2011.

[10] P.C. Rigby and M.-A. Storey, Understanding broadcast based peer review on open source software projects. in *Proc. of the 33rd Int'l Conf. on Software Engineering (ICSE '11)*, (Waikiki, Honolulu, USA), 541-550, 2011.

[11] A. Bacchelli and C. Bird, Expectations, outcomes, and challenges of modern code review. in *Proc. of the 35th Int'l Conf. on Software Engineering (ICSE '13)*, (San Francisco, CA, USA), 712-721, 2013.

[12] X. Xia,D. Lo,X. Wang*, et al.*, Accurate developer recommendation for bug resolution. in *2013 20th Working Conference on Reverse Engineering (WCRE)*, 72-81, 2013.

[13] D. Surian,N. Liu,D. Lo*, et al.*, Recommending People in Developers' Collaboration Network. in *2011 18th Working Conference on Reverse Engineering*, 379-388, 2011.

[14] Y. Tian,P.S. Kochhar,E.-P. Lim*, et al.* Predicting Best Answerers for New Questions: An Approach Leveraging Topic Modeling and Collaborative Voting. in *Proc. of the Int'l Workshops, QMC and Histoinformatics*, Springer, 2014, 55-68.

[15] J. Geldenhuys, Finding the Core Developers. in *36th Euromicro Conf. on Soft. Eng. and Advanced Applications*, (Lille, France), 447-450, 2010.

[16] J. Asundi and R. Jayant, Patch Review Processes in Open Source Software Development Communities: A Comparative Case Study. in *the 40th Annual Hawaii International Conference on System Sciences*, 166-166, 2007.