Knowledge Flows Within Open Source Software Projects: A Social Network Perspective

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. Developing software is knowledge-intensive activity, requiring extensive technical knowledge and awareness. The abstract part of development is the social interactions that drive knowledge flows between contributors, especially for Open Source Software (OSS). This study investigated knowledge sharing and propagation from social perspective using social network analysis (SNA). We mined and analyzed the issue and review histories of three OSS from GitHub. Particular attention has been paid to the socio-interactions through comments from contributors on reviews. We aim at explaining the propagation and density of knowledge flows within contributor networks. The results show that review requests flow from the core contributors toward peripheral contributors and comments on reviews are in a continuous loop from the core teams to the peripherals and back; and the core contributors leverage on their awareness and technical knowledge to increase their notoriety by playing the role of communication brokers supported by comments on work items.
Keywords: Knowledge flows, Expertise, SNA, Open Source.

1 Introduction

Open source communities can be perceived as knowledge-sharing ecosystems in which contributors learn from the community and from each other [1]. They share both domain and technical knowledge through contributions to the source code repositories or by reviewing source code from one another. Interactions between contributors, which can be materialized by looking to co-edited files [2], constitutes the backbone of socio-technical perspective which has gained increased attention over the past decade[3-5]. Social Network Analysis (SNA) has been used to capture and understand such information about relations among people [6] with the aim at enhancing team performance and software product quality.

Previous research has shown that there are expert reviewing technical contributions involved in most OSS projects [7]. However, this past research does not explain how developers identify experienced contributors to review their code and how awareness and knowledge are spread through the contributors' community. Many open source (OSS) projects adopt the practice of code reviews to increase the quality of their software products [8]. Collaboration on code review aims not only to improve the

quality of code changes made by contributors [9], but also for the purpose of knowledge transfer and awareness [10], [11]. If we could explain the propagation of knowledge flows within contributor networks throw code source reviews, we can enhance the quality of the code and improve the signal to noise ratio of comments on commits which decrease teams' performance. One way of locating reputed domain expert, to ask for reviewing a piece of code, is to build contributors networks and analyze it.

Historically, SNA has been known to be effective in many areas [12]. In this paper, we examine the socio-technical interactions for three OSS. Using histories of version control data, we constructed contributors' networks based upon which files are commonly modified by contributors. Using network analysis, we can uncover details of knowledge sharing and the circulation of knowledge flows between the core and peripheral contributors. Our research questions can be summarized as follows:

RQ1. Is there a Relationship between Contributors' Network and Knowledge Sharing?

RQ2. Does the network position of contributors affect the review process and the number of comments on GitHub projects?

RQ3. Does the Socio-Technical analysis make knowledge transfer an actionable concept?

RQ4. What Kind of Knowledge is transferred?

The main contributions of the paper are as follows:

- A thorough Social Network Analysis of three OSS projects that provides insights into socio-interaction of contributors and their knowledge sharing;
- A view of knowledge circulation through code review practice along with the kind of knowledge that is transferred;
- An exploration of how SNA metrics can inform to answer whether or to what extent an open source community has a good underpin knowledge and awareness sharing mechanisms;
- An understanding of whom are requesting code review; whom are commenting on reviews; and whom are performing reviews according to their network position and degree.

Paper organization. The remainder of the paper is organized as follows. Section 2 presents related work and background. Section 3 introduces our SNA-based method for identifying knowledge flows. Section 4 describes the selected projects from GitHub and data collection process. Section 5 provides our study results. Section 6 discusses our finding and points out practical implications. Section 7 discloses the threats of validity. Section 8 concludes and outlines future work.

2 Related Work and Background

Considering people at the heart of OSS projects, SNA in software development teams shows that social networking contains tremendous information that can be leveraged for purposes such as: defects prediction [3, 13], teams' organization and

coordination [14], team productivity [15], and tools or techniques for the purpose of studying developer communities [1, 4]. We first summarize related work according to these three different perspectives. Then we introduce previous work on knowledge sharing and propagation. Finally, we present what we know about SNA measures.

Defects Prediction- Rigby and Storey [16] examined manually hundreds of code reviews across five high-profile OSS projects aiming to investigate the mechanisms and behaviours 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. Baysal et al. [9] studied the factors that influence the outcome of the review process and found that review positivity (i.e., the proportion of accepted patches) can be influenced by non-technical factors such as organization.

Furthermore, a recent qualitative study at Microsoft [10] showed that identification of defects is not the only motivation for code review, but **sharing knowledge among team members** is also considered as a very important motivation of modern code review. This related work indicates that our findings are not specific to the open source community but can be applied within commercial organizations.

Organization and Coordination- Recently, there has been considerable interests and work on improving the coordination between software team's members [17]. Knowledge dependencies drive the need to coordinate software process activities. Saying that an SNA approach can support identification of coordination needs by identifying previous collaboration and communications. Social Network metrics arise as a response to those questions such as who should do what, when it is required.

Productivity- It has been reported that higher socio-technical congruence usually correlates with higher developer productivity [15] and reduces integration failures [17]. Both researchers and OSS projects leads could use STC to diagnose project members' collaboration and improve team coordination [18].

Social Knowledge Sharing- Prior work has shown that social networking contains plenty of information that can be leveraged for other purposes[14]. For instance, the socio-cultural learning theories state that people learn from each other through observation, interaction and communication [19]. Seeing learning through its social aspect emphasizes the fact that OSS projects are increasingly growing. Contributors are part of a community of practice, organization, and belong to a group of people where there is competence knowledge already established. Source code review practice and comments are seen as ideal vehicles for leveraging tacit knowledge and learning.

3 SNA-Based Knowledge Flows

According to SNA [20], a Network consists of a set of nodes and a set of edges. Thus, we represent contributors as **nodes** as shown in figure 1. **Connections**, between those

nodes, are weighted and represented based on the number of files the pair has collaborated on.

When two contributors are directly connected by an edge they are *adjacent*. The number of adjacent connections for a given contributor is called the *Degree* of that contributor. As illustrated in figure 1, C_3 has a degree of 3 and C_4 has a degree of 1.

Geodesic path refers to the shortest social distance between two contributors represented such as adjacent and unique connections. While networks' **diameter** refers to the longest path between two contributors.

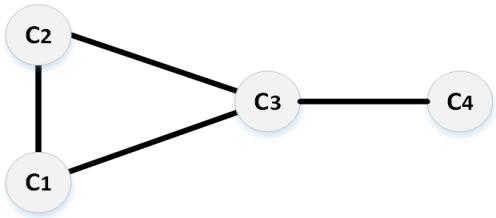

Fig. 1. An example of Contributors' Network with four nodes

3.1 Contributor Network Metrics

Connectivity metric measuring direct connections between nodes. SNA has come up with three distinct structural properties to measure the centrality of a given node.
Centrality metrics measure how closely contributors are indirectly connected to each other in the network. SNA measures the centrality based on two metrics: *closeness* and *betweenness*.

Closeness refers to the average distance from a node to any other node in the network. For example, Closeness for $C_1 = (1+1+2)/3 = 4/3$ noticing that the shortest paths from C_1 to (C_2 and C_3) are each 1 and the shortest path from C_1 to C_4 is 2. For instance, figure 1 shows that C_3 has the maximum possible degree (3) meaning that it is central in this network. While C_1 and C_2 have a degree equal to 2; and C_4 has a degree of 1 meaning that this contributor is peripheral in this network.

Betweenness is another centrality metric calculated for a given node as the number of shortest paths that include this node divided by the total number of shortest paths in the network. In the example of figure 1, we have a total of 6 shortest paths. Saying that the betweenness of C_1 and C_2 is 3/6. while the betweenness of C_3 is 5/6.

4 DataSets

We focus our study on three large and rapidly evolving open-source systems which are highly 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 1

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 shows the characteristics of each network.

Table 1. Overview of the studied systems.

	Overview				Commits	
	Language	Contributors	Releases	LOC	Request Review	Total
Angular.Js	JavaScript	1403	161	369.574	349	7534
Docker	Go	1314	129	670.722	2399	22318
JQuery	JavaScript	250	134	62.566	10	6050

For each project we queried the GitHub API with the query https://api.github.com/repos/<owner>/<repo>/commits?page=<n>, where <owner> is a GitHub user account and <repo> is the name of the repository. Hence, we extracted the commits data for each project including details such as the programming language used, the time period covered, the number of commits and developers, information about releases as well as the number of edited files.

Once the commits and edited files were linked, we were interested by all requests of reviews for each project. Since our study is focused on knowledge sharing between contributors, we also extracted all comments on each code review. Our query retrieves open and closed issues (i.e., state=all), along with labels tagged as 'need review'. Figure 2 summary interval times needed to close code review requests.

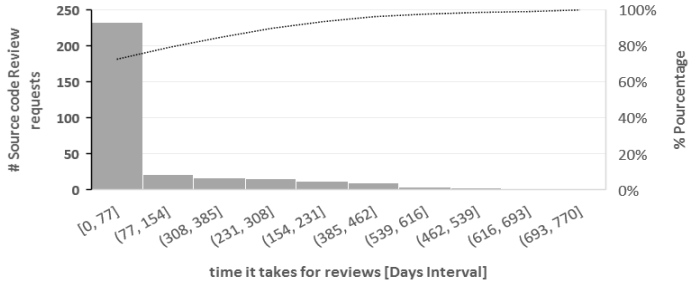

Fig. 2. Time required to close a request for code reviews.

Table 2. Metric of the Networks

	Clustering Coefficient	Network Centralization	Avg. # of neighbours	#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
JQuery	0.834	0.725	61.179	244	0.252	0.837

5 Study Results

RQ1. *Is there a Relationship between Contributors' Networks and knowledge sharing?*

We were interested to find the position of contributors within the contributors' network, that are asking for code reviews, commenting on code reviews, and carrying out reviewing activity. Figure 3 shows a comparative between Angular and Docker projects. We represent in red colour contributors' network on top of which we map sub-networks. For instance, Fig.3a1 illustrates the social network of *Angular* (read colour) and a sub network of contributors that requested code review (blue colour). One can observe the differences between the two projects in terms of density and position of requesters of code reviews.

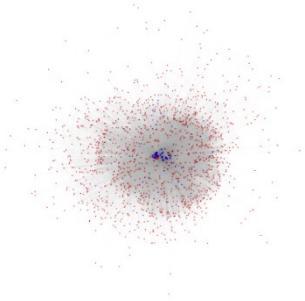

Fig.3.a1 Angular project Contributors Network which requested a code review

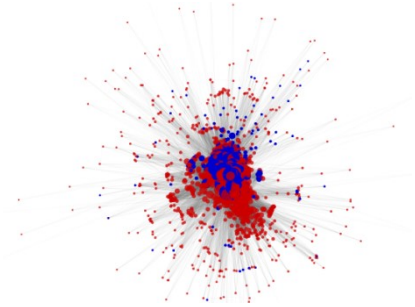

Fig.3.a2 Docker project Contributors Network which requested a code review

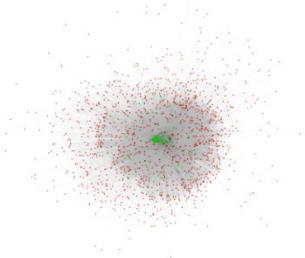

Fig.3.b1 Angular Network of commenters on code review

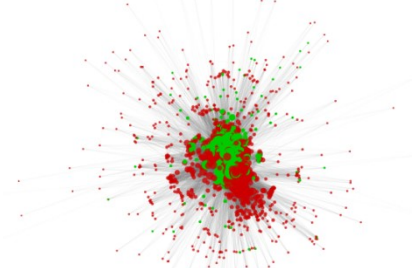

Fig.3.b2 Docker Network of Commenters on code review

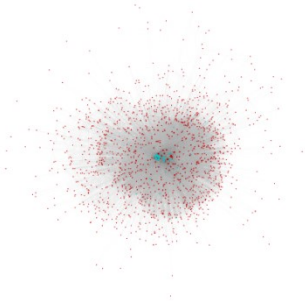

Fig.3.c1 Angular Network of Contributors that Resolve and close the code review

Fig.3.c2 Docker Network of Contributors that Resolve and close the code review

Fig. 3. Comparing Different Networks of Angular and Docker Projects

Fig 3.$b_{1\text{-}2}$ show the mapping of the contributors who have commented on code reviews (green). And finally, Fig.3c. compares the relative network position of contributors that carried out the code reviews.

We found that core contributors act such as knowledge brokers and boundary spanners across comments loops not only from the periphery to the core, but also from the core to the periphery. We pay a close attention to how core contributors (experts) influence communication patterns through comments on code reviews and issues in OSS projects as well as transferring and spreading knowledge to peripherals.

RQ2. *Does the network position of contributors affect the review process and the number of comments on GitHub projects?*

We found a strong correlation between the position of contributors in the Network (*Degree*) and the number of comments on code reviews. Figure 4 illustrates the trend of communication in the *Angular* project. One can also notice that more the Degree is highly likely the contributors are active in transferring their knowledge and awareness to other contributors. For instance, surprisingly in *Angular* project we have identified 16.7% of contributors such as core[1] developers that generate 81.6% of communication against 83.3% of peripheral developers generating only 18.4% of the comments flow.

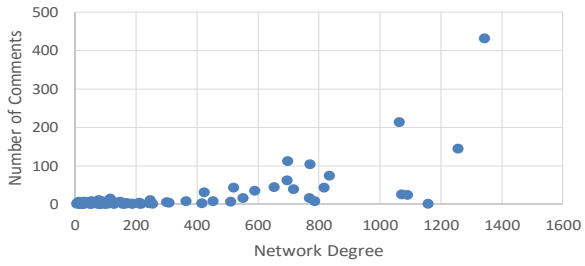

Fig. 4. Mapping the Contributors Centrality Degree Metric with the Number of Comments

RQ3. Does the socio-technical analysis make Knowledge Transfer an actionable concept?

Core contributors are communication brokers that have awareness and both technical and domain knowledge. SNA allows us to identify central core contributors. We segregate contributors according to the degree of centrality they have. Our analysis shows that we can go further with SNA metrics and patterns that can support studying

[1] We define a Degree threshold > 500 to filter on core developers which are marked as central in our SNA analysis.

knowledge flows in OSS communities similar to previous studies that attempt to predict software failures based on SNA metrics [13].

Figure 5 shows a comparative of the betweenness centrality metric as well as the distribution of degree for contributors.

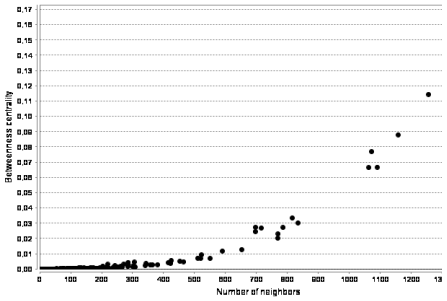

Fig. 5.a1 Angular Betweenness metric

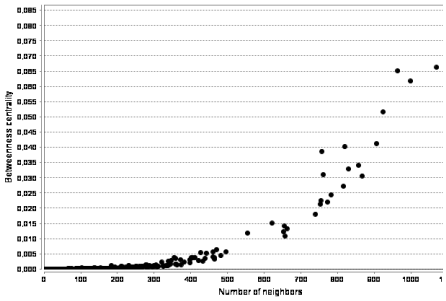

Fig. 5.a2 Docker Betweenness metric

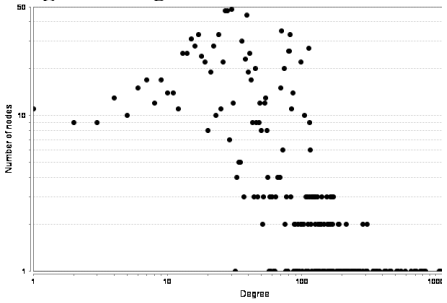

Fig. 5.b1 Angular Degree Metric

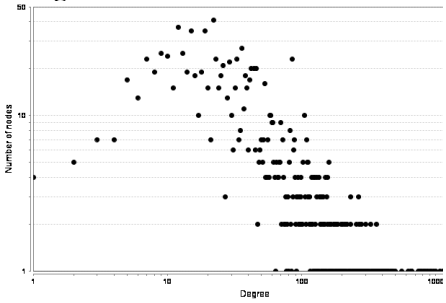

Fig. 5.b2 Docker Degree Metric

Fig. 5 Comparing Networks Metrics of Angular and Docker Projects

Table 3 summarizes SNA metrics for each network. Those metrics help to understand the nature of the project as well as the architecture. For instance, we observed a high density for Docker project probably meaning cohesive architecture of this project.

Table 3. Network SNA Mertics

Projects	Density	Centrality	Avg. Neighbours	Clustering Coef.
JQuery	0.252	0.725	61.197	0.834
Angular	0.048	0.918	66.428	0.897
Docker	0.062	0.794	81.385	0.874

Table 4 shows intrinsic SNA metrics emphasizing characteristics of interactions between contributors such as degree and centrality metrics (betweenness and closeness).

Table 4. Contributors SNA metrics

Projects	Degree	Betweenness	Closeness	Clustering Coef.
JQuery	[1- 236] Median =48	[0-0.8]	[0.46-0.97]	[0-1]
Angular	[1-1343] Median=38	[0-0.16]	[0.35-0.96]	[0-1]
Docker	[1-1122] Median=38	[0-0.08]	[0.41-0.87]	[0-1]

6 Discussion

Review assignments are not sufficient to explain comments flows in a project. Figure 6 shows the distribution of contributors pre-assigned for code reviews (yellow) within the overall contributors' network. Nerveless, we have seen quite lot of code reviews carried out by other contributors with different degrees of centrality, which is not necessarily problematic but may indicate areas where the review assignment was not supported by either awareness or cross-functional knowledge or the distribution of domain knowledge in the core team. OSS Team leads can optimize the team configuration when forming new teams, especially for the code review activity.

The core contributors are assumed to be structurally more central, in the contributors' networks, than other contributors. They have enough either awareness or knowledge about the product to manage other developers' contributions.

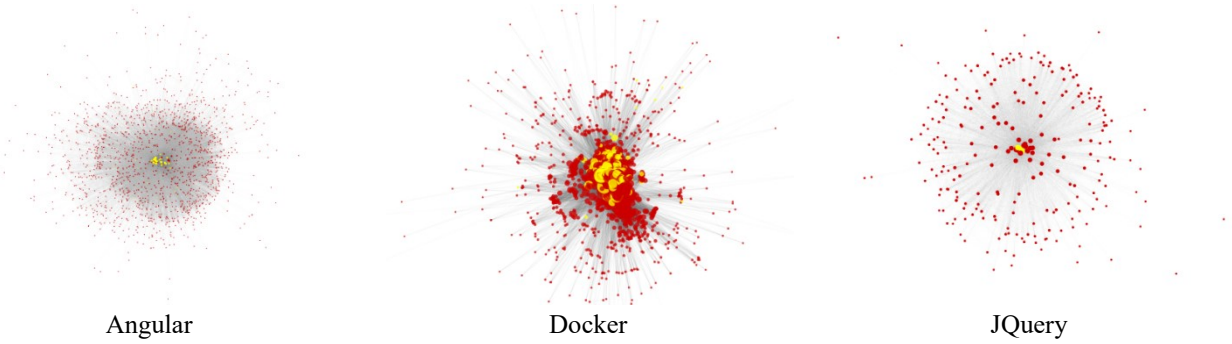

Fig. 6 Assigned Contributors for Reviewing Source Code

RQ4. *What kind of Knowledge is transferred?*

We manually classify comments according to technical or domain knowledge. Another category emerged throughout our classification process: Awareness. The majority of knowledge transfer is about Awareness (46.3%), then technical (34.5%) generating large contributors' debates and domain knowledge (19.1%). For example, contributor 13286 in Angular project commented on an implementing approach that he perceived as an anti-pattern:

[*I'm not quite sure why you're against this. The job of `inject` is to inject a function, as its name implies. Not inject a function and eliminate its return value. I would argue, instead, that is an anti-pattern of function decorators. It's confusing and unnecessary….]* 13286.

7 Threats to Validity

Construct Validity - In this paper, we adopt co-edited files as a heuristic to build the graph of contributors' networks. We do not consider the time frame such as co-edition within one month or under releases. In fact, we could rely on comments for SNA instead of co-edited files. However, focusing only on comments will hide the big analysis of all socio-interactions. Furthermore, our heuristic based on file co-edition does not consider the amount of LOC the contributors make. However, file editing is considered by many studies as a fine-grained enough indicator of developers' collaboration [2]. Furthermore, we assume that all communications occur with either review requests or comments within the review process. We cannot assume that developers on GitHub are not using an external social media or mailing list to communicate.

Internal Validity- We are aware that we might miss transitive dependencies between technical elements. For instance, changing the framework on which depends many files is unseen such as a technical interaction. Moreover, software development is dynamic, and as contributions are made over time, the nature of the socio-interaction changes. We mitigated this threat by studying multiple open source projects, using different languages, within the GitHub community. Furthermore, our analysis is time-agnostic. Since contributors are changing over time, the number of core developers may vary as well. We plan to conduct a temporal analysis of core contributors in future work to get more insights on how those contributors rich their actual position in the Network.

External Validity- In this study, we choose three projects which therefore might limit the generation of our results. However, we choose carefully mature and long-lived projects running in different languages and with an amount of contributors ranging from 250 to 1403. We filtered away projects that have fewer than 250 contributors or fewer than 1,000 edited files to remove projects that are immature or without an underpinning socio-technical interaction, and thus alleviate potential bias.

8 Conclusion

In this paper, we have performed Social Network Analysis on three open source projects. We showed how knowledge is transferred between core contributors and peripherals when using code review activity. We build contributors' networks based on co-edited files and then we build sub-networks for contributors requesting code reviews, commenting on, and those performing the code reviews. SNA visualization makes the identification of the structural interactions analysis of those networks possible. We found that there is a strong correlation relationship between the degree centrality of contributors and their implication on knowledge and awareness transfer.

By understanding the knowledge flows between OSS collaborators, socio-technical interactions structure, OSS communities gain an increased ability to facilitate code reviews in their projects. We hope this will lead to software projects with more efficient knowledge transfer, less overhead of review assignment, and increased leverage of the software quality and teams' performance.

9 References

1. VonHippel, E. and G. VonKrogh, *Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science.* Organization Science, 2003. **14**(2): p. 209-223.
2. Dabbish, L., et al. *Social coding in GitHub: transparency and collaboration in an open software repository*. in *the conference on Computer Supported Cooperative Work*. 2012. Seattle, WA, USA.
3. Begel, A., R. DeLine, and T. Zimmermann, *Social media for software engineering*, in *FSE/SDP workshop on Future of software engineering research*. 2010: Santa Fe, New Mexico, USA. p. 33-38.
4. Yang, X., *Social network analysis in open source software peer review.* 2014: p. 820-822.
5. Yang, X., et al., *Understanding OSS Peer Review Roles in Peer Review Social Network (PeRSoN).* 2012: p. 709-712.
6. Bird, C., et al. *Latent Social Structure in Open Source Projects*. in *Proc. of the 16th Int'l Symp. on Foundations of Soft. Eng. (FSE' 08)*. 2008. Atlanta, Georgia.
7. Asundi, J. 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*. 2007.
8. Bissyande, T.F., et al. *Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub*. in *24th International Symposium on Software Reliability Engineering (ISSRE)*. 2013.
9. Baysal, O., et al. *The Influence of Non-Technical Factors on Code Review*. in *In Proc. of the 20th Working Conference on Reverse Engineering*. 2013. Koblenz, Germany.
10. Bacchelli, A. and C. Bird. *Expectations, outcomes, and challenges of modern code review*. in *Proc. of the 35th Int'l Conf. on Software Engineering (ICSE '13)*. 2013. San Francisco, CA, USA.
11. Kilamo, T., et al., *Knowledge transfer in collaborative teams: experiences from a two-week code camp*, in *36th Int'l Conf. on Software Engineering (ICSE '13)*. 2014: Hyderabad, India. p. 264-271.
12. Yarosh, S., et al., *I need someone to help!: a taxonomy of helper-finding activities in the enterprise*, in *Proc. of the 27th Int'l Conf. on Computer Supported Cooperative Work (CSCW' 13)*. 2013: Texas, USA. p. 1375-1386.

13. Meneely, A., et al. *Predicting Failures with Developer Networks and Social network Analysis* in *Int'l Symp.on Foundations of Soft. Eng.(FSE'11)*. 2011. Atlanta, Georgia.
14. Hossaina, L. and D. Zhub, *Social networks and coordination performance of distributed software development teams.* The Journal of High Technology Management Research, 2009. **20**(1): p. 52–61.
15. Cataldo, M. and J.D. Herbsleb, *Coordination Breakdowns and Their Impact on Development Productivity and Software Failures.* Transactions on Softw. Eng., 2013. **39**(3): p. 343-360.
16. Rigby, P.C. 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)*. 2011. Waikiki, Honolulu, USA.
17. Kwan, I., 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., 2011. **37**(3): p. 307-324.
18. Cataldo, et al. *Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools* in *Proc. the 20th Int'l Conf. on Computer Supported Cooperative Work*. 2006. Banff, Alberta, Canada.
19. Nam, K.K., M.S. Ackerman, and L.A. Adamic. *Questions in, Knowledge in?: A Study of Naver's Question Answering Community*. in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2009. Boston, MA, USA.
20. Kadushin, C., *Understanding Social Networks: Theories, Concepts, and Findings*. 2011: Oxford University Press.