Anti-patterns in Modern Code Review: Symptoms and Prevalence

Moataz Chouchen*, Ali Ouni*, Raula Gaikovina Kula†, Dong Wang†,
Patanamon Thongtanunam‡, Mohamed Wiem Mkaouer§, Kenichi Matsumoto†
*ETS Montreal, University of Quebec, QC, Canada
†Nara Institute of Science and Technology, Nara, Japan
‡University of Melbourne, Melbourne, Australia
§Rochester Institute of Technology, Rochester, NY, USA

***Abstract*—Modern code review (MCR) is now broadly adopted as an established and effective software quality assurance practice, with an increasing number of open-source as well as commercial software projects identifying code review as a crucial practice. During the MCR process, developers review, provide constructive feedback, and/or critique each others' patches before a code change is merged into the codebase. Nevertheless, code review is basically a human task that involves technical, personal and social aspects. Existing literature hint the existence of poor reviewing practices *i.e.*, *anti-patterns*, that may contribute to a tense reviewing culture, degradation of software quality, slow down integration, and may affect the overall sustainability of the project. To better understand these practices, we present in this paper the concept of Modern Code Review Anti-patterns (MCRA) and take a first step to define a catalog that enumerates common poor code review practices. In detail we explore and characterize MCRA symptoms, causes, and impacts. We also conduct a series of preliminary experiments to investigate the prevalence and co-occurrences of such anti-patterns on a random sample of 100 code reviews from various OpenStack projects.**

***Index Terms*—Modern code review, review anti-pattern**

I. Introduction

Modern Code Review (MCR) is an established and broadly adopted software engineering practice in both commercial and open-source software (OSS) projects [1], [2]. Code review is defined as the process of reviewing other developers code to ensure software quality, and find potential problems in their code changes before they are merged with the codebase. MCR derives from the formal and disciplined process of software inspection, which requires synchronous face-to-face meetings among developers to make a checklist-based code inspection and interactive discussion [3]. Conversely, MCR provides practitioners with a convenient environment to read and discuss code changes and makes this activity lightweight, less formal and asynchronous through a tool specialized support for geographically distributed code review [1], [4]. There is an increasing number of available MCR platforms including Gerrit, ReviewBoard, and Phabricator.

The MCR process is most effective when developers follow best practices, such as efficient, collaborative and timely review discussions and code updates to improve the code quality, enhance knowledge transfer, increase team awareness and share code ownership. Such practices help reducing conflicts in the team while ensuring that the code meets common quality standards before it is merged into the code base [5]. In practice, it is often challenging to follow these standards due to the nature of code review being basically a human task involving technical, personal and social aspects [1], [5]–[11]. Hence, some poor code review practices can be observed and manifest in the form of *anti-patterns*, *i.e.*, common but ineffective practices to a recurring problem that should be avoided. In recent years, researchers and practitioners attempted to defined catalogs of MCR anti-patterns, which become a major problem that hinders software quality, maintainability and sustainability, if not properly addressed [12]–[14].

In practice, such code review anti-patterns can manifest in different forms such as divergent reviews, low reviewer participation and responsiveness, toxic conversations, etc. [7], [11], [15], [16]. For example, since reviewer opinions may differ, patches can receive both positive and negative scores leading to conflicts in the peer review process, due to the disagreement about whether a developer's contribution should be accepted. Hence, if reviews with divergent scores are not carefully resolved, they may contribute to a tense reviewing culture and may slow down integration and lead to detrimental effects on contributors' continuing participation in the community affecting the sustainability of the project [11], [16].

In this paper, we further study this phenomenon to overcome these problems. With the aim of providing practitioners with a code review quality-oriented dashboard, we compiled a catalog for MCR anti-patterns. It contains 5 common anti-patterns related to different aspects of the MCR management and process, explaining their symptoms, causes and potential impacts on the development team and project. To gain more understanding, we further conducted a set of preliminary experiments to investigate the prevalence and occurences of such anti-patterns on a random set of 100 code reviews from various OpenStack projects that use Gerrit as a MCR platform. Overall, our results indicate that MCR anti-patterns are indeed prevalent in the studied projects. Practitioners should be aware of these anti-patterns and consider detecting and preventing them using dedicated techniques.

II. Related Work

Factors that impact the effectiveness of the MCR process. Various studies focused on the MCR process in both

open-source and industry. Bosu et al. [17] studied code review in Microsoft by investigating various factors that make code reviews useful to developers based on review comments. Jiang et al. [18] studied the factors that impact the decision on the acceptance of Linux patches. They conclude that patches written by experienced developers are easily accepted and that the number of invited reviewers have an impact of review time. Ram et al. [19] suggest that the change description, size, and coherency with the project coding style impacts the likelihood of the code change being reviewed. Baysal et al. [20] showed that various technical and non technical factors affect the review process including the complexity of the patch, the patch writer experience and the reviewer previous history and workload. Recently, Hirao et al. [16] studied patches with divergent review scores and found that 15%–37% of patches that receive multiple review scores suffer from divergent scores. Furthermore, [8] showed that a low level of agreement is more likely to take a longer reviewing time and discussion length. Thongtanunam et al. [21] investigated patches that do not attract reviewers, not discussed, and receive slow initial feedback. They found that the length of the patch description plays an important role in the likelihood of receiving poor reviewer participation or discussion.

Socio-technical aspects in MCR. Huang et al. [11] studied issues related to potential conflicts in code review. They indicate that conflicts generally have detrimental effects on contributors' continuing participation in the community, while constructive suggestions increase retaining the contributors. Bosu et al. [22] showed that core developers receive quicker first feedback on their review request, complete the review process in shorter time, and are more likely to have their code changes accepted into the code base. Later, Bosu et al. [23] studied the impact of interpersonal relations on the patch review in MCR. They found that the patch author is one of the important factors for peer reviewers to decide to review a patch or not. In addition, Steinmacher et al. [24] gave evidence of the existence of several social barriers faced by newcomers in the code review process. Baysal et al. [25] studied the patch life cycle in code review process in Firefox and found that patches submitted by casual contributors are disproportionately more likely to be abandoned compared to core contributors. Later, Mcintosh et al. [26] suggested that coverage, reviewers participation and expertise play high impacts on the code quality. Recently, Uchoa et al. [27] found that long discussions and review disagreements increase design degradation. Ebert et al. [6] found that missing rationale and lack of familiarity with the code are the major reasons for confusions in code review. Raman et al. [15] studied toxic conversations and unhealthy interactions in open source projects indicating their potential impacts to demotivate and burn out developers, creating challenges for sustaining open source.

III. MCRA: A CATALOG OF MCR ANTI-PATTERNS

Despite bringing several benefits, MCR can be problematic and challenging especially when it does not follow good practices [5], [28]. A catalog of MCR anti-patterns is of crucial importance to increase the practitioners awareness towards such practices. In particular, we identify a list of common anti-patterns based on the existent literature, and provide an illustrative example.

A. Description of MCR Anti-patterns

1) Confused reviewers (CR): Confusion in code review refers to the inability or the uncertainty of the reviewers to understand the reason(s) for the code change or any related aspects to the patch [6]. There are several reasons behind confusion in code review such as missing rationale, lack of experience with the source code, and complex patches [10].

2) Divergent reviewers (DR): A code patch under review can suffer from divergent reviewers when reviewers cannot agree on the final evaluation by providing conflicting reviews and scores [16]. DR can lead to several problems in the review process including developer abandonment [11], poor team performance [29] and slow integration processes [8].

3) Low review participation (LRP): This anti-pattern is defined as the low involvement of reviewers when reviewing a given code patch. Patches with low number of reviewers can be more defect-prone [7]. Rigby et al. [30] showed that the level of review participation is the most influential factor in the code review efficiency.

4) Shallow review (SR): The SR anti-pattern happens when the review comments are not relevant for the patch author and/or focus on insignificant details namely, code nitpicking, variables name, spaces, etc, instead of addressing quality issues in the patch [5].

5) Toxic review (TR): Developers and reviewers can have high stress levels due to several socio-technical factors []. Toxic conversations and unhealthy interactions may demotivate and burn out developers and reviewers, creating challenges for sustaining open source.

To get more details about the salient aspects of each anti-pattern type, Table I describes the symptoms and potential impacts/consequences on the software product, the review process, and the team.

B. Illustrative examples

To show the salient aspects of MCR anti-patterns, Figure 1 depicts an example of a code review[1] taken from the `Opendev` project, using the Gerrit code review platform. Several anti-patterns can be found in this example including confusion in reviewers comments. For example, we can see that the reviewer "`Sean McGinnis`" is not clear about the rationale of the patch through his comment "*Do you have a link to somewhere that says this is deprecated? I tried to find one, but I don't see anything stating they are deprecating this. Nothing in the source either[...]*" (cf. discussion box Ⓑ in Figure 1). Moreover, this code review suffers from low review participation since the patch was updated on May 11, 2018 and the first reviewers comment was on May 29, 2018. Furthermore, we observe from the review scores and

[1]https://review.opendev.org/#/c/567926/

TABLE I: MCR anti-patterns and their associated symptoms and consequences.

MCR anti-pattern	Symptoms	Potential Consequences
Confused reviews (CR)	• Reviewers ask question(s) about the rationale or the solution approach of the patch. • Reviewers express incertitude in review comments.	• *Process:* Confusion decreases the efficiency and the effectiveness of the review [6], [10]. • *Artifact*: The patch may still have poor quality after the review as reviewers may not fully understand the patch [6]. • *People:* Reviewers may feel frustrated and may express negative sentiments due to the confusion [6].
Divergent reviews (DR)	• The review decisions do not reach consensus. • The review scores are diverged. • Reviewers post conflict review comments to each others.	• *Process:* The divergence can slow down the integration process [16] • *Artifact*: It is correlated with negative development outcomes (i.e., patches without changing) [16] • *People:* The divergence increases contributors' likelihood of leaving the communities and lead to the poor team performance [11], [29]
Low review participation (LRP)	• The patch does not have other developers as a reviewer. • The patch receive few/short comments. • The patch does not receive prompt/timely feedback from reviewers.	• *Process:* The lack of review participation has a negative impact on review efficiency and effectiveness [30]. • *Artifact*: Patches with low number of reviewers can be more defect-prone [7] • *People:* The inefficient reviewer feedback could make the patch author forget the change. [5]
Shallow review (SR)	• The patch receive a superficial or shallow comments despite the complexity or size of the patch. • The review comments mainly focus on the visual representation (*e.g.*, code styling) or minor issue(s). • The review comments are unclear or a concern was raised with a clear explanation. • The absence of inline comments despite the complexity of the patch.	• *Process:* Focusing on small and irrelevant issues would waste time for no benefit [5]. • *Artifact*: Unknown. There is potential gap in literature on its impact. • *People:* Small problems (*i.e.*, a lot of style comments) would make the author annoyed [5].
Toxic review (TR)	• The patch has a controversial discussion that does not relate/focus to criticizing the code. • The patch receives a comment with a negative sentiment (*e.g.*, harsh, rage expressions).	• *Process:* The reviews with negative sentiments took longer time to get accepted [31]. • *Artifact*: Harmful sentiments could erode the benefits of suggested changes [31]. • *People:* Sentiments influence the quality of relationship between two persons [32].

discussions that there are divergent review scores (cf. box Ⓐ in Figure 1), in which the reviewer `Jay Bryant` provided **+2** score (*i.e.*, accept), whereas the reviewer `Sean McGinnis` provided **-1** score (*i.e.*, reject). Clearly, all these problems resulted into heated discussions between reviewers and the patch author and resulted in *abandoning* the patch as shown in the last comment of the author `Eric Harney` saying : "*Sure. I mean, it's not. But ... yeah, let's go with the "it doesn't matter" plan.*". Hence, from the example we can observe the importance to identifying such anti-patterns and prevent them as early as possible during the code review process.

IV. Preliminary experiments

While our long-term agenda is much broader, we conducted a preliminary study to investigate the phenomenon of anti-patterns in MCR. We first conducted a manual inspection to detect the existence of anti-patterns in a sample of code reviews. Thereafter, we design our experiments to address two main research questions on the frequency of each anti-pattern type (RQ1), and the prevalence of such anti-patterns in practice (RQ2) to gain more insights on this phenomenon.

A. RQ1: How frequent are MCR anti-patterns?

Context selection. To investigate the frequency of each MCR anti-pattern in practice, we considered the OpenStack project that adopts a review process based on the *Gerrit* review system. OpenStack is a large open source software ecosystem (*i.e.*, OpenStack attracts more than 100,000 contributors spread more than 600 repositories [33]), where many well-known organizations and companies collaboratively develop a platform for cloud computing. From the latest available online OpenStack datasets [34], [35], we randomly selected a sample of 100 code reviews to be manually inspected to identify the possible existence of anti-patterns. The random sample covers across all repositories and covered reviews from November 2011 to July 2019. This dataset has been used in several similar studies, especially for those that have motivated our anti-patterns [6], [10], [16], [34]–[36]. We also provide our replication package[2] for future replications and extensions of our study.

Analysis Method. To detect anti-patterns in the studied sample, each code review has been manually inspected by three authors individually. The manual inspection of each code review consists of reading through (1) the author/reviewers discussion threads, and (2) the source code change diff in Gerrit, in order to identify potential symptoms of MCR anti-patterns. In a preliminary iteration, the authors conducted an open discussion using a sample of 10 code reviews to discuss whether the defined anti-patterns match with the definition and symptoms. Thereafter, considering the workload required for the manual inspection (100 code reviews $\times$ 5 anti-pattern types, *i.e.*, 500 inspections), we divided the authors into two three-person groups, so that each group perform 250 inspections. Each inspection took on average from 5 to 25 minutes, depending on (1) the length of the review

[2]https://github.com/moatazchouchen/MCRA

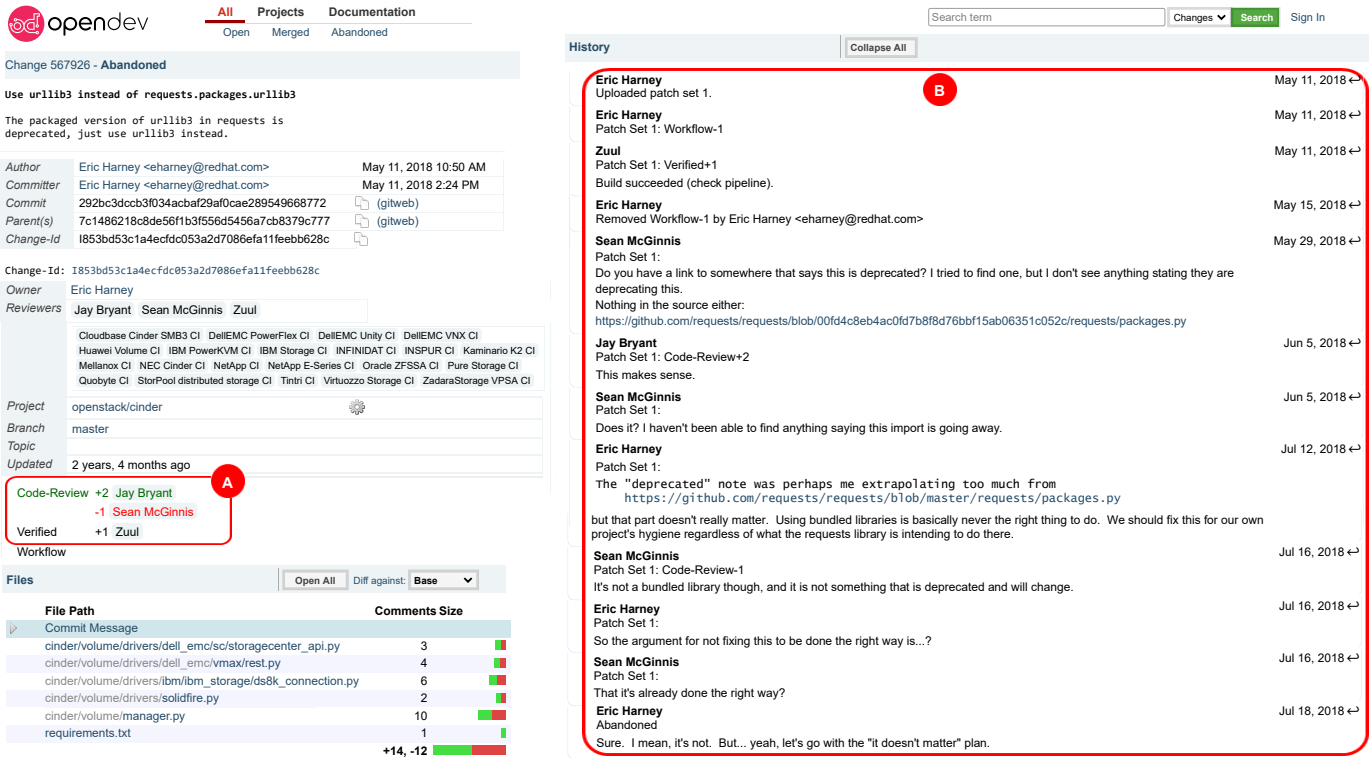

Fig. 1: Example of MCR anti-patterns from the OpenStack project, code change ID #567926 [1].

TABLE II: The detection results of MCR anti-patterns.

Anti-pattern	# instances	Kappa	Agreement
Confused Reviews (CR)	21	0.93	Perfect
Divergent Reviews (DR)	20	1.0	Perfect
Low Review Participation (LRP)	32	1.0	Perfect
Shallow Review (SR)	14	0.81	Substantial
Toxic Review (TR)	5	0.65	Substantial

discussions and (2) the size of the code change. To validate the consistency between the participants inspection, we measured the inter-rater agreement using Fleiss's Kappa coefficient κ [37]. Fleiss's Kappa coefficient κ is interpreted as: *Poor agreement* if $\kappa < 0$; *Slight agreement*, if $0.01 \leq \kappa \leq 0.2$; *Fair agreement*, if $0.21 \leq \kappa \leq 0.40$; *Moderate agreement*, if $0.41 \leq \kappa \leq 0.60$, *Substantial agreement*, if $0.61 \leq \kappa \leq 0.80$, and *Almost perfect agreement*, if $0.81 \leq \kappa \leq 1.00$. Based on the encouraging Kappa scores (i.e., near perfect for CR, DR, LRP, and substantial for SR, TR), two three-person groups coded the remaining samples.

Results. Table II shows the frequencies of anti-patterns and the obtained Fleiss's Kappa coefficient. the low review participation (LRP) anti-pattern is the most frequent affecting 32% of analyzed code reviews. The divergent review (DR) and confused review (CR) anti-patterns manifest in 20% and 21% of the studied code reviews, respectively. The shallow review (SR) is detected in 14% of the examples, and finally the toxic review (TR) turns out to be the lowest frequent one with 5%. Moreover, we achieved *perfect agreement* for 3 out the five anti-patterns, CR, DR and LRP, with 0.93, 1, and 1, respectively. We also achieved a *substantial agreement* for the two remaining anti-patterns, SR and TR, with a Kappa score of 0.81 and 0.65, respectively. Having a lower agreement level in SR and TR could be explained by the need for a degree of subjectivity to detect them. Therefore, it is challenging to manually detect code review anti-patterns which motivates the need of automated support tools and deeper understanding of the main roots behind these anti-patterns.

TABLE III: The prevalence of MCR anti-patterns.

Category	Count
Code reviews affected by one anti-pattern	67
Code reviews affected by two anti-patterns	21
Code reviews affected by three anti-patterns	4

B. RQ2: How prevalent are MCR anti-patterns?

To gain more insights from the detected anti-pattern instances, we further analyze their prevalence in the dataset.

Analysis Method. Our analysis consists of counting the number of anti-patterns that exist in each studied code review in RQ1 regardless of the specific anti-pattern types.

Results. The obtained results are summarized in Table III. We observe that 67% of the studied code reviews contain at least one instance MCR anti-pattern. Moreover, we can see that 21% of the studied code reviews have at least two anti-pattern instances, and found that 4% contain three or more anti-patterns. These results indicate that MCR anti-patterns are indeed highly prevalent, thus practitioners should be aware of them and consider detecting and preventing them using dedicated techniques.

V. Conclusion and Future Work

In this paper, we identified a list of five common MCR anti-patterns, their symptoms and potential impacts on the quality of the software product, the process and the team. To showcase the prevalence of these anti-patterns, we conducted a study on a random sample of 100 code reviews extracted from Openstack project. Preliminary results show that these anti-patterns are indeed prevalent in MCR affecting 67% of code reviews. While we do not claim that the provided catalogue is exhaustive, further investigations should be conducted.

As part of our future work, we plan to study the phenomenon of MCR anti-patterns in more depth by surveying developers in order to provide an exhaustive list of MCR anti-patterns. Moreover, we plan to design automated techniques to detect these anti-patterns. We also plan to study the sensitivity of MCR anti-patterns detection in different scenarios mainly within-project detection and cross-project detection. Finally, we plan to design and integrate dedicated bots in MCR tools that help developers to avoid such anti-patterns.

Acknowledgments. This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) discovery grant RGPIN-2018-05960, JSPS KAKENHI Grant Numbers 18H04094, 20K19774, and 20H05706. P. Thongtanunam was partially supported by the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE210101091).

References

[1] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *35th International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721.

[2] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Working Conf. on Mining Software Repositories*, 2014, pp. 202–211.

[3] M. Fagan, "Design and code inspections to reduce errors in program development," in *Software pioneers*. Springer, 2002, pp. 575–607.

[4] V. Kovalenko, N. Tintarev, E. Pasynkov, C. Bird, and A. Bacchelli, "Does reviewer recommendation help developers?" *IEEE Transactions on Software Engineering*, 2018.

[5] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, and J. Czerwonka, "Code reviewing in the trenches: Challenges and best practices," *IEEE Software*, vol. 35, no. 4, pp. 34–42, 2017.

[6] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion in code reviews: Reasons, impacts, and coping strategies," in *Int. Conference on Software Analysis, Evolution and Reengineering*, 2019, pp. 49–60.

[7] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Investigating code review practices in defective files: An empirical study of the qt system," in *W. Conf. on Mining Soft. Repositories*, 2015, pp. 168–179.

[8] T. Hirao, A. Ihara, Y. Ueda, P. Phannachitta, and K.-i. Matsumoto, "The impact of a low level of agreement among reviewers in a code review process," in *Int. Conference on Open Source Systems*, 2016, pp. 97–110.

[9] H. S. Qiu, A. Nolte, A. Brown, A. Serebrenik, and B. Vasilescu, "Going farther together: The impact of social capital on sustained participation in open source," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 688–699.

[10] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion detection in code reviews," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 549–553.

[11] W. Huang, T. Lu, H. Zhu, G. Li, and N. Gu, "Effectiveness of conflict management strategies in peer review process of online collaboration projects," in *ACM Conf. on Computer-Supported Cooperative Work & Social Computing*, 2016, pp. 717–728.

[12] E. Dietrich, "Manual code review anti-patterns - https://dzone.com/articles/manual-code-review-anti-patterns," in *DZone, Agile Zone*, 2017.

[13] T. Gee, "Five code review antipatterns - https://blogs.oracle.com/javamagazine/five-code-review-antipatterns," in *ORACLE Java Magazine*, 2020.

[14] T. Knierim, "Code review antipatterns. - https://thomasknierim.com/code-review-antipatterns/," 2018.

[15] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu, "Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions," in *International Conference on Software Engineering, New Ideas and Emerging Results (ICSE-NIER)*, 2020.

[16] T. Hirao, S. McIntosh, A. Ihara, and K. Matsumoto, "Code reviews with divergent review scores: An empirical study of the openstack and qt communities," *IEEE Transactions on Software Engineering*, 2020.

[17] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 146–156.

[18] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast? case study on the linux kernel," in *Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 101–110.

[19] A. Ram, A. A. Sawant, M. Castelluccio, and A. Bacchelli, "What makes a code change easier to review: an empirical investigation on code change reviewability," in *Joint Meeting on European Soft. Eng. Conf. and Symp. on the Foundations of Software Eng.*, 2018, pp. 201–212.

[20] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "Investigating technical and non-technical factors influencing modern code review," *Empirical Software Engineering*, vol. 21, no. 3, pp. 932–959, 2016.

[21] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Review participation in modern code review," *Empirical Software Engineering*, vol. 22, no. 2, pp. 768–817, 2017.

[22] A. Bosu and J. C. Carver, "Impact of developer reputation on code review outcomes in oss projects: An empirical investigation," in *Int. Symp. on Empirical Software Eng. and Measurement*, 2014, pp. 1–10.

[23] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, and C. Chockley, "Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 56–75, 2016.

[24] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in *18th ACM conference on Computer supported cooperative work & social computing*, 2015, pp. 1379–1392.

[25] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey, "The secret life of patches: A firefox case study," in *19th Working Conference on Reverse Engineering*, 2012, pp. 447–455.

[26] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Eng.*, vol. 21, no. 5, pp. 2146–2189, 2016.

[27] A. Uchôa, C. Barbosa, W. Oizumi, P. Blenílio, R. Lima, A. Garcia, and C. Bezerra, "How does modern code review impact software design degradation? an in-depth empirical study," *36th ICSME*, pp. 1–12, 2020.

[28] M. Greiler, C. Bird, M.-A. Storey, L. MacLeod, and J. Czerwonka, "Code reviewing in the trenches: Understanding challenges, best practices and tool needs," *IEEE Software*, pp. 34–42, 2016.

[29] A. Filippova and H. Cho, "The effects and antecedents of conflict in free and open source software development," in *ACM Conf. on Computer-Supported Cooperative Work & Social Computing*, 2016, pp. 705–716.

[30] 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 TOSEM*, vol. 23, no. 4, pp. 1–33, 2014.

[31] I. E. Asri, N. Kerzazi, G. Uddin, F. Khomh, and M. Janati Idrissi, "An empirical study of sentiments in code reviews," *Information and Software Technology*, vol. 114, pp. 37 – 54, 2019.

[32] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "Senticr: A customized sentiment analysis tool for code review interactions," in *International Conference on Automated Software Engineering*, 2017, pp. 106–111.

[33] Y. Zhang, M. Zhou, A. Mockus, and Z. Jin, "Companies' participation in oss development - an empirical study of openstack," *IEEE Transactions on Software Engineering*, 2019.

[34] P. Thongtanunam and A. E. Hassan, "Review dynamics and their impact on software quality," *IEEE Transactions on Software Engineering*, 2020.

[35] X. Yang, R. G. Kula, N. Yoshida, and H. Iida, "Mining the modern code review repositories: A dataset of people, process and product," in *Int. Conference on Mining Software Repositories*, 2016, pp. 460–463.

[36] 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 *MSR*, 2013, pp. 49–52.

[37] J. L. Fleiss, "Measuring nominal scale agreement among many raters," *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.