2015 6th International Workshop on Emerging Trends in Software Metrics

Measuring and Understanding the Effectiveness of JIRA Developers Communities

Marco Ortu*, Giuseppe Destefanis¶, Mohamad Kassab‡, Michele Marchesi*
*DIEE, Department of Electronic and Electrical Engineering
University of Cagliari, Italy
Email: {marco.ortu,michele}@diee.unica.it
¶CRIM, Computer Research Institute of Montreal, Canada
Email: giuseppe.destefanis@crim.ca
‡The Pennsylvania State University, Penn State Great Valley, Malvern, PA
Email:muk36@psu.edu

***Abstract*—Tools for project management and issues/bugs tracking are becoming useful for governing the development process of Open Source software. Such tools simplify the communications process among developers and ensure the scalability of a project. The more information developers are able to exchange, the clearer are the goals, and the higher is the number of developers keen on joining and actively collaborating on a project. In this paper we present a preliminary empirical analysis of the communities-structure of developers in JIRA by analyzing 7 popular projects hosted in the repository. We analyze how these communities perform in terms of issue-resolution time of any given issue. The main contributions of this work are the confirmation of the existence of communities in developer networks, and the empirical finding that the issue resolution-time of any given issue is not correlated with the dimension of a developer community.**

I. Introduction

During the last decade, researchers have been exploring the effectiveness of Open Source software communities, and today, Open Source systems are no longer considered to be children of a lesser God. The quality reached by such systems is well known and recognised [9]. Faster Internet connections, smartphones, tablets and devices always connected to the Internet help open source developers to stay in touch with each other in order to generate software. Communication processes are a key-factor for open source paradigm development, and tools able to manage the communication within a group of people developing and creating something together are therefore vital. The communication aspect is central and is the key to correctly manage the development process. The knowledge of a project should always be easily accessible for the development team during the development process. When a new developer joins a development team, the better the communication process works, the faster the new developer can become productive and the learning curve can be reduced.

In this study we analysed the developers' structure for seven projects hosted in JIRA[1], a proprietary issue tracking product developed by Atlassian. JIRA provides bug tracking, issue tracking, and project management functions,and includes tools allowing migration from competitors. According to Atlassian[2], it is used for issue tracking and project management by over 25,000 customers around the world. An Issue Tracking System (ITS) is a repository used by software developers as a support for the software development process. It supports corrective maintenance activity like Bug Tracking systems, along with other types of maintenance requests.

Since JIRA is becoming more and more popular among developers, it is worthwhile to understand the impact of such a product on the structure of developer communities. Several studies have demonstrated that developers involved in the process of creating new open source software are organised, and that a clear structure exists. Linux, for example, is not the result of a disorganised process executed by disorganised developers. Their success in developing a very complex system, used even by NASA[3], was not simply based on luck. However, this doesn't mean that open source paradigms always lead to the development of quality products. Questions such as the following arise about open source communities: "Can I trust something produced by people working for free, driven only by passion and pleasure?" [11] "Can software developed without a commercial plan and strict deadlines be of high quality?" "How can developers work efficiently without central project coordination?" [8].

Diseconomies of scale can affect the communication process of a group of developers working together. When the number of developers increases (e.g. when newcomers join a project), the structure becomes more difficult to manage, and it may be difficult to keep track of who is doing what. Tools such as JIRA help to reduce co-ordination problems, to increase the level of communication and to scale up the project by reducing releases' time.

The main goal of this study was to provide empirical evi-

[1]https://www.atlassian.com/software/jira

[2]http://blogs.atlassian.com/2013/05/why-people-choose-jira-6/

[3]http://www.linux.com/news/featured-blogs/191-linux-training/711318-linux-foundation-training-prepares-the-international-space-station-for-linux-migration

2327-0969/15 $31.00 © 2015 IEEE
DOI 10.1109/WETSoM.2015.10

dence showing whether developers are organised with defined structures/teams, and if such teams perform differently in terms of productivity. We define the productivity of a team as the average fixing time for any given issue.

We answer the following research questions:

RQ1: Does the open source software developer's network graph contain communities?

Open Source projects hosted in JIRA have communities. Results show that the 7 open source projects analyzed have a number of communities, ranging from 8 to 16.

RQ2: Are there differences in fixing time between communities?

While JIRA Developer Communities have different average issue fixing times, the distribution of issue types and priority is similar across the communities.

The rest of the paper is organized as follows: in Section II we give an overview of the related works, in Section III we explain the dataset we used and the methodology, in Section IV we discuss our results and we conclude with Section V and VI by analyzing the threats to validity and summarizing our findings.

II. Related Work

The *bazaar* is one of the most used figurative expression to explain the Linux's development style, and several studies have used the "Cathedral-Bazaar" metaphor [18] to describe the properties of a (differently) organized approach of development [2] [3] [5] [6] [13].

The structure of such decentralized development has been deeply analyzed by many researchers. Small world phenomenon and scale free behaviours are found in the SourceForge development network by Xu et al [25], considering two developers socially related if they participate in the same project. Wagstrom et al. [23] congregated empirical social network data from blogs, email lists and web sites, to build models of development used to simulate how users joined and left projects. Ehrlich et al [10] used social network analysis to study how individuals in global software development teams detect and gain expertise.

Madey et al. [15] analyzed structural data on over 39,000 open source projects hosted at SourceForge.net involving over 33,000 developers, and hypotized that open source software development could be modeled as self-organizing, collaboration, social networks. Project sizes, developer project participation, and clusters of connected developers are analyzed. The authors found evidence to support their hypothesis in the presence of power-law relationships on project sizes (number of developers per project), project membership (number of projects joined by a developer), and cluster sizes.

Crowston et al. [7] examined 120 project teams from SourceForge, detecting that open source development teams vary in their communications centralization, from projects centered on one developer to projects highly decentralized and exhibit a distributed pattern of conversation between developers and active users. Larger teams tend to have more decentralized communication patterns. Other researchers[16] examined the structure of developer collaboration with the developer network in order to predict failures at the file level. Failure prediction models were developed using test and post-release failure data from two releases of a mature Nortel networking product, then validated against a subsequent release.

Sharif et al. [20] showed that open source developers are "implementation centric" and "team focused" in their use of mailing lists.

Other studies have analyzed the structure of the open source software communities to understand social aspects in development Steinmacher et al. [21][22], identified 20 studies providing empirical evidence of barriers faced by newcomers to OSS projects while making a contribution. They identified 15 different barriers, which we grouped into five categories: social interaction, newcomers previous knowledge, finding a way to start, documentation, and technical hurdles. The authors also classified the problems with regard to their origin: newcomers, community, or product. Zhou et al. [27] found, using issue tracking data of Mozilla and Gnome, that the probability for a new joiner to become a Long Term Contributor is associated with her willingness and environment. Shah [19] explored the motivations of participants from two software development communities and finds that most participants are motivated by either a need to use the software or an enjoyment of programming. The latter group, hobbyists or enthusiasts, are critical to the long-term viability and sustainability of open source software code: they take on tasks that might otherwise go undone, are largely "need-neutral" as they make decisions, and express a desire to maintain the simplicity, elegance, and modularity of the code. The motives of hobbyist evolve over time; most join the community because they have a need for the software and stay because they enjoy programming in the context of a particular community.

Ortu et al. [17] studied 14 open source software projects developed using the Agile board of the JIRA repository. They analysed all the comments committed by the developers involved in the projects and we studied whether the politeness of the comments affected the number of developers involved over the years and the time required to fix any given issue. Results indicated that the level of politeness in the communication process among developers does have an effect on both the time required to fix issues and the attractiveness of the project to both active and potential developers. The more polite developers were, the less time it took to fix an issue, and, in the majority of the analysed cases, the more the developers wanted to be part of project, the more they were willing to continue working on the project over time.

In this paper we perform a more general study by measuring

and describing communities of developers within JIRA.

III. Experimental Setup

A. Dataset

We built our dataset by collecting data from the Apache Software Foundation Issue Tracking system, JIRA[4]. We mined the Issue Tracking Systesm (ITS) of the Apache Software Foundation by collecting issues from 2002 to December 2013. Table I shows the corpus of the 7 projects selected for our analysis, highlighting the number of comments recorded for each project and the number of developers involved. We selected projects with the highest number of comments.

TABLE I: selected projects statistics

Project	# of comments	# of developers
HBase	91016	951
Hadoop Common	61958	1243
Derby	52668	675
Lucene Core	50152	1107
Hadoop HDFS	42208	850
Hive	39002	757
Hadoop Map-Reduce	34793	875

B. Developer's Network

We extracted the developer network based on the data contained in JIRA. JIRA allows users to post issues (with a set of properties such as maintenance type, priority, etc.) and to comment on them. We built developer network modeling nodes, which represent developers, and edges from *node A* to *node B*, which represent when *developer A* was commenting on *developer B's issue*. In this manner, we obtained a directed network.

We then used Gephi[5] [1] to analyze the obtained network. Gephi is an interactive visualization and exploration tool. We ran the modularity algorithm, based on the algorithms developed by Blonde [4] and Lambiotte[14], in order to obtain the network communities. We finally measured the obtained networks.

Figure 1 shows an example of network graph we obtained. Node size represents the number of issues posted by a developer, and the edge size from *node A* to *node B* represents the number of comments posted by *node A* in response to issues posted by *node B*.

IV. Result and Discussion

A. Does the open source software developer's network graph contain communities?

Motivation. Understanding developer structure in open source projects allows both work teams and management to have better control over the whole project. Both issue triage and workload scheduling may benefit from having a clear view of how the developers are organized and how productivity is spread across work teams. For this reason

[4]https://www.atlassian.com/software/jira
[5]http://gephi.github.io/

Fig. 1: example of developer's graph extracted from Lucene-Core project

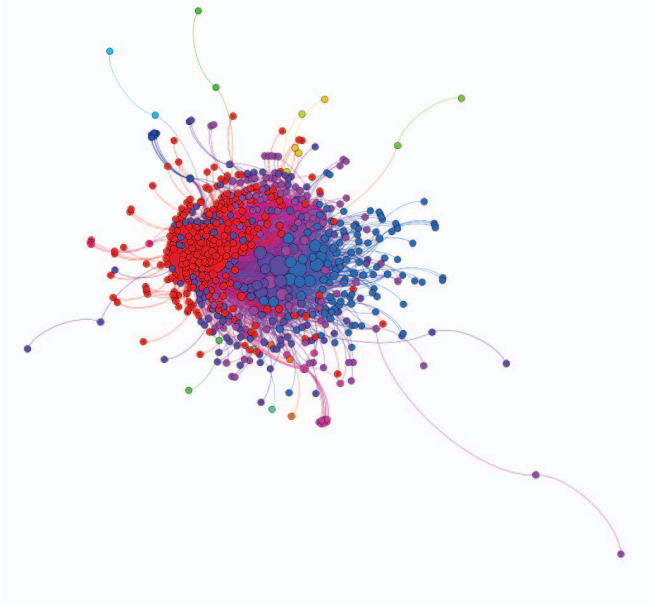

Fig. 2: example of developer's graph extracted from Derby project

TABLE II: selected projects network statistics

Project	Modularity	Avg. Degree	Avg. Clustering Coeff.	# of Communities
HBase	0.283	5.552	0.459	8
Hadoop Common	0.334	5.617	0.296	15
Derby	0.193	5.170	0.484	11
Lucene Core	0.269	3.493	0.339	14
Hadoop Map-Reduce	0.332	5.040	0.242	12
Hive	0.333	4.358	0.287	16
Hadoop HDFS	0.284	5.275	0.311	8

our first research question aims to explore the presence of communities in JIRA developer networks.

Approach. We created the developer network graph as described in III-B. Each Node represents a developer who posted/commented on an issue, and each Edge represents a developer who commented on another developer's issue. We applied the modularity algorithm [14] to obtain developer communities.

Findings. Open Source Projects hosted in JIRA do have communities.
Table II shows the network metrics for the analyzed projects. The Modularity metric represents the fraction of the edges that fall within the given groups minus the expected such fraction if edges were distributed at random. It is positive if the number of edges within groups exceeds the number expected on the basis of chance. For a given division of the network's vertices into some modules, modularity reflects the concentration of edges within modules compared with the random distribution of links between all nodes regardless of modules. Avg. Degree represents the average node degree (as the sum of in-degree and out-degree). The Avg. Clustering Coeff. metric quantifies on average how close node neighbours are to being a clique (complete graph) [24]. The last column represents the number of communities found by the algorithm [14]. Results show that the 7 open source projects analyzed have a number of communities, ranging from 8 to 16.

B. Are there differences in fixing time between work teams?

Motivation. Based on the findings related to the first research question, we know of the presence of communities in JIRA developer networks. It is also of interest to analyze these communities in order to understand if and how the workload is distributed. This kind of information can be useful during issue triaging and scheduling of workload distribution. For example if we find a community resolving a high percentage of maintenance issues, then it is likely that a maintenance issue will be assigned to them, or if there is a community with a fast average issue resolution time, then this community can be assigned bug issues when the release date is imminent.

Approach. We used the developer network obtained in the first research question. For each project, we analyzed its developer communities by evaluating the average issue resolution time, the number of issues resolved, and the distribution of maintenance type and priority of fixed issues.

Findings. While JIRA Developer Communities have different average issue fixing times, the distribution of issue types and priority is similar across the communities.
Tables III to IX show, for each project, the number of developers belonging to a particular community, the number of issues resolved, and the average issue fixing time.

TABLE III: Derby communities statistics

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
6	2	1	5.6
4	20	186	159.4
13	6	2	175
3	245	2919	214.7
2	109	879	216.3
1	143	2498	242.1
12	3	2	258.2
0	90	372	260.8

TABLE IV: Hadoop Common communities statistics

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
9	2	1	0.7
10	5	1	10.9
6	26	221	52.3
3	141	2359	71.8
2	502	2162	86.1
0	293	2992	120.9
1	60	207	122.2
4	84	215	158.8
5	31	363	193.7

The first result is related to Pareto's law (20% of developers doing 80% of the issue resolution) [12]. There are only a few communities taking care of the majority of issues. These communities have a different average resolution time, and this number is independent from the community size and from the number of fixed issues.

We evaluated the Pearson's correlation coefficient between the average issue fixing time and the number of issues resolved, and between the average issue fixing time and the community size. We found a rather weak correlation (<0.3) with the only exception being for Hadoop Map/Reduce, in which the correlation between average issue fixing time and the community size was 0.68. This result indicates that the

average issue resolution time is a property of a developer community, and it does not depend on the community size.

There is a great difference in the number of issues resolved by the communities and the average issue resolution time per community. This fact is consistent across all of the 7 projects analyzed. In order to understand how the productivity is distributed across the developer communities, we calculated, for each community, the fixed issue distribution of maintenance type and priority. Figures 3 and 4 show, for each project, the distribution of maintenance type and the priority of the issues fixed by a community. For each project, the bar-chart on the left represents the distribution of fixed issue maintenance type, and the bar-chart on the left represents the distribution of fixed issue priority.

For almost all the projects, these two distributions are similar; namely, there are no specialized communities, for example, communities solving mostly Bug with Critical priority. We can then conclude that the average issue resolution time is property of the community and does not depend on the community size, the number of fixed issues, or the maintenance type and priority.

TABLE V: Hadoop HDFS communities statistics

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
4	3	1	10.7
6	113	1621	45.1
2	199	1803	66
1	146	528	218.5
0	234	595	317.7

TABLE VI: Hadoop Map/Reduce communities statistics

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
5	108	519	66.6
2	152	1862	70.8
7	57	84	119.4
0	113	412	128.8
1	169	1049	152.6
3	209	445	471.8

TABLE VII: HBase communities statistics

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
0	62	517	45.5
2	153	1629	47.8
6	43	378	60.2
5	91	578	64.3
1	143	1494	69.7
3	286	3215	85.3
4	163	1707	90.2

TABLE VIII: Hive communities statistics

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
14	2	1	0.09
9	2	1	0.9
12	2	1	3.2
2	2	1	7.3
8	4	1	14
6	2	2	14.9
1	126	1105	39.4
0	229	967	96.9
7	267	2221	97.9
3	92	383	98.1
10	2	2	100.9
5	50	113	133.2
11	2	1	379.4

TABLE IX: Lucene - Core communities statistics

Community Id	Community Size	# of Fixed Issues	AVG Fixing Time [Days]
7	94	1	2.9
14	2	89	28.4
17	211	608	112.7
8	8	7	147.1
13	275	377	208.5
11	86	9	251.6
10	95	8	300.6
20	70	3	323.5
19	193	7	436.9
16	79	46	549.7

V. Threats to validity

We now discuss the threats to validity of our study, following common guidelines for empirical studies [26]. Construct validity threats concern the relation between theory and observation. For the calculation of the issue resolution time, we have not taken into account the complexity of a given software, or the complexity of a specific sub-system of a software. This simplification could affect our findings. Our model is a starting point, but in order to obtain more precise it would be necessary to insert in the model a complexity factor. Developers working on the core part of a e-commerce system, would need more time to solve a given issue related to a part of the system that performs payment-operations, than developers involved in the same project, but working on a web-page that visualizes an item in the basket.

Threats to internal validity concern our selection of subject systems, tools, and analysis method. With respect to the system studied in this work we considered only 7 systems hosted in JIRA.

In our study, we created the developer network graph in which each Node represents a developer who posted/commented on an issue, and each Edge represents a developer who commented on another developer's issue. There might be other ways to define developer networks, considering for example temporal locality, or link between developers editing the same sections of code. Graphs built considering these factors would lead to obtain different results.

Threats to external validity are related to generalisation of

Fig. 3: examples of communities distributions of issue's maintenance types and priorities

our conclusions. Our results are not meant to be representative of all projects hosted in the repository and we have analyzed only the JIRA issue-traking system.

VI. Conclusion and Future Works

In this study we analysed the developer networks of 7 open source projects hosted in JIRA. We found the presence of developers communities for all the projects analyzed. This result agrees with other studies about the social structure of open source projects. We further investigated how the productivity is distributed across the communities. To measure the productivity we considered factors such as the community size, the number of fixed issues, the distribution of fixed issues's maintenance type and priority, and the average issue fixing time. As a first result we found the presence of Pareto's law (20% of developers doing 80% of the work), there are a few developers that post and comment the majority of issues.

The presence of Pareto's law need further investigation, one may expect this is due to the nature of JIRA issue tracking system and the structure of the open source community and how we built the developer's working network. For example there may be a group of core developers devoted to report issues. We analysed the average community issue fixing time and we found it varies across the communities. We showed the independence of the average issue resolution time from the other factor considered, such as the community size and the kind of issues maintenance and priority. There are many other factors that may impact the average community issue fixing time, for example software component involved in the issue resolution or the portion of code involved. In order to better understand the communities productivity we will analyse these factors in future study.

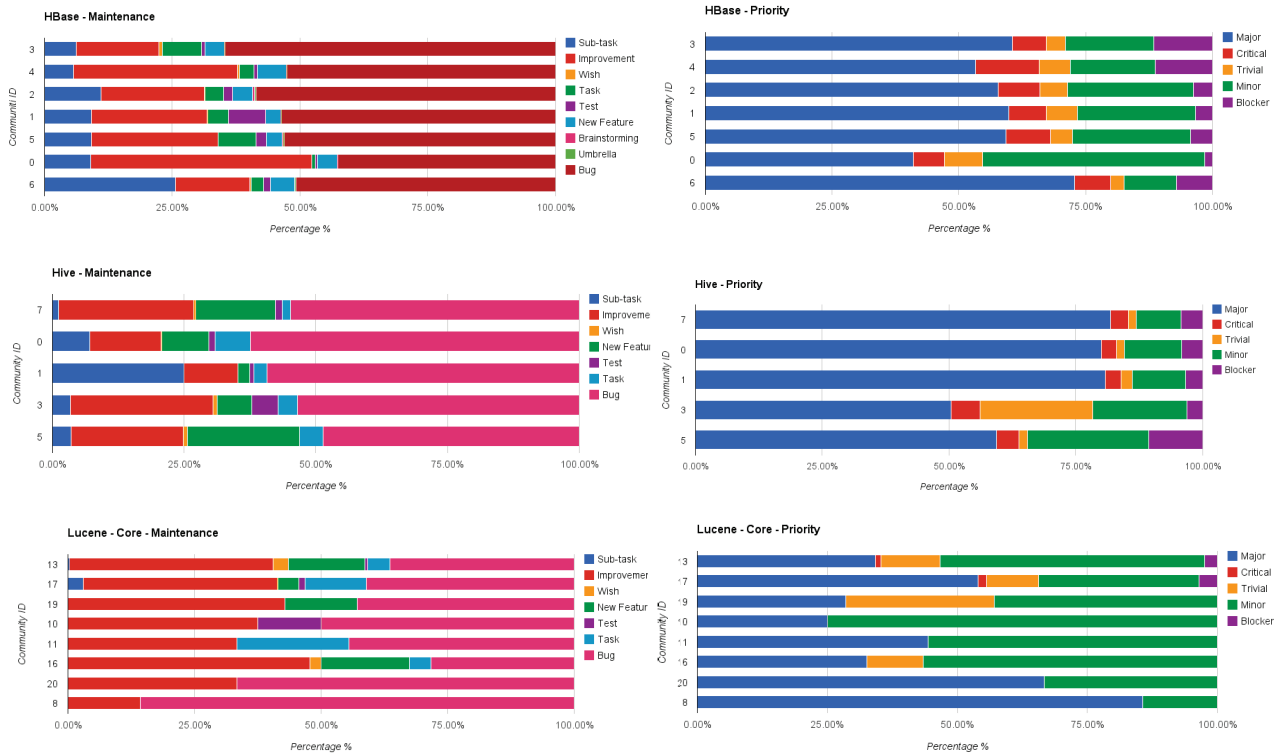

Fig. 4: examples of communities distributions of issue's maintenance types and priorities

This study is a starting point to better understand how groups of developers perform when working together. The issue resolution time is a useful metric that represents the productivity of a certain community. A promising study could be related to the optimal number of developers involved in a project. An interesting future experiment could be to investigate if diseconomies of scales occur in development activity. Considering this "dilemma" as an operational research problem, could lead to promising results.

REFERENCES

[1] M. Bastian, S. Heymann, M. Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[2] N. Bezroukov. A second look at the cathedral and the bazaar. *First Monday*, 4(12), 1999.

[3] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Chapels in the bazaar? Latent social structure in OSS. In *16th ACM SigSoft International Symposium on the Foundations of Software Engineering, Atlanta, GA*, 2008.

[4] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[5] A. Bonaccorsi and C. Rossi. Why open source software can succeed. *Research policy*, 32(7):1243–1258, 2003.

[6] A. Capiluppi and M. Michlmayr. From the cathedral to the bazaar: An empirical study of the lifecycle of volunteer community projects. In *Open Source Development, Adoption and Innovation*, pages 31–44. Springer, 2007.

[7] K. Crowston and J. Howison. The social structure of free and open source software development (originally published in volume 10, number 2, february 2005). *First Monday*, 2005.

[8] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison. Self-organization of teams for free/libre open source software development. *Information and software technology*, 49(6):564–575, 2007.

[9] G. Destefanis, S. Counsell, G. Concas, and R. Tonelli. Software metrics in agile software: An empirical study. In *Agile Processes in Software Engineering and Extreme Programming*, pages 157–170. Springer, 2014.

[10] K. Ehrlich and K. Chang. Leveraging expertise in global software teams: Going outside boundaries. In *Global Software Engineering, 2006. ICGSE'06. International Conference on*, pages 149–158. IEEE, 2006.

[11] A. Hars and S. Ou. Working for free? motivations of participating in open source projects. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, pages 9–pp. IEEE, 2001.

[12] R. Koch. *The 80/20 principle: the secret to achieving more with less*. Crown Business, 2011.

[13] S. Krishnamurthy. Cave or community?: An empirical examination of 100 mature open source projects. *First Monday*, 2002.

[14] R. Lambiotte, J.-C. Delvenne, and M. Barahona. Laplacian dynamics and multiscale modular structure in networks. *arXiv preprint arXiv:0812.1770*, 2008.

[15] G. Madey, V. Freeh, and R. Tynan. The open source software development phenomenon: An analysis based on social network theory. *AMCIS 2002 Proceedings*, page 247, 2002.

[16] A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 13–23. ACM, 2008.

[17] M. Ortu, G. Destefanis, M. Kassab, S. Counsell, M. Marchesi, and R. Tonelli. Would you mind fixing this issue? qn empirical analysis of politeness and attractiveness in software developed using agile boards. In *Agile Processes, in Software Engineering, and Extreme Programming*, pages 129–140. Springer, 2015.

[18] E. S. Raymond. The cathedral and the bazaar, 2000. *Available from*

World Wide Web: http://www. catb. org/~ esr/writings/cathedral-bazaar, 2004.

[19] S. K. Shah. Understanding the nature of participation & coordination in open and gated source software development communities. In *Academy of Management Proceedings*, volume 2004, pages B1–B5. Academy of Management, 2004.

[20] K. Y. Sharif, M. English, N. Ali, C. Exton, J. Collins, and J. Buckley. An empirically-based characterization and quantification of information seeking through mailing lists during open source developers' software evolution. *Information and Software Technology*, 57:77–94, 2015.

[21] I. Steinmacher, T. U. Conte, M. Gerosa, and D. Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*, pages 1–13, 2015.

[22] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59:67–85, 2015.

[23] P. Wagstrom, J. Herbsleb, and K. Carley. A social network approach to free/open source software simulation. In *Proceedings First International Conference on Open Source Systems*, pages 16–23, 2005.

[24] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, 1998.

[25] J. Xu, Y. Gao, S. Christley, and G. Madey. A topological analysis of the open souce software development community. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 198a–198a. IEEE, 2005.

[26] R. K. Yin. Case study research design and methods third edition. *Applied social research methods series*, 5, 2003.

[27] M. Zhou and A. Mockus. What make long term contributors: Willingness and opportunity in oss community. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 518–528. IEEE Press, 2012.