2-2019

Evolutionary trends in the collaborative review process of a large software system

Subhajit DATTA
Singapore Management University, subhajitd@smu.edu.sg

Poulami SARKAR

Evolutionary Trends in the Collaborative Review Process of a Large Software System

Subhajit Datta
Singapore University of Technology & Design
Singapore
subhajit.datta@acm.org

Poulami Sarkar
PES Institute. of Technology
India
poulamisarkar@acm.org

ABSTRACT

In this paper, we study the evolutionary trends in the collaborative review process of a large open source software system. As expected, the number of reviews, the number of reviews commented on, as well as the number of reviewers, and the interactions between them show increasing trends over time. But unexpectedly, levels of clustering between developers in their interaction networks show a decreasing trend, even as connections between them increase. In the context of our study, clustering is an indicator of developer collaboration, whereas connection points to how intensely developers work together. Thus the trends we observe can inform how developer interactions become concentrated around specific units of work as the project progresses. The dichotomy between the simultaneous increase in connection and decrease in clustering also points to the interplay between collective and individual efforts in the review process, and the distinct nature of peer review in the software development life cycle.

CCS CONCEPTS

• **Software and its engineering → Programming teams**;

KEYWORDS

Peer review, network science, connection, clustering

ACM Reference Format:
Subhajit Datta and Poulami Sarkar. 2019. Evolutionary Trends in the Collaborative Review Process of a Large Software System. In *12th Innovations in Software Engineering Conference (formerly known as India Software Engineering Conference) (ISEC'19), February 14–16, 2019, Pune, India.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3299771.3299792

1 INTRODUCTION AND MOTIVATION

Software development represents a confluence of diverse skills and activities that is rare in other large scale industrial enterprises. Across the phases of *inception*, *elaboration*, *construction*, and *transition* developers are involved in related but distinct tasks [13]. Almost all major software systems – open source, as well as proprietary – are now developed by large and distributed teams of developers. Members of such teams are located across geographies and time-zones and have very little synchronous contact. In such situations, development processes are organized to facilitate exchange of observations, opinions, insights towards more effective team outcomes. Peer review is one such process that occupies a central role in ensuring the delivery of software solutions within cost, time, and quality constraints. We posit that studying the dynamics of peer review over the progression of the software development life cycle can offer helpful insights on some of the enduring concerns of software development in the large.

In the peer review process, a code unit is reviewed by multiple reviewers, who record and track their observations using some review management system such as Gerrit[1] or Rietveld[2]. These observations are commented upon by peer reviewers, and a consensus is hopefully reached through this co-commenting process, on whether the code unit can be integrated in the main body of code. Subsequently, the code unit is tested and then approved for merging, or abandoned; followed by closure of the review.

In a sense, peer review is unlike any other software development activity. As in any engineering enterprise, in software development too, work products of individual developers or sub-teams need to *fit in* to the structure of the larger system being built. This requires such a work product, as well as the thinking and effort that goes into its making, to *conform* to the interfaces defined by other work products. And complementarily, the work product also needs to function within the ambit of the interfaces defined for it. But conformance is contrary to the very spirit of review. As in the review of scientific papers, most effective code review outcomes are reached when reviewers offer their own points of view, unprejudiced by others. This distinct characteristic of the review process is likely to influence the mores of interaction between developers engaged in peer review of code.

Developer interaction in any large software ecosystem is multifaceted. From the project governance point of view, the extent of connections between developers, as well as the levels of their clustering, are particularly important. The former is an indicator of how developers are positioned to exchange information between them, while the latter signifies how closely they are working together. With this background, we investigate the following *research question* in this paper: **How does the extent of connection between developers engaged in peer review and their level of clustering vary across time in a large development ecosystem?**

In the next section we outline the study setting and methodology. Subsequently, the results are presented along with the discussion of their implications. We next highlight the threats to the validity of

ISEC'19, February 14–16, 2019, Pune, India

ACM ISBN 978-1-4503-6215-3/19/02…$15.00
https://doi.org/10.1145/3299771.3299792

[1] https://www.gerritcodereview.com/
[2] https://github.com/rietveld-codereview/rietveld

the results, followed by a brief overview of related work. The paper ends with a summary of the study along with our conclusions.

2 STUDY SETTING AND METHODOLOGY

2.1 Network construction

Ecosystems of interacting individuals have been widely studied using the network paradigm in various domains; networks offer a range of metrics to calculate features of interest for different kinds of studies [15]. In our context, we define a *review interaction network* (*RIN*) whose vertices are developers, and two vertices are connected by an undirected edge, if both the corresponding developers have co-commented on at least one review item.

The average number of edges per vertex, that is, the average degree is a measure of a network's interconnectedness [2]. For a network with V vertices and E edges, average degree can be calculated as $\frac{2E}{V}$. In our context, we take the average degree of the *RIN*s to reflect the extent of *connection* between developers participating in the peer review process. The average path length of the networks indicate the level of *separation* between the developers, and it is calculated by computing the mean of the shortes path lengths between all pairs of vertices in a network [2].

In a network, the clustering coefficient (C_v) for a vertex v is defined as follows: If v has a degree of k_v, that is there are k_v vertices directly linked to v, the *maximum* number of edges between these k_v vertices is $\frac{k_v(k_v-1)}{2}$. If the *actual* number of such edges existing is N_v, then $C_v = \frac{2N_v}{k_v(k_v-1)}$. Thus, the clustering coefficient of a vertex is the ratio of the actual number of edges existing between its neighbors and maximum number of such edges that can exist [2]. For an entire network, the clustering coefficient is the average clustering coefficient of all its vertices. For this study, we take the clustering coefficient of *RIN* to represent the level of *clustering* between developers as they interact in the peer review process. Such clustering can be a strong indicator of how developers collaborate. Furthermore, we also examine the *closeness* between the developers using the closeness centralization of *RIN*. Closeness centrality of a vertex in a network is the sum of the length of the shortest paths between that vertex and all other vertices in the network [2]. Here, *RIN*'s centralization indicates how central its most central developer is, compared to other developers.

2.2 Accessing and filtering data

In this project we have used peer-review data from the Chromium[3] project for our analysis [12]. This data was obtained from the Chromium database which is available on the internet for public use[4]. In this study, we considered 209617 reviews spanning across approximately four years (1519 days), which have been commented upon by 3102 distinct developers; there were 826398 such comments. After extracting the raw data from its source, it is stored in a MySQL[5] database with tables organized around entities such as *developer*, *review*, *comment* and *approval*.

For our analysis we generated review interaction networks (*RIN*s) as defined earlier, for 50 time-steps, each of 30 days duration. The edge weights are the number of common reviews commented upon by pairs of developers connected by the edge. The network was generated using the *igraph*[6] library and edge weights were calculated using the *Pandas*[7] library in Python.

[3]http://www.chromium.org/
[4]http://sdlab.naist.jp/reviewmining/data/chromium-reviews-20121030.zip
[5]https://www.mysql.com/

	Mean	Median	Standard deviation	Skewness
Reviews	4192.34	4281.5	1630.1	0.07
Comments	16527.96	14675	9555.52	0.6
Developers	62.04	63.5	21.5	-0.05

Table 1: Descriptive statistics of reviews, comments, and developers commenting on reviews across time-steps.

2.3 Understanding the data

Before discussing the results, let us examine the nature of our data-set in some detail. With reference to Table 1 and Figure 1, we observe that across the time-steps, the numbers of reviews, comments, and developers commenting on reviews do not have highly skewed distributions. This implies that the intensity of development activities was notably low or high in few time-steps, while being moderate in most of the time-steps.

3 RESULTS AND DISCUSSION

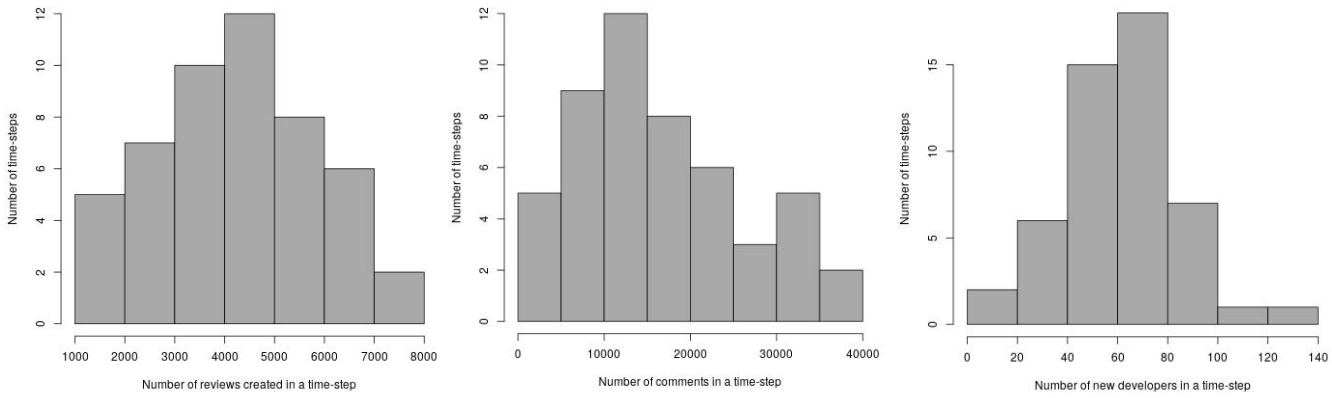
We will now present our results and examine their implications in the context of this study.

3.1 Observations

Figures 2 and 3 offers evidence of the growing developer attention in review activities as the project progressed. As shown in Figure 2, while the number of reviews increased over time, the number of comments increased at a far faster rate. We observe that the rate of increase of the number of reviews is fairly constant, almost linear. However, the number of comments grows at a far faster rate in the latter time-steps than the earlier ones. This is not unexpected, as with the maturity of the system, the code units being reviewed encapsulate more complex functionality, and hence entail more intense discussions among the reviewers. This intensity of discussion leads to more co-commenting on reviews, which manifests in rapidly increasing edges in the *RIN*s of the latter time-steps, as evident in Figure 3. While the rate of growth of the number of edges is considerable, it important to note that the curve for the number of edges in Figure 3 is far flatter than the $O(n^2)$ upper bound for the maximal number of interactions between n vertices in a network. *This signifies that developers are segregated into smaller groups, each working on specific sub-systems, with their interactions being largely contained within the groups.*

However, as development progresses, such interactions increase, as is exhibited in the plot for connections, shown in Figure 4. We note that there is more than three fold increase (from 9.7 to 31.18) in the average number of edges per vertex across duration of the study period. This increase is brought about by increasing instances of co-commenting on reviews. But interestingly, we also observe from

[6]http://igraph.org/python/
[7]https://pandas.pydata.org/



(a) Reviews (b) Comments (c) Developers

Figure 1: Distribution of number of reviews, comments, and developers in different time steps

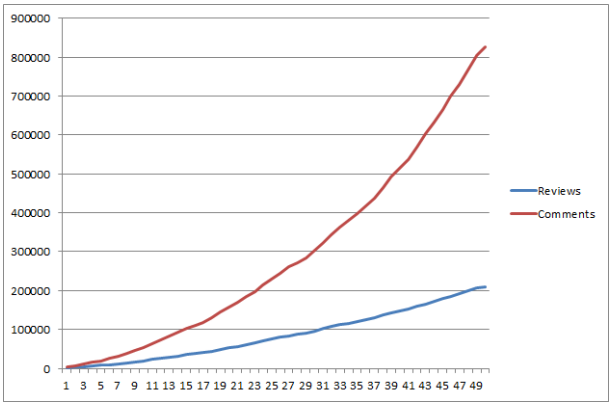


Figure 2: Number of reviews and comments across the time-steps

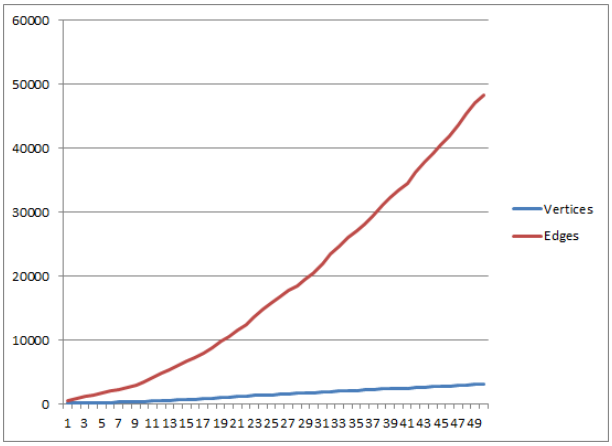


Figure 3: Number of vertices and edged of *RIN* across the time-steps

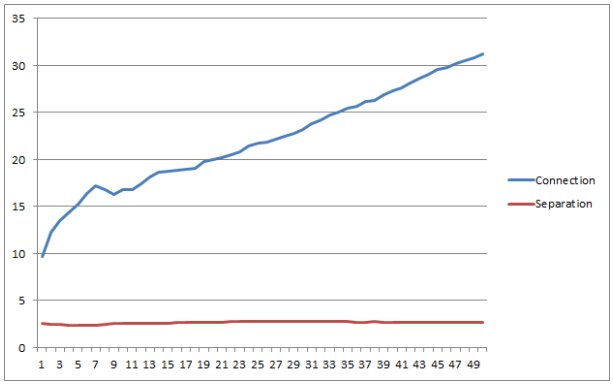


Figure 4: Developer connection and separation across the time-steps

Figure 4 that the separation between developers remains nearly constant (maximum being 2.78 and minimum being 2.38) over this regime of increasing connections.

With sharply increasing number of edges as well as connections between developers (Figures 3, 4), it is expected that the level of clustering will also increase over time. However, Figure 5 shows a *very different trend*. We observe that the level of clustering between developers rises for a very short while in the early time-steps, stays approximately the same (around 0.4) for the next few time-steps, and then steadily decreases to around 0.2 towards the end of the study period. Initially, the rate of decrease is relatively high, which settles to a slower rate with progressing time. The closeness centralization in Figure 5 shows a similar trend which peaks in the early time steps and then has an even sharper fall compared to the clustering. These observations indicate some interesting dynamics underlying the peer review process, that is manifested in counter-intuitive patters of developer interaction as the project moves from inception to transition; we discuss their implications below.

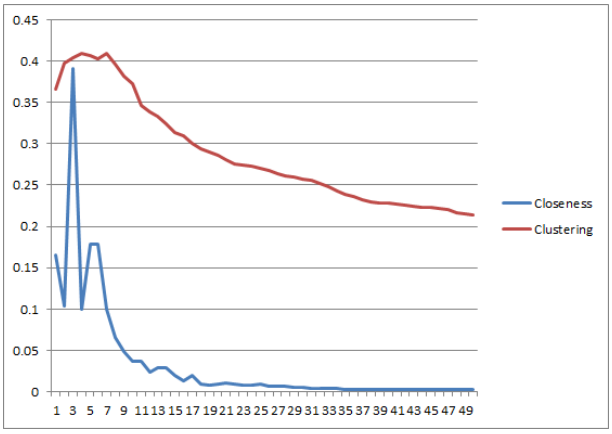


Figure 5: Developer closeness and clustering across the time-steps

3.2 Implications

We notice that even as the extent of connection between developers *increases* by a factor of three over the study period, their level of clustering *decreases* by a factor of two. With reference to the network metrics for connection and clustering as defined in Section 2.1, it appears that higher average degree is likely to contribute to higher clustering coefficient. So, the opposite trend we observe is unexpected. With reference to Figure 5 we additionally observe that the closeness centralization [2] peaks towards the beginning of the project, even as there are relatively fewer developers participating in the review process. However it decreases notably towards the end of the project, notwithstanding the fact that there are many more developers participating in the review process now. *This points to a higher fragmentation of the RINs away from a strong central presence in the latter time-steps, and complements the observations around reduced clustering.*

As evident from its definition in Section 2.1, triadic closure is a key element in the definition of the clustering coefficient. In a network context, triadic closure can signify collaboration between individuals [11]. So, the evidence of lower clustering points to developers collaborating less, even as they are getting connected to more developers on average, as the project progresses. This reflects on a key characteristic of the peer review activity.

As emphasized in Section 1, effective peer review of code calls for a delicate balance between the absorption of contextual information and sharing of individual perspectives. In early time-steps, with the scope of the system being still defined, and a relatively smaller number of developers actively engaged in the project, it is natural for each developer to be collaborating more closely with others. As the system's functionality widens in reach and depth, individual developers are constrained to have a narrower and deeper focus on their immediate review items, even as they need to remain connected to a wider pool of developers. *The trends of increasing connection, coupled with decreasing clustering, as seen in Figures 4 and 5 are manifestations of these dynamics of the peer review process. In a large and complex system such as the one we are studying, such interplay of connection and clustering is essential for the effectiveness of the peer review process.*

3.3 Utility

Even as we share preliminary results which we plan to investigate further along the lines identified in Section 3.4, the observations and implications discussed above indicate how the results can be utilized.

State of the art collaborative development environments (CDE) are now tuned towards facilitating closer connections among developers with a view to enabling higher clustering. While this may be helpful in some development activities, our results indicate that allowing developers relative isolation as they work on review items can also be beneficial for the effectiveness of the review process.

In an earlier study, we investigated the importance of isolation for developers as they engage in the peer review process [9]. Observing the clustering trends over time can offer insights on whether developers are engaging too closely with one another as they review code, thus raising possibilities of the detrimental influences of *groupthink* in review outcomes [20].

In any large project involving developers spread across a wide geographical area, adopting effective organization and management practices becomes crucial for ensuring timely deployment a product or service. It has been found that high performing project teams differed from low performing teams in terms of process management, relational development, and proactive technology use behaviors [6]. Observations from this study can be useful for managers while organizing their teams towards making the review process more effective.

In our study we have have found that developers tend to decrease close collaboration while increasing in connection with their peers. This can offer an interesting insight about the nature of knowledge workers in the software development context; developers seem to prefer engaging in many different problems rather than focusing on few.

3.4 Threats to Validity and Future Work

Threats to the construct validity of our results can arise from the way the *RIN* has been defined, as well as the metrics we have used as proxies for connection and clustering. We have used the standard protocol for affiliation networks while constructing the *RIN*; however, considering the directionality of edges can yield different results. The average degree and the clustering coefficients are widely used indicators of connection and clustering in a network and are unlikely to introduce notable threats. As we have accessed our data from a publicly available source, threats to internal validity are confined to the veracity of the data source. This study examines only one system; thus we do not claim our results to be generalizable and recognize the concomitant threat to external validity. To mitigate the threat to reliability, we make available the source code used to analyse the data[8].

In this paper, we have reported some initial observations around the evolutionary trends of developer interaction as they work on peer review of code. The observations allow us to conjecture about some of the underlying mechanisms that have given rise to the

[8]https://github.com/santonus/chromium_new/tree/master/Peer-Review-Analysis/

trends we notice. In our future work, we plan to examine whether and to what extent these mechanisms have causal influences on the observed trends, through controlled experiments and/or simulations.

4 RELATED WORK

Over the past few decades, peer review has come to be recognized as an important activity in software development [1, 10]. Review of software by a group of people is not only an essential part of software quality assurance [14], it also fosters sharing and dissimination of knowledge in the organization [5, 8, 16]. We broadly divide the state of the art in this area into i) the role of the peer review in the software development methodology of industrial as well as OSS software projects and ii) social aspects of an OSS project. The work by Rigby et al.on the open source Apache project [18] analyzed two review techniques, namely review followed by commit and commit followed by review. Subsequently more investigations towards various mechanisms of peer-reviews have been made [17, 19] on a large number of software projects. Researchers have revisited Linus laws in the context of peer reviews and investigated the impact of the skills possessed by participants during collaborative analysis and bug findings of the Mozilla software [21]. The work by Baccheli et al [4] reports the review mechanisms in a wide range of Microsoft products. An early work by Bird et al. [7] studied formation of communities among the participants in an OSS project. Social network analysis has been used in categorizing bugs in OSS projects [23]. The work by Yang et al. [22] proposed a peer review network to understand the importance of roles such as contributor, author, reviewer, committer etc., in the peer review process. Our current work focuses on the study of network characteristics among the OSS reviewers. Our previous study [9] in this area revealed that – contrary to the common belief [3, 21] – reviewers who collaborate more with others, take more time to close a review.

5 SUMMARY AND CONCLUSIONS

In this paper, we have studied how connection between developers and their clustering vary over time when they are working on peer review of code in a large open source software development ecosystem. Contrary to conventional wisdom, we find evidence that separation between developers remains nearly constant over time, while connection increases, and clustering and closeness decrease. These characteristics point to the distinct nature of the review activity, vis-a-vis other activities of the software development life cycle.

REFERENCES

[1] A.F. Ackerman, L.S. Buchwald, and F.H. Lewski. 1989. Software inspections: an effective verification process. *IEEE Software* 6 (1989), 31–36.

[2] Reka Albert and Albert-Laszlo Barabasi. 2001. Statistical mechanics of complex networks. *cond-mat/0106096* (June 2001). Reviews of Modern Physics 74, 47 (2002).

[3] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug?. In *Proc. 28th Intl. Conference on Software engineering*. 361–370.

[4] Alberto Bacchelli and Christian Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *Proc. Intl. Conference on Software Engineering*. 712–721.

[5] O. Baysal, O. Kononenko, R. Holmes, and M.W. Godfrey. 2013. The influence of non-technical factors on code review. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*. 122–131. https://doi.org/10.1109/WCRE.2013.6671287

[6] Catherine Beise, Traci Carte, Chelley Vician, and Laku Chidambaram. 2010. A Case Study of Project Management Practices in Virtual Settings: Lessons from Working in and Managing Virtual Teams. *ACM SIGMIS Database* 41 (2010), 75–97.

[7] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. 2008. Latent social structure in open source projects. In *Proc. FSE*. 24–35.

[8] Amiangshu Bosu, Jeffrey C. Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2017. Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft. *IEEE Transactions on Software Engineering* 43, 1 (2017), 56–75. https://doi.org/10.1109/tse.2016.2576451

[9] S. Datta, D. Bhatt, M. Jain, P. Sarkar, and S. Sarkar. 2015. The Importance of Being Isolated: An Empirical Study on Chromium Reviews. In *Intl. Symp. on Empirical Software Engg. and Measurement*. 1–4. https://doi.org/10.1109/ESEM.2015.7321215

[10] M.E. Fagan. 1986. Advances in software inspections. *IEEE Transactions on Software Engineering* 12 (1986), 744–751.

[11] Roger Guimer, Brian Uzzi, Jarrett Spiro, and Luis A Nunes Amaral. 2005. Team assembly mechanisms determine collaboration network structure and team performance. *Science* 308, 5722 (2005), 697–702.

[12] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, A. E. Camargo Cruz, Kenji Fujiwara, and Hajimu Iida. 2013. Who Does What During a Code Review? Datasets of OSS Peer Review Repositories. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 49–52. http://dl.acm.org/citation.cfm?id=2487085.2487096

[13] Ivar Jacobson, Grady Booch, and James Rumbaugh. 1999. *The Unified Software Development Process*. Addison-Wesley.

[14] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2014. The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects. In *Proc. 11th Working Conference on Mining Software Repositories*. 192–201.

[15] M. E. J Newman. 2003. The structure and function of complex networks. *cond-mat/0303516* (March 2003). SIAM Review 45, 167-256 (2003).

[16] Peter C. Rigby and Christian Bird. 2013. Convergent Contemporary Software Peer Review Practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 202–212.

[17] Peter C. Rigby, Daniel M. German, Laura Cowen, and Margaret-Anne Storey. 2014. Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 35 (2014), 33 pages.

[18] Peter C. Rigby, Daniel M. German, and Margaret-Anne Storey. 2008. Open Source Software Peer Review Practices: A Case Study of the Apache Server. In *Proc. 30th Intl. Conference on Software Engineering*. 541–550.

[19] Peter C. Rigby and Margaret-Anne Storey. 2011. Understanding Broadcast Based Peer Review on Open Source Software Projects. In *Proc. 33rd Intl. Conference on Software Engineering*. 541–550.

[20] Marlene E Turner and Anthony R Pratkanis. 1998. Twenty-Five Years of Groupthink Theory and Research: Lessons from the Evaluation of a Theory. *Organizational Behavior and Human Decision Processes* 73, 2 (Feb. 1998), 105–115. https://doi.org/10.1006/obhd.1998.2756

[21] Jing Wang, Patrick C.Shih, and John M.Carroll. 2015. Revisiting Linus's law: Benefits and challenges of open source software peer review. *International Journal of Human-Computer Studies* 77 (2015), 52–65.

[22] Xin YANG, N. YOSHIDA, R GAIKOVINA KULA, and H. Iida. 2016. Peer Review Social Network (PeRSoN) in Open Source Projects. *IEICE Trans. Inf. & Syst.* E99-D (2016), 661–670.

[23] M.S. Zanetti, I. Scholtes, C.J. Tessone, and F. Schweitzer. 2013. Categorizing bugs with social networks: a case study on four open source software communities. In *Proc. ICSE*. 1032–1041.