Clustering, Separation, and Connection: A Tale of Three Characteristics

1st Subhajit Datta
Singapore Management University, Singapore
subhajit.datta@acm.org

2nd Aniruddha Mysore
PES University, Bengaluru, India
aniruddha.mysore@gmail.com

3rd Haziqshah Wira
Singapore Institute of Technology, Singapore
haziqshahwira@gmail.com

4th Santonu Sarkar
BITS Pilani, Goa, India
https://orcid.org/0000-0001-9470-7012

***Abstract*—In large and complex software development ecosystems, developers collaborate in multiple dimensions. How characteristics of such collaboration vary over time can offer insights about the dynamics of large scale software development. In this paper we have constructed networks of developers who co-comment on and co-change units of code, and analysed the patterns of variation of clustering, connection, and separation between such developers over time, using development data from a large open source system. Though clustering, connection, and separation essentially represent facets of the same collaboration activities, we found them exhibiting distinct time-varying characteristics. The variation of clustering indicate that developers congregate more closely towards the beginning of the project when the architecture of the system is not yet stable and they need to reach out to one another to fulfil their collective responsibilities. However, separation between developers shows a quick rise and then a saturation around a particular value. Developer connection continues rise throughout the observation period. Using time-series analysis, our results allow us to derive insights on the evolutionary trends in large scale software development and can inform the tuning of tools and process towards effective team assembly and governance.**

***Index Terms*—Software development, clustering, separation, connection, time series analysis**

I. Introduction

Over the past few decades large scale software development has become an increasingly interactive enterprise. Teams of developers – often temporally and spatially separated – build and maintain software-intensive systems through close interaction. As a software project traverses the phases of the software development life cycle, characteristics of developer interaction also vary over time. Analysing the patterns of such variation can offer valuable insights on some of the key dynamics of software development in the large.

The inherent invisibility and unvisualizability of software [1] along with software essentially being ”peopleware” [2] leads to the typical complexities of large and distributed software development. Thus developer interaction is *multi-faceted* for any non-trivial system. To capture the essence of such interaction it is inadequate to examine a single aspect. Like the wise men in Saxe's parable, we will be unable to appreciate the whole if we look at a part in isolation [3]. In this paper we consider three characteristics of developer interaction as reflected by their *connection*, *clustering*, and *separation*, and observe how each characteristic varies over time for two distinct contexts of developer interaction.

As in the study of other collaborative human enterprises, interaction in software development ecosystems can be examined using the network paradigm: vertices represent developers and two developers are linked if they both co-participate in some unit of development activity. In such a network, the extent of connection between developers reflects the ease with which information can flow between them, while clustering is an indication of the level of their collaboration, and separation denotes the degree to which developers are at a distance from one another. Established metrics from network science can capture these characteristics [4].

The software development life cycle is inherently time dependent. Projects move across the phases of *inception*, *elaboration*, *construction*, and *transition* as the system being developed matures through collective development effort [5]. We posit that a deeper understanding of developer interaction is only possible when we consider the time varying characteristic of such interaction. Accordingly, in this paper, we examine the following research question: **How does the variation of connection, clustering, and separation over time in developer interaction networks reflect the dynamics of large scale software development?** To address this question, we use data from a large real-world software development ecosystem to construct time-sliced interaction networks from instances of developers co-commenting on and co-changing units of work. We next compute network metrics reflecting on the levels of developer connection, clustering, and separation in these networks. Time-series analysis of each of these metrics allows us to observe the components of their variations over time. These observations lead to insights on how the variation of the metrics manifest some of the underlying dynamics of large scale software development. A deeper examination the key drivers of team assembly and governance in software development can be informed by such insights. The relevance of our research, both in terms of approach and results, arises from the inherently dynamic nature of the software development life cycle. Thus examining the time-varying characteristics of developer interaction is more appropriate than considering their static parameters, towards a deeper understanding of how developers interact across different stages of maturity of the software system being developed.

II. Study Setting and Methodology

A. Choice of dataset

To address our research question, we analysed data from the OpenStack development ecosystem as collected, curated, and published by Gonzalez-Barahona et al. [6]. OpenStack is a popular open source cloud computing platform that is available for deployment as infrastructure-as-a-service (IaaS)[1]. Given the widening application of cloud computing, the OpenStack ecosystem is an active arena of development. The choice of this study setting was additionally influenced by the fact that the OpenStack development data shared by Gonzalez-Barahona et al. include developer interactions across four distinct yet related development activities – source code change, problem ticket resolution, code reviews, and developer communication via mailing lists [6]. The OpenStack dataset consists of four databases: source code (135 repositories; 183,413 commits; 3,836 authors), problem tickets (55,044 tickets; 635,895 updates; 7,582 identities), mailing list (15 lists; 88,842 messages; 4,399 posters), and reviews (119,989 code reviews; 3,533 submitters). The richness of this data offers an opportunity to extract insights at a fine level of granularity from multiple activities of the development process.

B. Pre-processing and filtering data

To identify the most active individuals in the development ecosystem, we looked for developers who have participated in *all* four activities: resolving problem tickets, reviewing code, committing code changes, and posting messages in the mailing lists. For the 879 such developers who were identified we constructed time-sliced *co-comment networks* and *co-change networks* using the following construction protocols.

C. Construction of networks

For each of the following network types vertices represent developers and two vertices are connected by an edge (undirected link) if the developers corresponding to the vertices at either end of the edge have both participated in some common unit of development activity. The networks are independent of one another and constructed cumulatively over 50 time-steps between 2010 and 2014. Each time-step is of three weeks. This duration was decided after examining how the development activities were distributed over time and observing the typical iteration lengths in large scale open source software systems.

1) Co-Comment Network: In a co-comment network, two developers are connected by an edge if both of them have co-commented on a problem ticket.

2) Co-Change Network: In a co-change network, two developers are connected by an edge if both of them have co-changed a unit of code.

[1]https://en.wikipedia.org/wiki/OpenStack

D. Measuring characteristics of developer interaction

As mentioned earlier, we use established network metrics to measure connection, clustering, and separation in the context of our study.

1) Connection: The average degree as given by the average number of edges per vertex, measures the level of *connection* in a network [7]. If a network has V vertices and E edges, average degree is given by 2 * E / V.

2) Clustering: The clustering coefficient (C_v) of a vertex v in a network is defined in the following way: If v has a degree of k_v, implying that 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 we assume that the *actual* number of such edges existing is N_v, then C_v = 2 * N_v / $k_v * (k_v - 1)$. So the clustering coefficient of a vertex is the ratio of the actual number of edges existing between its neighbours and maximum number of such edges that can exist [8]. For an entire network, the clustering coefficient (sometimes called the *global* clustering coefficient) is the average clustering coefficient of all its vertices. Evidently, this reflects the extent of *clustering* among the vertices of the network.

3) Separation: The ease with which two vertices i and j in a network can communicate with one another is influenced by the length of the *shortest* path l_{ij} between them. The average of l_{ij} taken over all pairs of vertices is the average separation of the network [7]. We take this to denote the degree of *separation* between developers in this study.

E. Time series analysis

To address our research question, we need to examine the nuances of variation of connection, clustering, and separation over time. Towards that end, we utilize *time-series analysis* to examine how these metrics vary for each of the time-sliced co-comment and co-change networks [9].

Datasets involving time varying metrics exhibit certain well-documented patterns such as autocorrelation, global trend over time, cyclicality, seasonality and non-linear breaks. In this study, we will focus specifically on two kinds of components – the non-periodic component of *trend* and the periodic component of *seasonality*. Cyclicality is a common time-series pattern, that is very closely related to seasonality. Cyclical variations typically occur over a longer period of time and are non-uniform. We choose seasonality since it is easier to separate data into trend and seasonality using decomposition than it is to to measure cyclicality in isolation. Our choice of these particular time-series components is influenced by the underlying nature of large scale software development: the increasing complexity of the system being developed, and iterative development cycles leading to incremental releases that aim to tame that complexity.

III. Results and Discussion

Observations: We first observe the time variations of connection, clustering, and separation for the co-comment and co-change networks *before* decomposing them into time-series components in Figure 1. We notice that connection increases

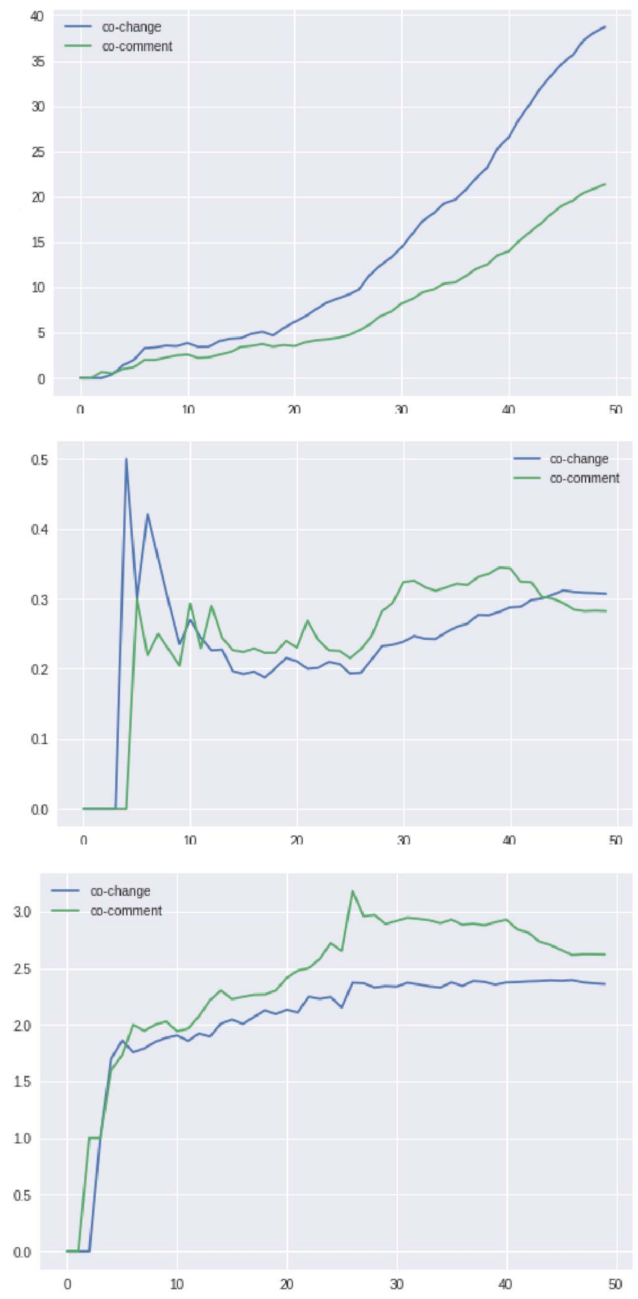

Fig. 1: Variation of connection, clustering, and separation over time (first, second, third from top respectively)

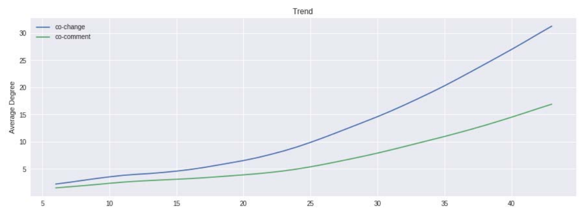

Fig. 2: Trend component of connection: Co-Change and Co-Comment networks

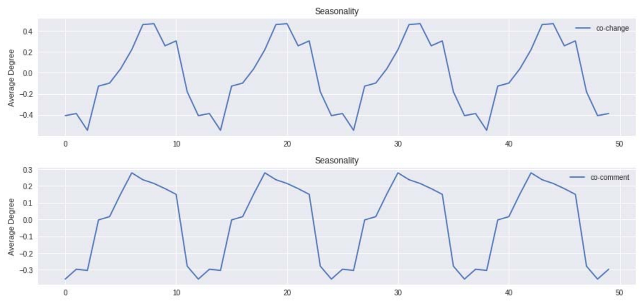

Fig. 3: Seasonality component of connection: Co-Change and Co-Comment networks

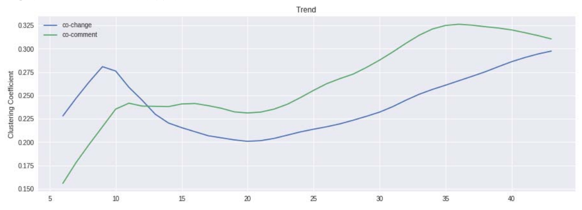

Fig. 4: Trend component of clustering: Co-Change and Co-Comment networks

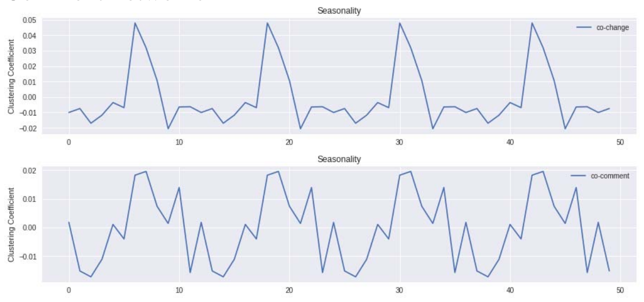

Fig. 5: Seasonality component of clustering: Co-Change and Co-Comment networks

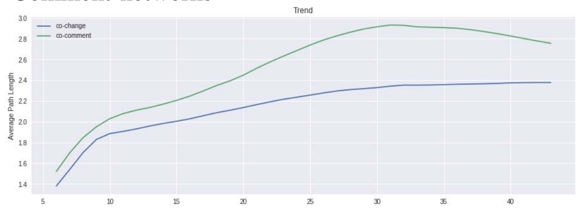

Fig. 6: Trend component of separation: Co-Change and Co-Comment networks

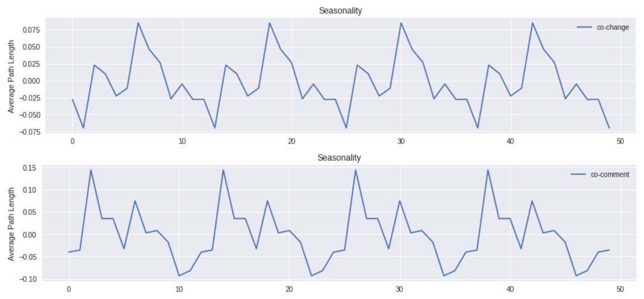

Fig. 7: Seasonality component of separation: Co-Change and Co-Comment networks

gently for both networks till about the 25th time-step and then commences a steep ascent toward their peak values. On the other hand, the variation of clustering is non-monotonic for both networks, with rapid variations in the initial time-steps settling down in the latter time-steps. Separation for both networks rises sharply during the initial time-steps, and then stabilizes. Co-commenting and co-changing represent two distinct development activities; the former is more of "talk" while the latter is essentially "work" [10]. Even then, we observe an overall congruence in how connection, clustering, and separation vary over time for both the co-comment and co-change networks. This is an indication of a commonality underlying diverse development activities. Let us now examine the components of these characteristics as identified by time-series analysis.

Network Parameter	Co-change network	Co-comment network
	Clustering	
Trend	Initial increase followed by U-shape	Alternate period of increase and stability
Seasonality	Single peaked, greater magnitude.	Multi-peaked, lesser magnitude
	Separation	
Trend	Only initial increase	Similar to co-change
Seasonality	Weak signal	10x magnitude, multi-peaked signal
	Connection	
Trend	Exponential increase	Lesser rate of increase
Seasonality	Weak signal	Weak signal

TABLE I: Trends: Clustering, separation, connection

Figures 2, 3, 4, 5, 6, and 7 show the trend and seasonality components of connection, clustering, and separation for the co-comment and co-change networks.

With reference to Table I we note how the trend and seasonality components of each metric compare across the two networks. Even as clustering, separation, and connection are all manifestations of interconnections between developers – via either co-commenting or co-change of development artefacts – we notice congruence as well as differences between their variations. What is the dynamic behind such patterns?

Implications: For all the metrics for both types of networks, the seasonality components show a clear periodicity. This is most likely a reflection of the iterative and incremental nature of development; the repetitive patterns in seasonality denote each iteration. On the other hand, the trend component of connection is monotonically increasing; for clustering, it rises at the beginning, then falls, and then rises again; while for separation, it increases with a flattening gradient. To understand why these time varying characteristics are thus oriented, let us revisit the progression of a software project. At the inception of a software development life cycle, developers cluster together to discus and arrive at key architectural and design decisions. As the system matures, each developer's interactions are predominantly confined to his/her peers working on the same module. This cycle repeats over successive iterations of development, each leading to an incremental release. Even as the level of clustering undergoes this variation, the extent of familiarity between members of the developer pool increases as developers reach out to one another towards fulfilment of their collective responsibilities. This is reflected in continual increase in the level of connection among developers along with an eventual stabilization of separation. These trends underscore how clustering, connection, and separation have a nuanced relationship as developers collaborate to build large and complex systems. Project governance processes as well as tools and technologies need to be tuned to facilitate each of these aspects towards effective team outcomes.

Key message: While investigating our research question, we find evidence that even as clustering, separation, and connection reflect aspects of the same underlying mechanisms of developer interaction, the trend and seasonality components of their time-varying characteristics differ amongst one another in both the co-comment and co-change networks. These differences point to the multi-dimensional nature of developer interaction as a software system is collectively developed.

IV. Related Work

In large scale software development ecosystems, developers collaborate to discover complex bugs, performance issues, or violation of architectural principles which are sometimes difficult to detect using tools. With the availability of software development process data, there have been a growing interest in the quantitative assessment of the impact of such collaboration. Rigby et al. have examined various aspects in a code review collaboration processes including interaction profile of stakeholders over a significant period of time [11]–[14]. Tourani et al. [15] analysed the same OpenStack dataset we have used to establish that experienced developers are involved in less risky code commits. The study by Pabon et al. [16] examined email interactions to discover relationships between socio-technical aspects and personality traits of the developers, by analysing several open source project datasets similar to ours. Presenting a contrarian perspective, Caulo et al. have shown that code reviews do not have any significant impact on the knowledge transfer among developers using GitHub pull requests over time [17]. However, Bacchelli and Bird reported in a study of diverse teams at Microsoft that over and above defect detection, code reviews do engender heightened team awareness, and transfer of knowledge [18]. Baysal et al. have examined WebKit – a large, open source project and infer that non technical factors such organizational and personal matters notably influence development outcomes [19]. McIntosh et al. have studied how software quality is associated with code review coverage and participation in Qt, VTK, and ITK projects; they conclude that gaps in the collaborative review process have negative impact on software quality [20]. Paixao et al. [21] examined developer comments in the context of architectural changes. Contrary to previous beliefs, studies such as [22] have shown that enhanced collaboration among reviewers may not necessarily lead to diminution of productivity. Datta et al. have analysed interaction among reviewers over time and observed a distinction between the manifestations of collective and individual efforts in the code review process [23].

V. Threats to Validity

Threats to **construct validity** relates to measurement errors in the variables of interest that may invalidate conclusions from a study. In this study, connection, separation, and clustering are calculated using established network metrics. Thus threats to construct validity are not present in this study. **Internal validity** is concerned with systematic errors and biases that may challenge a study's findings. As we have used a dataset whose collection and curation procedures have been published in a peer-reviewed paper [6], we expect threats to internal validity to have already been addressed. A study's **external validity** is associated with the generalizability of its results. In this paper, we report results from studying a single system. Given the diversity of the system's development context, we believe the results are insightful. However, they can not be claimed to be generalizable as yet. A study's **reliability** arises from the reproducibility of its results. With access to the dataset, our results can be fully reproduced.

In our **future work**, we plan to address the threat to external validity by extending our analysis to include other datasets. As discussed earlier, results from time-series analysis reported in this paper has pointed to distinct patterns in the variation of clustering, separation, and connection. We have sought to offer intuitive explanations of such trends in the general context of large scale software development. We plan to develop and validate generative models that can explain such trends in our future work.

VI. Summary and Conclusion

Collaboration characteristics between developers in large software development ecosystems vary over the time-line of a project life cycle. In this paper, we have analysed the variation of three such characteristics – connection, separation, and clustering – over time, for two different aspects of developer collaboration in the OpenStack development ecosystem. Observing the trend and seasonality components of the variation allows us to infer a certain commonality of purpose and execution in diverse development activities, as well as distinct pattern of variation. These results can inform developers on the benefits of conjoint work as well as individual effort at different junctures of the development life cycle and guide the assembly and governance of co-located and distributed teams.

VII. Acknowledgement

The authors thank Mr Moses Ban for his contribution in processing the dataset analysed in this paper.

References

[1] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley, 1995.

[2] T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*. Dorset House Pub. Co., 1987.

[3] J. G. Saxe, "The blind men and the elephant," http://bygosh.com/Features/092001/blindmen.htm, 1850.

[4] M. E. J. Newman, "The structure and function of complex networks," *cond-mat/0303516*, Mar. 2003, SIAM Review 45, 167-256 (2003). [Online]. Available: http://arxiv.org/abs/cond-mat/0303516

[5] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley, 1999.

[6] J. M. Gonzalez-Barahona, G. Robles, and D. Izquierdo-Cortazar, "The metricsgrimoire database collection," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15, 2015, pp. 478–481.

[7] A. L. Barabasi, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Vicsek, "Evolution of the social network of scientific collaborations," *cond-mat/0104162*, Apr. 2001, physica A 311, (3-4) (2002), pp. 590-614. [Online]. Available: http://arxiv.org/abs/cond-mat/0104162

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

[9] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications: With R Examples*, 4th ed., ser. Springer Texts in Statistics. Springer International Publishing, 2017. [Online]. Available: https://www.springer.com/gp/book/9783319524511

[10] S. Datta, R. Sindhgatta, and B. Sengupta, "Talk versus work: characteristics of developer collaboration on the jazz platform," in *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, ser. OOPSLA '12, 2012, pp. 655–668.

[11] 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.

[12] 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.

[13] 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, 2013, pp. 202–212.

[14] 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.

[15] P. Tourani and B. Adams, "The impact of human discussions on just-in-time quality assurance: An empirical study on openstack and eclipse," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 189–200.

[16] Paruma-Pabon *et al.*, "Finding relationships between socio-technical aspects and personality traits by mining developer e-mails," in *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering*, ser. CHASE '16, 2016, pp. 8–14.

[17] M. Caulo, B. Lin, G. Bavota, G. Scanniello, and M. Lanza, "Knowledge transfer in modern code review," in *ICPC2020*, IEEE/ACM, Ed., 2020.

[18] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the International Conference on Software Engineering*, 2013, pp. 712–721.

[19] 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*, 2013, pp. 122–131.

[20] 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*, 2014, pp. 192–201.

[21] M. Paixao, J. Krinke, D. Han, C. Ragkhitwetsagul, and M. Harman, "The impact of code review on architectural changes," *IEEE Transactions on Software Engineering*, 2019.

[22] S. Datta, D. Bhatt, M. Jain, P. Sarkar, and S. Sarkar, "The Importance of Being Isolated: An Empirical Study on Chromium Reviews," in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Oct. 2015, pp. 1–4.

[23] S. Datta and P. Sarkar, "Evolutionary trends in the collaborative review process of a large software system," in *Proceedings of the 12th Innovations on Software Engineering Conference*, 2019.