The Importance of being Isolated
An Empirical Study on Chromium Reviews

Subhajit Datta
Singapore University of
Technology and Design
Email: subhajit.datta@acm.org

Devarshi Bhatt
BITS-Pilani
Goa, India
Email: f2012097@goa.bits-pilani.ac.in

Manish Jain
Heritage Institute of Technology
Kolkata, India
Email: manishjain2792@gmail.com

Proshanta Sarkar
IBM Global Services
Kolkata, India
Email: proshant.cse@gmail.com

Santonu Sarkar
BITS-Pilani
Goa, India
Email: santonus@goa.bits-pilani.ac.in

***Abstract*—As large scale software development has become more collaborative, and software teams more globally distributed, several studies have explored how developer interaction influences software development outcomes. The emphasis so far has been largely on outcomes like defect count, the time to close modification requests etc. In the paper, we examine data from the Chromium project to understand how different aspects of developer discussion relate to the closure time of reviews. On the basis of analyzing reviews discussed by 2000+ developers, our results indicate that quicker closure of reviews owned by a developer relates to higher reception of information and insights from peers. However, we also find evidence that higher engagement in collaboration by a developer is associated with slower closure of the reviews she owns. Within the scope of our study, these results lead us to conclude that peer review of code may have a distinct dynamic that is facilitated by developers working in relative isolation.**

I. Introduction and Motivation

In *Cathedral and the Bazaar*, Raymond emphasized how Linux development enshrined a new way of software building, which was more interactional than instructional [1]. Canonizing his observations in the so-called *Linus's Law*, Raymond pointed out how higher developer attention seems to relate to quicker resolution of bugs. Although the veracity and validity of Linus' Law has been questioned [2], it does offer an epigrammatic insight into the benefits of engaging a large pool of developers and users in resolving defects in complex software systems. In the quarter century since the first release of Linux, distributed development of large scale software has become quite the norm. Today, open source as well as proprietary software development initiatives increasingly leverage global teams. Members of these teams are often separated by continents and cultures, and wide difference in time zones offer rare opportunities for synchronous communication. In such situations, peer review of code becomes a critical mechanism for delivering quality software, within project constraints. While many empirical studies [3], [4] have been reported over the years on factors influencing the number of defects in software systems, there have been relatively few studies on the outcome of code reviews. The increasing adoption of collaborative code review tools such Gerrit[1] and Rietveld[2] facilitates a better understanding of the review process.

With reference to Figure 1, the review life cycle usually goes through the following stages: a unit of code relating to a work item is raised for review and verified, then reviewed by one or more reviewers; these reviewers give their review comments which are responded to by the developers; through this co-commenting process a consensus is reached on the changes to the code that needs to be implemented; the code is accordingly changed and tested for residual or regression defects; then approved for merging with the main code base or abandoned; followed by closure of the review. Peer reviewing offers a great opportunity for developers to enhance their awareness about the development ecosystem [5]. It is in the interest of smooth project governance that reviews are closed quickly, and if they can not be closed due to external dependencies, they are marked as abandoned and parked aside.

Developer interaction in collaborative software development has many implications. While interaction among team members is usually encouraged in the hope of fostering better collaboration, some interactions may be helpful, while others can be distracting. Interaction in the software development context, such as co-commenting or co-editing work items easily lends itself to be studied through the analysis of interaction *networks*, whose vertices (nodes) are developers, and two developers are connected if there is an instance of some predefined interaction between them [6]. There is a body of literature on how developer interaction relates to outcomes in software development, and it is generally recognized that different aspects of interaction affect outcomes differently [7].

We recognize the following aspects of interaction in the peer review process: developers advising their peers by sharing specific information or insight - we call this *dissemination*; developers receiving advise from their peers - we call this *reception*; and developers working together with one another on closing the reviews - we call this *collaboration*. (The next section describes how we measure these parameters in this

[1]https://code.google.com/p/gerrit/
[2]https://code.google.com/p/rietveld/

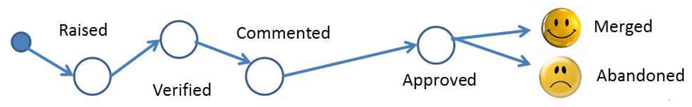

Fig. 1. Life-cycle of a review

study.) The collective aim of peer review is to complete each unit of review as effectively as possible, within the constraints of time. Thus disseminating and receiving relevant know-how, as well as collaborating with peers, stand to influence the outcome of the review process. To study these influences, we examine the following hypotheses in this paper:

- H1: *The reviews owned by a developer are closed faster if the developer advises a large number of her peers.*
- H2: *The reviews owned by a developer are closed faster if the developer receives advice from a large number of her peers.*
- H3: *The reviews owned by a developer are closed faster if the developer collaborates more with her peers.*

In the next section we describe the study setting and methodology, followed by a presentations of the results and threats to validity. A brief overview of related work is given next, followed by discussions and conclusions.

II. Study Setting and Methodology

Code review data from software projects has been made available in the public domain, as described in [8]. We took the dump of review data from the Chromium project[3] and parsed the data[4] into a MySQL database for the ease of querying. The entire data-set contained 826398 comments, made by 5865 developers on 209618 review records. For our study we selected a subset 2103 developers who own at least one review and have peer-reviewed at least one review owned by some other developer. We filtered the set of reviews owned by this subset of developers by removing those reviews with missing or absurd attributes (such as last modified date recorded being earlier than the created date) to 159625 reviews for our analysis, which is about 76% of the all the reviews in the data-set.

From the filtered data, we extracted a *review comment network* (RCN) whose vertices are developers and two developers are connected by an arc, such that there is a link starting from the developer who sends a comment and the link ends at the developer who receive the comment. We do not consider the weights of these arcs in this study. Senders and recipients of each comment is specified in the data-set, which enables us to the draw these arcs. To examine our hypotheses, we develop multiple linear regression models with variables as described below. There is a base model with only the dependent variable and the control variables (as explained below, details in column titled I in Table I), and a refined model with the independent variables additionally included (column titled II in Table I).

[3]http://www.chromium.org/

[4]http://sdlab.naist.jp/reviewmining/data/chromium-reviews-20121030.zip

As RCN is a directed network, the number of incoming edges, or in-degree of a vertex indicates the level of information received by that vertex, whereas the corresponding out-degree is a measure of the amount of information disseminated by the vertex. Thus, the *independent variables* are calculated as - **Dissemination:** A developer's out-degree in RCN; **Reception:** A developer's in-degree in RCN. **Collaboration:** A developer's clustering coefficient in RCN. If a vertex v in a network 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 k_v *choose* 2 or $k_v * (k_v - 1)/2$. If the *actual* number of such edges existing is N_v, then the clustering coefficient is, $C_v = \frac{2*N_v}{k_v(k_v-1)}$. The clustering coefficient of a vertex as defined above reflects on the extent to which the individual corresponding to the vertex collaborates with her peers [9]. The *control variables* for the models are calculated as - **Workload:** The number of reviews owned by a developer; **Interest:** The number of reviews commented upon by a developer. **Reach:** The number of comments by a developer on reviews that are not owned by the developer. **Influence:** The number of peer reviews conducted by the developer. **Inputs:** The median number of peer developers who have commented on each review owned by a developer. **Span:** Elapsed time between first and last comment made by a developer. **Delay:** Median elapsed time of all reviews owned by a developer for which the reviews have not been closed. The *dependent variable* **ClosureTime** for the models is taken as median of the elapsed time of the reviews owned by the developer. The elapsed time for each review is calculated as the time period in days between the review creation date and the date it is last modified. For the set of reviews we include in our analysis, the last modified date marks the closure of the review. The choice of our dependent variable is based on the assumption that time taken to close a review is a reliable proxy for the attention received by the review, and its calculation is supported by the information available in the data-set. We recognize that effectiveness of the review process can also be measured by other parameters such as review quality, not considered in this study.

As the dependent variable is continuous rather than a count, we chose multiple linear regression as the modeling paradigm over Poisson or negative binomial regression. Multiple linear regression rests on the assumptions of linearity, normality, and homoscedasticity of the residuals, and absence of multicollinearity between the independent variables. The residual properties were verified using histogram, P-P plot and scatter plot of the standardized residuals. The variables are transformed by taking square roots, whenever necessary for a better fit (the transformed variables are denoted by "sqrt" before their names in Table I). The correlations among the independent variables were low, the highest being between *Collaboration* and *Dissemination* (around 0.45). To further verify whether multicollinearity between variables artificially altered the significance of the overall regression and the regression coefficients' stability, we calculated the variance inflation factor (VIF) for each variable. For all the variables, the VIF was found to be below the recommended upper limit [10]. Based on the above discussion, it can be concluded that the assumptions of linear multiple regression hold within permissible limits in our case, and thus it is appropriate to use this modeling paradigm [10].

TABLE I. Results of regression for the effects on review closure time.(Superscripts '***', '**', '*', '·' denote $p \leq 0.0001$, $p \leq 0.001$, $p \leq 0.01$, $p \leq 0.05$, respectively)

	I	II
	Base model	*Refined model*
Intercept	118.0404^{***}	102.3478^{***}
	(11.6881)	(12.9848)
Control variables		
sqrt(Workload)	-4.4908	$6.1556^{\cdot}$
	(3.3314)	(3.6659)
sqrt(Interest)	-0.32	-1.0497
	(0.652)	(2.1495)
sqrt(Reach)	3.5146^{**}	3.7754^{**}
	(1.1434)	(1.2468)
sqrt(Influence)	-7.3375^{**}	-7.6015^{**}
	(2.5499)	(2.7586)
Inputs	$9.8196^{\cdot}$	9.8141
	(5.8720)	(6.5288)
Span	0.1068^{***}	0.1338^{***}
	(0.0182)	(0.0193)
Delay	0.7331^{***}	0.7162^{***}
	(0.0259)	(0.0256)
Independent variables		
sqrt(Reception)		-29.3159^{***}
		(4.4076)
sqrt(Dissemination)		0.5138
		(4.8821)
Collaboration		177.7117^{***}
		(31.8272)
N	2103	2103
R^2	0.301	0.32
df	2095	2092
F	101	131
p	< 0.001	< 0.001

III. Results and Threats to Validity

Table I presents results of the regression models; the numbers in parentheses give the standard errors for each of the coefficients. We see from the corresponding p-values in the lower portion of the table that both the base and refined models are statistically significant overall and the R^2 values (0.301 vis-a-vis 0.32) denote that addition of the independent variables over and above the control variables have increased the goodness-of-fit of the model. We also observe that there is statistically significant effect of *Reception* and *Collaboration* on the dependent variable, whereas the effect of *Dissemination* is not statistically significant. On the basis of the sign of the coefficients, we see evidence that higher reception relates to faster closure of reviews, whereas higher collaboration is associated with slower closure of reviews. Thus we find empirical evidence in favor of H2 and in opposition to H3; whereas H1 can not be validated with statistical significance in this study. We examine the implications of these findings in Section V section after outlining the threats to validity in the remainder of this section, and related work in the next section.

In this paper we report results from an observational study rather than a controlled experiment. Thus, in the statistical models developed, correlation does not imply causation. Also, from the values of the R^2, it is evident that the models are able to explain only about 30-32% of the variability of the data; thus there are some factors which we have not considered in our models. **Construct validity** relates to whether the variables are measured correctly. Our independent variables are measured using established network metrics. The measurement of control variables are also grounded in standard software engineering practices. However we recognize that there may be alternative ways to measure similar parameters [6] and the communication networks can be constructed differently [11]. However, these differences are unlikely to change the general direction of our conclusions. Our independent variables do not measure the quality or intensity of interactions between developers. In our future work, we plan to examine how these factors affect our results. **Internal validity** ensures that a study is free from systematic errors and biases. As the Chromium repository is our only source of data, common issues affecting internal validity such as mortality and maturation do not arise. A key influence on internal validity is the extent to which the development team used the review platform; we assume the usage was widespread. **External validity** is concerned with the generalizability of results. Our study covers the review cycle of a single product. As demonstrated in existing literature, useful insights can be drawn from observational studies on single subjects [12], [7]. Thus even as we do not claim our results to be generalizable as yet, we believe our results throw interesting light on the study objective. **Reliability** of a study is established when the results are reproducible. From the discussion in earlier sections, it is evident that there is limited human intervention in the extraction and processing of data. Once the data is extracted, we use standard statistical techniques for analysis. Assuming our data source is accessible, our results can be easily reproduced.

IV. Related Work

In recent times, there is an increasing interest in various aspects of the code review process. Rigby, German, and Storey investigate two peer review techniques - review-then-commit, and commit-then-review - that are used in the successful Apache server project, and arrive at a set of general observations on how peer review is actually conducted in the project [13]. Rigby and Storey report findings from an empirical study that examines how developers identify code changes they can review as well as interaction profiles of stakeholders [14]. Bacchelli and Bird report a study of review comments across a wide range of teams at Microsoft; they find that in addition to finding defects, reviews foster enhanced team awareness, and knowledge transfer [15]. Baysal et al. study the code review process of WebKit - a large, open source project and conclude that non technical factors involving organizational and personal dimensions significantly impact code review outcomes [16]. In addition to other results, the authors of [5] conclude that "that conducting peer review increases the number of distinct files a developer knows about by 66% to 150% depending on the project". McIntosh et al. study the relationship between software quality vis-a-vis code review coverage and code review participation in Qt, VTK, and ITK projects and conclude that poorly reviewed code negatively impacts software quality in large systems [17]. Beller et al. found that bug-fixing tasks lead to fewer changes and tasks with more changed files and higher code churn have more changes [18]. Rigby et al. study the review policies of 25 OSS projects and the archival records of six large, mature, and successful OSS projects to conclude that OSS peer review is "drastically" different from traditional inspection [19]. The studies listed above - among others - establish that peer review processes have many subtleties. Our current work complements existing results by examining how review closure time is related to the various ways in which developers interact in the peer review process.

V. Discussions and Conclusion

With reference to the results presented in Section III, we find evidence that for developers, higher *Reception* relates to quicker closure of reviews. Developers who receive more information are likely to have higher familiarity with the project ecosystem [5]. And it is expected that this will relate to quicker closure of reviews owned by such developers. But surprisingly, we find higher collaboration to be associated with developer reviews taking more time to be closed. This *confronts* the conventional wisdom that collaboration is beneficial for many collective enterprise [20] [21] including some software development activities, such as bug resolution [22].

Our evidence points to the nature of the review activity being inherently different from other software development tasks. It is likely that reviewers need to operate in a zone relatively free from the pressures of conformity that other collective activities bring with it. Perhaps peer review of code has something in common with the review of scientific manuscripts. It is expected - if not imperative - that a competent scientific review reflects the individual perspective of the reviewer, and not a conjoint opinion arrived through consensus. Similarly, code review may call for the focused attention of developers, which too much collaboration can impair.

In today's milieu of growing emphasis on collaboration, it is often overlooked that while collaboration brings many essential signals, it can also carry much inessential noise. Depending on the nature of the activity being collaborated around, this noise can be detrimental at times. *Our results indicate that for peer review of code by developers, sharing one's knowledge with peers rather than closely collaborating with them can be more helpful.*

In an interview in 2002, Dijkstra had provocatively remarked "...thanks to the greatly improved possibility of communication, we overrate its importance. Even stronger, we underrate the *importance of isolation*. ...Thanks to my isolation, I would do things differently than people subjected to the standard pressures of conformity. I was a free man."[5](italics added). In resonance with Djikstra's perspective, our results can inform how peer review teams are assembled and managed, with a deeper understanding of the importance of isolation.

References

[1] E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, 2001.

[2] A. Meneely and L. Williams, "Strengthening the empirical analysis of the relationship between linus' law and software security," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:10.

[3] A. G. Koru and H. Liu, "Building defect prediction models in practice," *IEEE Softw.*, vol. 22, no. 6, p. 2329, Nov. 2005.

[4] T. Zimmermann and N. Nagappan, "Predicting defects with program dependencies," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, Oct. 2009, pp. 435 –438.

[5] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 202–212.

[6] M. Cataldo and J. D. Herbsleb, "Communication networks in geographically distributed software development," in *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, ser. CSCW '08. New York, NY, USA: ACM, 2008, p. 579588.

[7] T. Wolf, A. Schroter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, p. 111.

[8] K. Hamasaki, R. G. Kula, N. Yoshida, A. E. C. Cruz, K. Fujiwara, and H. Iida, "Who does what during a code review? datasets of oss peer review repositories," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 49–52.

[9] R. Albert and A. Barabasi, "Statistical mechanics of complex networks," *cond-mat/0106096*, Jun. 2001, reviews of Modern Physics 74, 47 (2002).

[10] B. Tabachnick and L. Fidell, *Using Multivariate Statistics*. Boston: Pearson Education, 2007.

[11] K. Ehrlich and M. Cataldo, "All-for-one and one-for-all?: a multi-level analysis of communication patterns and individual performance in geographically distributed software development," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 945–954.

[12] T. Wolf, T. Nguyen, and D. Damian, "Does distance still matter?" *Softw. Process*, vol. 13, no. 6, pp. 493–510, 2008.

[13] P. C. Rigby, D. M. German, and M.-A. Storey, "Open source software peer review practices: A case study of the apache server," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 541–550.

[14] P. C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 541–550.

[15] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 712–721.

[16] O. Baysal, O. Kononenko, R. Holmes, and M. Godfrey, "The influence of non-technical factors on code review," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*, Oct 2013, pp. 122–131.

[17] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 192–201.

[18] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 202–211.

[19] P. C. Rigby, D. M. German, L. Cowen, and M.-A. Storey, "Peer review on open-source software projects: Parameters, statistical models, and theory," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 4, pp. 35:1–35:33, Sep. 2014.

[20] K. Ehrlich, G. Valetto, and M. Helander, "Seeing inside: Using social network analysis to understand patterns of collaboration and coordination in global software teams," in *Proceedings of the International Conference on Global Software Engineering*, ser. ICGSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 297–298.

[21] R. Guimer, B. Uzzi, J. Spiro, and L. A. N. Amaral, "Team assembly mechanisms determine collaboration network structure and team performance," *Science (New York, N.Y.)*, vol. 308, no. 5722, pp. 697–702, Apr. 2005, PMID: 15860629.

[22] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*. Shanghai, China: ACM, 2006, pp. 361–370.

[5]http://cacm.acm.org/magazines/2010/8/96632-an-interview-with-edsger-w-dijkstra/fulltext