Code Review DAO (CRDAO)

Whitepaper - Draft Version 0.1

December, 2021

By

Wulf Kaal[1]

Abstract

The paper examines the existing shortcomings in the open-source code review process and how decentralized autonomous organizations can help optimize the code review process and quality. Optimized code reviews and code review processes, in turn, can help create open source software standards. The author evaluates the issuance of decentralized code reviews that are verified through a decentralized autonomous organization, the code review DAO (CRDAO). Key distinguishing features of the CRDAO include code review price discovery, increased speed of code reviews, community governance, lower prices for code reviews, full transparency, and, at a later stage, insurance for code reviews. Over time, the CRDAO code reviews can be combined with an underwriting service, through a code review underwriting DAO, where the CRDAO revenue from code reviews guarantees the consumer in certain circumstances that CRDAO code reviews are backed by assets.

***Key Words**:* Code Audit, Decentralized Autonomous Organization, Insurance, Underwriting, Finance, Token Models, Cryptocurrencies, Feedback Effects, Emerging Technology, Tokens, Blockchain, Distributed Ledger Technology

***JEL Categories**:* K20, K23, K32, L43, L5, O31, O32

Table of Contents

Background

The existing code review process is afflicted by many challenges. These challenges can be distilled into two separate categories, both of which make code review a timely and very expensive process.:

1. Human-code issues: Issues between the code being reviewed and the reviewer
2. Human-human issues: Issues between reviewers or between reviewers and authors

Code review is a process that is intended to ensure software quality. Usually, it consists of developers, other than the author of given piece of code, manually checking code or changing the code before they are merged into the main code repository.[2] Ideally, the process of code review finds defects or improvement opportunities without the software execution and before the product delivery, thereby reducing costs of future fixes.[3]

Modern code review (MCR) can be traced back to software "inspections."[4] Inspections consisted of formal meetings where participants would prepare ahead of time. Unlike inspections, many modern code review processes are asynchronous and often support geographically distributed reviewers.[5] The adoption of agile methods and distributed software development has resulted in less formal code reviews where the inefficiencies of inspections are replaced with a flexible, tool based, and frequent process known as modern code review.[6]

Code reviews are recognized as a vital practice to assure software quality. Many open-source projects such as Android, QT, and Eclipse, include code review as part of their software development process.[7] Moreover, many large technology companies like Microsoft, Oracle, and Samsung, all adopt code review for their software development.[8] Code reviews provide many benefits, for example: they attempt to find bugs in the code, help improve the quality of the code on productions, find better ways to implement the change, spread knowledge about the project, and create awareness of the changes in the code base.[9]

Despite the benefits and widespread adoption within the technology industry, code reviews can incur great cost and slow down the overall code development process.[10] Consequently, modern code review processes are expensive. Developers spend a significant amount of time reviewing

the changes of others – an average of 6 hours per week.[11] Not only is this a significant amount of time, it also requires an opportunity cost on project development as developers are forced to switch away from their current work.[12] Therefore, it is important to pinpoint the issues that make modern code review a timely and expensive process in efforts to understand solutions to these issues.

Survey of developers on the motivation for code reviews:[13]

Motivations for Code Reviews	
Survey respondents were asked to pick and rank their top 5 reasons for doing code reviews. The below items ranked first are valued higher than the following ranks.	
Reason for Code Reviewing	Overall Rank
Code improvement	1
Find defects	2
Increase knowledge transfer	3
Find alternative solutions	4
Improve the development process	5
Avoid breaking builds	6
Build team awareness	7
Lead to shared code ownership	8
Team assessment	9

Examples of modern code review processes:

Example 1 of a modern code review process:[14]

Fig. 1 The code review process

Example 2 of a modern code review process:[15]

Fig. 1. An overview of the tasks of modern code review.

Example 3 of a modern code review process:[16]

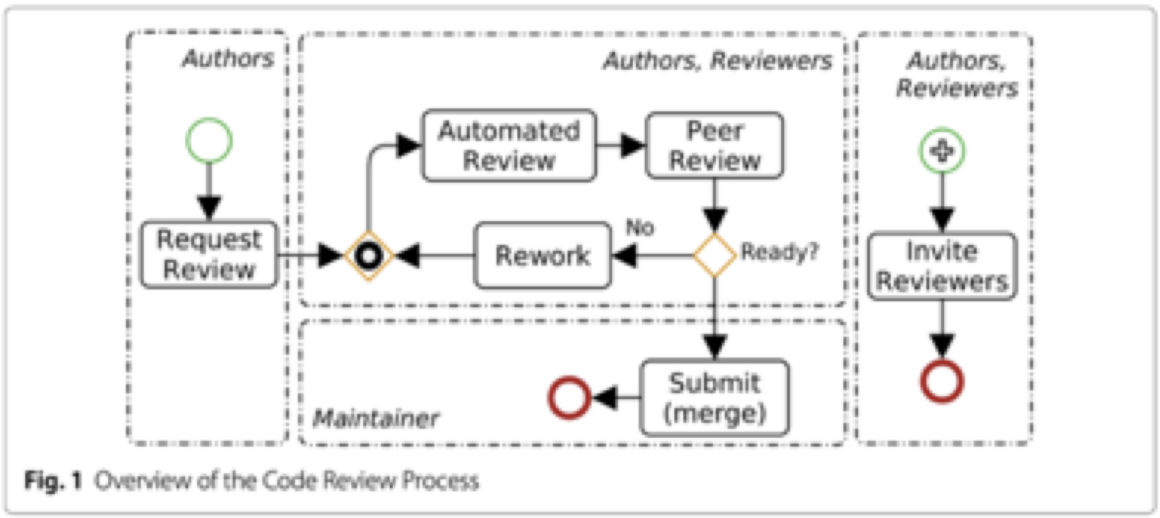

Fig. 1 Overview of the Code Review Process

Human–Code Issues

The first category of issues in modern code review identified in the literature are human-code issues, i.e., challenges that may arise between the code that is being reviewed and the reviewer. Two most common human-code issues are:

1. Patch size
2. Code confusion

Patch Size

One of the most notable challenges that reviewers face is the sheer size of the code review. Many commentators have noted that the effectiveness of code review decreases as the size of the patch increases. Therefore, the benefit of attaining a code review is negatively correlated with the size of the code under review. In particular, the larger the amount of files in a single review, the lower the rate of beneficial feedback from the reviewers.[17]

Several studies have examined the correlation of patch size with code review quality. For example, one study analyzed the effectiveness of code review as the patch size (measured in lines of code) increased. More specifically, this study measured the comment density of code reviews of different sizes and found that comment density by reviewer decreases as patches increased.[18] Accordingly, the quality and amount of contribution is affected by patch size.[19] And, patch size increases the time required to provide significant contributions.[20] Furthermore, patch size negatively affects all outcomes of code review effectiveness.[21]

The negative correlation between effectiveness and patch size is mirrored by other studies as well. For example, another study found that code reviewers with longer review queues tend to get overwhelmed and are more likely to submit poor-quality code evaluations.[22] Therefore, modern code review effectiveness is undermined by heavier review loads.

Attempts to address the effects of patch size on code reviews include distributing the work load among a broader set of reviewers and provide better transparency on developer review queues.[23] However, despite the proposed solutions, many commentators recognize that patch size remains to be an issue for modern code review affecting the quality, speed, and effectiveness of the process.

Code Confusion

The next challenge that reviewers face when reviewing code is the confusion reviewers experience when encountering unfamiliar code. Confusion about new code includes situations in which a reviewer is uncertain about something or unable to understand something which can impact the development process.[24] Nevertheless, confusion in code reviews is an inevitable starting point for reducing the cost of code reviews.[25] The figure below shows that confusion among reviewers during the code review process is very common.

Example of frequency of confusion for developers and reviewers:[26]

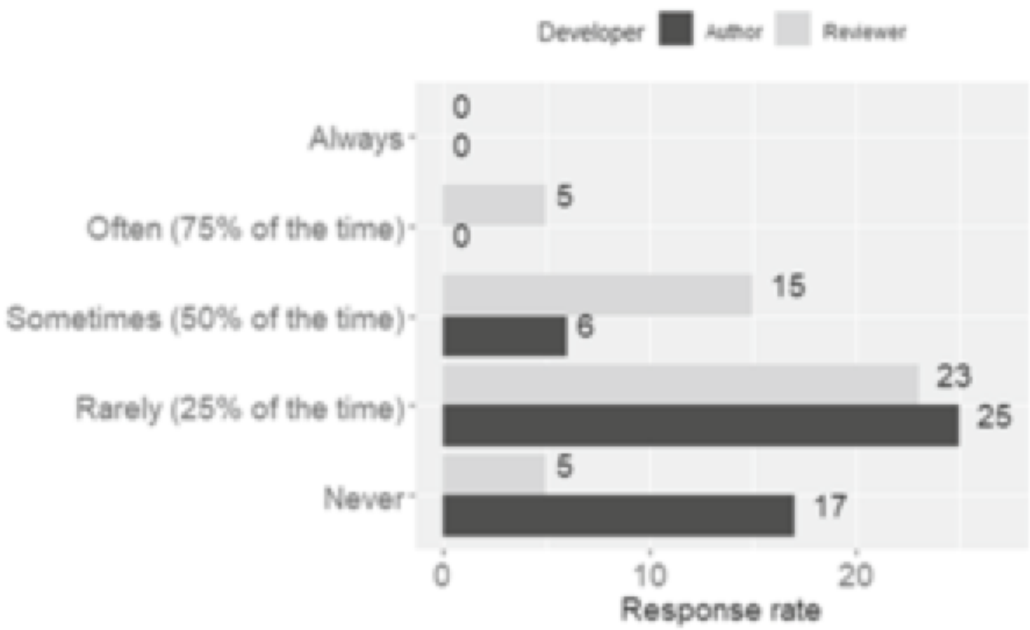

Fig. 3 Frequency of confusion for developers and reviewers

Organization of code is one of the most common reasons for why developers encounter confusion during the code review.[27] *Organization of work* includes unclear commit message (e.g., "*when the description of the pull request is not clear*"), the status of the change (e.g., "*I'm unsure about the status of your parallel move changes. Is this one ready to be reviewed?*") or the change addressing multiple issues (e.g., "*change does more than one thing*").[28] Another popular reason for confusion is the absence of the change rationale (e.g., "*I do not fully understand why the code is being modified.*").[29]

The notion that developers struggle with confusion during the code review process is recurrent throughout the literature. Past literature reviews concerning this topic have cited several studies that display multiple factors of confusion for developers. For example, one cited reason of confusion for developers is the failure of "understanding the motivation and purpose" of a proposed code change.[30] Gaining familiarity with the code is a technical challenge faced by reviewers.[31]

The impacts of code confusion on modern code review are substantial. For example, one study found 7 negative impacts on code review which resulted from code confusion.[32] The number

one mentioned impact was the review takes longer than it should.[33] In addition, the study also found that more confusion leads to an increase in the number of messages exchanged during the discussion, therefore reviews take longer.[34] If there is confusion between the reviewer and the code, the code review will take longer than it should. Another interesting impact of code confusion mentioned in the literature is the phenomena of "blind approval."[35] Some studies have mentioned that blind approval may occur when a reviewer hopes their co-worker knows what they're doing and blindly approves a change.[36] Blind approval can result in further bugs and delay production.[37]

Human-Human Issues

The second category of issues discussed in the literature can be characterized as human-human issues, i.e., issues between the author and reviewers or between the reviewers themselves. These include:

1. Developer participation
2. Lack of feedback
3. Organizational structure

Developer Participation

Research has found that the lack of participation in code review has a negative impact on the quality of the software. Many commentators have discovered that code review quality is affected by community involvement. For example, reviews accompanied with a good interaction among developers discussing bug fixes and patches are less prone to bugs themselves.[38] Furthermore, the number of comments made by patch owners has a significant negative impact on "bug-proneness" in review.[39] This finding is somewhat expected, but can be challenging to address because, the code review processes is subject to sensitivities resulting from the egos of the individuals involved.[40] Therefore, failing to maximize developer participation and communication may negatively affect the code review process. This ultimately results in inefficiencies in the review process creating unnecessary costs on software development.

Example of a developer survey for participation rates in code review:[41]

Code Review Participation		
During the previous week how often did you..	How often do you act as a code reviewer?	How often do you author code reviews?
At least once per day	39%	17%
A couple times during the week	36%	48%
Once during the week	12%	21%
None	13%	14%

Lack of Feedback

Another challenge that is mentioned in the literature is the difficulty in obtaining effective feedback. One study surveyed developers about the challenges they face during code review. The study found the number one challenge for developers was "receiving feedback in a timely manner."[42] Moreover, even when developers finally receive feedback on their code change, it is often not insightful or helpful. For example, the study also found that some reviewers focus on "insignificant details rather than larger issues."[43] Therefore, modern code review processes are negatively impacted by the amount of time it takes to receive constructive feedback among the developers.

Example of a developer survey for challenges faced during code review:[44]

Challenges faced during code reviewing	
Survey respondents were asked to pick and rank their top 5 challenges during doing code reviews. The below items ranked first are valued higher than the following ranks.	
Challenge during Code Reviewing	Overall Rank
Receiving feedback in a timely manner	1
Review size	2
Managing time constraints	3
Understanding the code's purpose	4
Understanding the motivations for the change	5
Obtaining insightful feedback	6
Bikeshedding (disputing minor issues while more serious ones are overlooked)	7
Understanding how the change was implemented	8
Maintaining code quality	9
Reaching consensus	10
Finding relevant documentation	11
Managing multiple communication channels	12
Identifying who to talk to	13

Figure 1: Overview of selected responses from the code review survey.

Organization

Organizational structure of code can influence software quality. For example, organization metrics including the number of developers working on a component, organizational distance between developers, and organizational code ownership, are "better predictors of defect-proneness than traditional metrics like churn, complexity, coverage, dependencies, and pre-release bug measures."[45] These findings support "Conway's Law" which states that a software system's design will resemble the structure of the organization that develops it.[46]

Different organizations, such as Apple and Google, have a statistically significant difference between how each organization approves their own patches. For example, one study found a statistically significant difference in the time it takes for Apple to accept their own patches versus the time it takes for Apple to accept Google patches.[47] From this, the authors argue that "Apple treats their own patches differently from external patches."[48] Therefore, organizational identity and structure may have impacts on the time and effectiveness of the code review process.

Example of response time for code review for different organizations:[49]

TABLE III
RESPONSE TIME (IN MINUTES) FOR ORGANIZATIONS.

Reviewer → Writer	Accepted		Rejected	
	Median	Mean	Median	Mean
Apple → Apple	25	392	60	482
Apple → Google	73	617	283	964
Google → Google	45	484	102	737
Google → Apple	42	483	80	543

Flawed Code Review Process

It is a common conception that code reviews increase the functionality and quality of code. Code reviews also clarify the intended functionality of the code which minimizes the risk that malfunctioning or suboptimal code gets released and causes damages.

Despite these benefits that derive from the code review process, the existing code review industry is subject to significant cost, inefficiencies, barriers to entry, and lack of insurance solutions, among other lacking client services for code reviews. In 2021, the code review industry is dominated by several key players who can charge cartel and monopoly-like prices. Despite these high prices, the code review process is subject to significant flaws.

First and foremost, the selection process for the reviewer is an ongoing challenge for the existing code review process in legacy code review firms. The more hierarchical the code review process is, the lower is the quality of the reviewed code. It makes intuitive sense that the more developers review a given set of code, the higher the quality of the code may turn out to be. However, in the legacy review process the first reviewer within the hierarchical structure of the code review process often gets the highest priority and is often merely followed with minor upgrades by follow-on reviewers. The collective of reviewers is also not incentivized to find flaws in the code to optimize code as a work product of the collective. Rather, it is often seen as the work product of the initial reviewer with minor input from follow on reviewers. Moreover, the more people review the code with comments that ask for clarification, the more likely it becomes that the code becomes simpler and clearer, which in turn typically increases code quality. However, that is no possible in hierarchical review processes. The hierarchical approach to code reviews undermines long-term participation with opinion from the edges of the reviewer spectrum because those reviewers either have no access or are in no position to help review the code. In other words, the more hierarchical the code review process and the more barriers to entry, the lower the quality of code.

Lack of Crowd Controls

In the existing legacy code review process, the views of the reviewer and the intent of the code author are often at odds with each other without any crowd control. Because the code reviewer may wish to impose their own logic on the code author, the code author may be required to rewrite code over and over even though the core functionality of the code is sound and dangerous issues were controlled for. This can be highly time consuming and inefficient. It also calls the overall role of the code review process into question. Instead, a code review should focus on functionality of the code and on keeping mistaken, badly constructed, and dangerous code out.

Timing

Depending on the setting of the code review, code reviews in legacy systems can last weeks and months. This can get exacerbated by market conditions in the digital asset market. These significant delays can impact development and may require complete rewriting of contracts because the underlying protocol may have upgraded core libraries during the code review.

Cartels

Despite the fact that most code review firms help clients who hope to decentralize different parts of the industry and capitalize on efficiencies created by decentralizing legacy systems, the code review industry is mainly dominated by a cartel. The top five code audit firms form a cartel.

The cartel of the top 5 audit firms also creates high barriers to entry for new players to enter into the code review market.

Given these downsides in the existing code review market, it is a bit ironic that one of the strongest forms of exploitation and centralized economies of scale are being created in a market that is, from the outset, supposed to help support the decentralization of disparate industries.

Overpricing

As a result of the cartelization of the industry, most code reviews are significantly overpriced. Clients pretty much pay any price to get the stamp of approval from one of the top 5 audit firms.

No Controls

The cartel power also undermines attempts by other industry players to create internal or external controls on the quality of code reviews. As a result, the public has no or very weak control over the quality of code review services it receives. Customers cannot afford to look for better priced code reviews and are forced into the cartel pricing to obtain market acceptance of their products. In turn, the cartelization of the market undermines any form of downward price pressure.

Because of the power of the cartel over the overall market and the process of the code review and its outputs, the quality of code review is often suboptimal.

Moreover, there is little or no recourse for clients in cases in which the code proved to be flawed even after functionality and quality review.

No Insurance

Despite the significant flaws in the code review process and the associated often flawed results of code reviews, the existing code review market does not allow for any form of insurance products that are associated with other products in other markets.

Even if code review and audit firms were to engage in a more thorough insurance underwriting and guarantee process for their clients, they would still be subject to the flawed and centralized legacy insurance market. In the current insurance market, insurance underwriting drives the insurance process and business. The cost of underwriting includes the cost incurred by insurance companies when determining whether to accept or decline an insurance contract and the associated risk.

Like the code review and audit industry, the insurance market is dominated by a cartel of a few core players. Accordingly, it is also subject to high barriers to entry for new market entrants and democratized access to decentralized insurance of code reviews and audit and collective decision making on code review risk does not exist.

Decentralized code review underwriting democratizes the functions of code reviews. This makes the code review market more efficient because it frees untapped sources of power and knowledge.

Code Review DAO (CRDAO)

The CRDAO is tackling many of the issues that afflict the modern code review market.

The CRDAO provides a decentralized community-driven code review process that utilizes a bidding process on code reviews to drive prices down. It provides open access for code reviews from anyone who qualifies – not just members of the code review cartel. At the same time, it provides full incentivization for community code reviews through its decentralized governance framework.

Given its universal access and price discovery methodology (through a public bidding process), the CRDAO creates low barriers to entry in the code review market. Anyone can join the CRDAO by submitting high quality code reviews through the CRDAO portal.

The CRDAO also uses a community policing and audit methodology for code reviews. This is enabled by the CRDAO governance model. Moreover, the CRDAO governance and policing functions ensure less duplication of code reviews. CRDAO code review process maintains strong incentives for community driven code review audits.

The CRDAO can function as a first-round code review or multi round code review with different code review teams. The CRDAO also offers decentralized underwriting of CRDAO code reviews.

In summary, customers receive low priced, high quality code reviews with community validation of reviews and, optionally, insurance on code review outcomes.

Feedback Loops

CRDAO provides an early feedback loop on code reviews for the developer community at a much lower price than the traditional code review and audit market.

CRDAO crowd controls filter out idiosyncratic code reviewer preferences. Code reviews in legacy code review firms are often highly subjective which leads to rather suboptimal outcomes without crowd controls. No single one developer may agree on a given set of code and its intended functionality and quality in achieving the coded objectives. This can be attributed to different programming languages with different styles and unique and often idiosyncratic preferences. Instead, the CRDAO mandates that reviews are subject to crowd review and policing votes by the CRDAO collective. Accordingly, code reviewers are less likely to engage in highly idiosyncratic reviews as they would need to fear slashing and loss of standing in the community.

The CRDAO also fills the void left in legacy reviews without common standards. The CRDAO creates a compendium of reviews and through it a common standard for code reviews that are otherwise lacking in legacy code review environments. Should a collective review code reviews under a common set of standards, the standards help guide both the reviewers and the collective come to a common form of expectations on the applied functionality and quality outcomes for the code.

Speed

The feedback provided by the CRDAO for the developer community enables risk-taking for dev teams who wish to move quickly through their governance and upgrade process, which in turn enables accelerated growth and scaling of experimentation.

Code Testing and Experimentation

The code reviews provided by the CRDAO provide a first instantiation of flat hierarchy-driven decentralized peer reviewed code reviews. All testing performed by the CRDAO follows the core standards established for the CRDAO community. All testing and standards are subject to constant review and experimentation and are continually, dynamically and evolutionarily updated in a constant feedback loop between all constituents, that is between the CRDAO VAs, the CRDAO clients, and public bidders and applicants for CRDAO VA status.

Price Discovery

The existing code review market does not provide publicly transparent pricing of code review services. As such, the existing market arguably harms the public for the benefit of the cartel and

its clients. In the existing system, the cartel pricing is not disclosed because both the client and code reviewer may not benefit from public scrutiny of the cartel prices. [------]

The CRDAO uses a unique and fully transparent price discovery mechanism for code reviews. The CRDAO's price discover mechanism works as follows:

- Job Poster posts the job on the CRDAO portal
- DAO internal - Job price discovery
    - Internal DAO bids are collected in a table format next to the job posting terms and show the bids up and down on the terms 1. Time +- 2. Price - + 3. Reputation score range of the bidder (actual reputation of bidder is within the range), 4. Reputation stake of the bidder
        - See table example in sheet 2: https://docs.google.com/spreadsheets/d/1rqtxdro8-ww9caFahHpjC2Y-3pgMGu7v6Ik1sIK8YXU/edit#gid=1073029785
        - At the end of the internal bidding period - user reviews all the bids and selects the winner and prices are revealed
        - If the user does not select a winner after a grace period, it automatically goes to public bidding and prices are never revealed.
        - Bidding is not public during the bidding process and (fully anonymized through reputation score ranges)
        - All bids on the post and winner will become public at the of the selection when the user has picked the winner
- Public - Job price discovery
    - If no bids for *n* days internally in the VA pool, the job post will be made public automatically for public bidding - public bidding follows these parameters.
        - External bidder posts DoS fee to get on the Service DAO (admin approval)
        - External bids are collected in a table format next to the job posting terms and show the bids up and down on the terms 1. Time +- 2. Price - +
        - At the end of the external bidding period - user reviews all the bids and selects the winner
        - If the user does not select a winner after a grace period, it automatically ends the bidding process with no winner.
        - Bidding is not public during the bidding process and (anonymized if preferred by bidder)
        - All bids on the post and winner will become public at the end of the selection when the user has picked the winner
- If the user accepts a bid (DAO internal or public) - user now has to post deposit into smart contract

This price discovery mechanism serves a key public service function in that it enables full price transparency for consumers based on the visibility of the internal and external bids on job posts. This price transparency is unique and unprecedented in a market that is otherwise dominated by a cartel.

Price discovery is a key public services function because without public pricing, consumers cannot realistically select the service provider that provides the highest value to the customer. The lack of transparency enables insider deals to the detriment of the clients, who are forced into the price the cartel dictates.

Community Governance

Developers on the CRDAO Network can participate directly in the evolution of the CRDAO Network through participation in the CRDAO DAO (CRDAO). The CRDAO follows a decentralized framework of governance that was developed by Craig Calcaterra and Wulf Kaal[50] and is further enhanced and implemented by the code review DAO (CRDAO). CRDAO members get paid to participate in the governance of the CRDAO.

Controls and Insurance

Customers benefit from the CRDAO model of code reviews through key controls and insurance:

1. Customers get full price discovery and transparency of pricing through the CRDAO bidding portal where all bids are displayed
2. Customers get to select the bid that pleases them best, whether it is objectively the best bid or subjectively the selected bid the customer prefers. The customer is in complete control
3. In future iterations, the CRDAO offers the customer a form of insurance if the code review does not correspond with the contractual obligations of the parties. The insurance will be paid from the CRDAO Code Review Pool (CRP) which consists of the assets raised during the CRDAO token offering as well as 10% of any incoming code review job revenue.

CRDAO Code Review Process

The code review process of the CRDAO revolves around CRDAO community engagement which minimizes issues of lack of crowd controls, lowers time requirements for code reviews, lowers prices of code reviews, increases developer participation, and increases overall feedback.

Conclusion

The CRDAO provides key solutions for the existing problems of modern code reviews in a fully decentralized community driven environment. As the CRDAO iterates on its design and moves into subsequent phases, the larger web3 community will continue to benefit from the key innovations.

Bibliography

Nicole Davila, Ingrid Nunes, “A systemic literature review and taxonomy of modern code review.” In *The Journal of Systems & Software*, 177 (2021) 110951.

Jacek Czerwonka, Michaela Greiler, Jack Tilford, “Code Reviews Do Not Find Bugs: How the Current Code Review Best Practice Slows Us Down”, 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), 2015 pp. 27-28, doi: 10.1109/ICSE.2015.131

Ebert, F., Castor, F., Novielli, N. *et al.* An exploratory study on confusion in code reviews. *Empir Software Eng* **26,** 12 (2021).

Laura Macleod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, Jacek Czerwonka, “Code Reviewing in the Trenches: Understanding Challenges and Best Practices.” *Institute of Electrical and Electronics Engineers,* Volume: 35, Issue 4 (2018), pp. 34 – 42.

dos Santos, E.W., Nunes, I. Investigating the effectiveness of peer code review in distributed software development based on objective and subjective data. *J Softw Eng Res Dev* **6,** 14 (2018).

Oleskii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. “Do people and participation matter” *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME*), 2015, pp. 111-120, doi: 10.1109/ICSM.2015.7332457

Baysal, Olga, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. “Investigating Technical and Non-Technical Factors Influencing Modern Code Review.” *Empirical software engineering : an international journal* 21, no. 3 (2015): 932–959.

McIntosh, Shane, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. “An Empirical Study of the Impact of Modern Code Review Practices on Software Quality.” *Empirical software engineering : an international journal* 21, no. 5 (2015): 2146–2189.

Baum, Tobias, Hendrik Leßmann, and Kurt Schneider. "The choice of code review process: A survey on the state of the practice." In *International Conference on Product-Focused Software Process Improvement*, pp. 111-127. Springer, Cham, 2017.

A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 712-721, doi: 10.1109/ICSE.2013.6606617.

A. Alami, M. Leavitt Cohn and A. Wąsowski, "Why Does Code Review Work for Open Source Software Communities?," *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 1073-1083

Mortiz Beller, Alberto Bacchelli, Andy Zaidman & Elmar Juergens, Modern Code Reviews in Open Source Projects: Which Problems Do They Fix? MSR 2014: Proceedings of the 11th Working Conference on Mining Software RepositoriesMay 2014 Pages 202–211

---

[1] The Author is grateful to many members of the CRDAO who provided valuable freedback. The author would especially like to thank Josh Bykowski for outstanding research assistance.

[2] Nicole Davila, Ingrid Nunes, "A systemic literature review and taxonomy of modern code review." In *The Journal of Systems & Software*, 177 (2021) 110951.

[3] *Id at 1*.

[4] Jacek Czerwonka, Michaela Greiler, Jack Tilford, "Code Reviews Do Not Find Bugs: How the Current Code Review Best Practice Slows Us Down", 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), 2015 pp. 27-28, doi: 10.1109/ICSE.2015.131

[5] *Id at 1.*

[6] Nicole Davila, Ingrid Nunes, "A systemic literature review and taxonomy of modern code review." In *The Journal of Systems & Software*, 177 (2021) 110951.

[7] Ebert, F., Castor, F., Novielli, N. *et al.* An exploratory study on confusion in code reviews. *Empir Software Eng* **26,** 12 (2021).

[8] *Id at 2*

[9] *Id at 2.*

[10] *Id.*

[11] Jacek Czerwonka, Michaela Greiler, Jack Tilford, "Code Reviews Do not Find Bugs: How the Current Code Review Best Practice Slows Us Down", 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), 2015 pp. 27-28, doi: 10.1109/ICSE.2015.131

[12] *Id at 28.*

[13] Laura Macleod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, Jacek Czerwonka, "Code Reviewing in the Trenches: Understanding Challenges and Best Practices." *Institute of Electrical and Electronics Engineers,* Volume: 35, Issue 4 (2018), pp. 34 – 42.

[14] Ebert, F., Castor, F., Novielli, N. *et al.* An exploratory study on confusion in code reviews. *Empir Software Eng* **26,** 12 (2021).

[15] Nicole Davila, Ingrid Nunes, “A systemic literature review and taxonomy of modern code review.” In *The Journal of Systems & Software*, 177 (2021) 110951.

[16] dos Santos, E.W., Nunes, I. Investigating the effectiveness of peer code review in distributed software development based on objective and subjective data. *J Softw Eng Res Dev* **6,** 14 (2018).

[17] Jacek Czerwonka, Michaela Greiler, Jack Tilford. “Code Reviews Do not Find Bugs: How the Current Code Review Best Practice Slows Us Down”, 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), 2015 pp. 27-28, doi: 10.1109/ICSE.2015.131

[18] dos Santos, E.W., Nunes, I. Investigating the effectiveness of peer code review in distributed software development based on objective and subjective data. *J Softw Eng Res Dev* **6,** 14 (2018).

[19] *Id at 15.*

[20] *Id.*

[21] *Id at 16.*

[22] Oleskii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. “Do people and participation matter” *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME*), 2015, pp. 111-120, doi: 10.1109/ICSM.2015.7332457

[23] *Id at 116.*

[24] Ebert, F., Castor, F., Novielli, N. *et al.* An exploratory study on confusion in code reviews. *Empir Software Eng* **26,** 12 (2021).

[25] *Id at 12.*

[26] *Id at 13.*

[27] *Id.*

[28] *Id.*

[29] *Id.*

[30] Nicole Davila, Ingrid Nunes, “A systemic literature review and taxonomy of modern code review.” In *The Journal of Systems & Software*, 177 (2021) 110951.

[31] *Id at 7.*

[32] Ebert, F., Castor, F., Novielli, N. *et al.* An exploratory study on confusion in code reviews. *Empir Software Eng* **26,** 12 (2021).

[33] *Id at 12.*

[34] *Id.*

[35] *Id.*

[36] *Id.*

[37] *Id.*

[38] Oleskii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. "Do people and participation matter" *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME*), 2015, pp. 111-120, doi: 10.1109/ICSM.2015.7332457

[39] *Id at 117.*

[40] *Id at 118.*

[41] Laura Macleod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, Jacek Czerwonka, "Code Reviewing in the Trenches: Understanding Challenges and Best Practices." *Institute of Electrical and Electronics Engineers,* Volume: 35, Issue 4 (2018), pp. 34 – 42.

[42] Laura Macleod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, Jacek Czerwonka, "Code Reviewing in the Trenches: Understanding Challenges and Best Practices." *Institute of Electrical and Electronics Engineers,* Volume: 35, Issue 4 (2018), pp. 34 – 42.

[43] *Id.*

[44] *Id.*

[45] Baysal, Olga, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. "Investigating Technical and Non-Technical Factors Influencing Modern Code Review." *Empirical software engineering: an international journal* 21, no. 3 (2015): 932–959.

[46] *Id at 932.*

[47] *Id at 937.*

[48] *Id at 938.*

[49] Id.

[50] Calcaterra, Craig and Kaal, Wulf A. and Andrei, Vlad, Blockchain Infrastructure for Measuring Domain Specific Reputation in Autonomous Decentralized and Anonymous Systems (February 18, 2018). U of St. Thomas (Minnesota) Legal Studies Research Paper No. 18-11, Available at SSRN: https://ssrn.com/abstract=3125822 or http://dx.doi.org/10.2139/ssrn.3125822