How Code Reviews Relate to the Bugs in Software

Shivani Gadipe
College of Engineering
Oregon State University
Corvallis, OR
gadipes@oregonstate.edu

Tu Lam
College of Engineering
Oregon State University
Corvallis, OR
lamtu@oregonstate.edu

Alexander Yang Rhoads
College of Engineering
Oregon State University
Corvallis, OR
rhoadsal@oregonstate.edu

***Abstract*—**
Code reviews are a standard practice in software development, being crucial for improving software quality, reducing bugs, and maintaining consistency within projects. However, there is an ongoing debate about the point of diminishing returns, such as when does the frequency and depth of reviews stop leading to significant improvements in software quality? This paper investigates the relationship that bug occurrences have with code review frequency, the number of code reviewers, and with code review depth by extracting those data with GitHub's API, then plugging them into the evaluation metrics. Using Pearson's Correlation Coefficient, a high positive correlation was found with the number of reviews and individual reviewers, as well as the depth of reviews. Furthermore, using Cliff's Delta, it was found that the frequency of code reviews has a large absence of overlap with bug occurrences. The data points to a high associativity of bug occurrences with review frequency, reviewer count, and review depth, but the reasoning behind that associativity is ultimately inconclusive.

***Index Terms*—Code review depth, code review frequency, Cliff's Delta, correlation coefficient, Pearson Product-Moment Correlation, internal validity, external validity**

I. Introduction

Code reviews are a standard practice in software development to improve code quality, reduce bugs, and maintain consistency across projects. It is unknown if frequent and in-depth code reviewers, made by more individual reviewers results in higher quality code. In this study, we aim to better understand this. To do this, we will conduct a quantitative analysis tracking the amount of bug reports against the amount of code reviews, the number of reviewers, and the depth of the reviews themselves, looking to see if those latter factors have any impact on the former.

As a vital part of the software development process, reviews help teams identify and fix errors before they reach production. However, there is often uncertainty about how frequently reviews should be conducted and how detailed they need to be to maximize their benefits.

By understanding the relationship between code review practices and the number of bugs in a software release, this research aims to help development teams to:

- Save time and resources by avoiding unnecessary or redundant reviews.
- Enhance team efficiency by focusing on effective review strategies.

This study will provide actionable information to guide teams in adopting better review practices, ultimately leading to more reliable and user-friendly software.

Being a major quality assurance technique for development teams, code reviews are meant to prevent bugs and improve code quality. However, the best practices in terms of number of reviews, detail of reviews, and the number of reviewers are unknown. Evaluating the effect these factors have on the amount of bug occurrences will provide data that can inform the best practices of code reviewing.

The code review frequency, number of reviewers, depth of reviews, and amount of bug occurrences will be extracted from the Hyrax repository [1], throughout five distinct stages in its life-cycle. That data will be plugged into correlation and overlap functions to evaluate the relationship that bug occurrences have with the other factors.

When measuring code review frequency, depth, and number of reviewers, it's important to consider certain assumptions about the process in order to properly research in this area. One key assumption is that developers are familiar with the code review practices (e.g., frequency, depth, and review criteria) and that these practices are consistent within the team. This ensures that we can isolate the impact of review frequency, review depth, and quantity of reviewers on bug counts that will aid with the evaluation of correlation and overlap. Another assumption involves review depth, which is calculated by taking the total amount of review comments within a time period, and dividing it by the total number of commits in that same time period, effectively measuring the average length of a review per code change. The assumption is that the number of comments in a review reflects the level of detail of a review, which will allow us to quantify the depth of a code review. Finally, we assume that closed GitHub tickets labeled 'bug' specifies the number of bugs found within the repository, as this is the main way the Hyrax repository denotes found bugs. This allows us to count the number of bugs within the life-cycle of a version of Hyrax.

As a quality assurance mechanism, several aspects of code reviews are open to the question of examination. Concepts such as the effects of review depth, review frequency, and the reviewer count inspire questions regarding how they influence the number of bugs within the software. Below is the research question that we are using to evaluate these effects:

1) RQ1: Does the number of code reviews, reviewers, or

review depth of a project have a correlation with the number of bugs found after release?

The dataset focuses on evaluating the open-source project Hyrax [1], a Ruby on Rails engine made by the Samvera community. The repository is publicly available on GitHub, as are the issues and the PR (Pull Request). Four quantitative parameters are taken or calculated from metadata extracted using the GitHub API: bug occurrences, code review frequency, reviewer count, and code review depth. The parameters are taken from five distinct periods in Hyrax's life-cycle, such as Hyrax V1 going into V2, V2 to V3, V3 to V4, V4 to V5, and V5 to V5.04 (the latest version of Hyrax). The entire collection of parameters is then plugged into formulas for the correlation coefficient and overlap, which produces the evaluation data.

Pearson Correlation Coefficient serves as the evaluation metric for correlation, whereas Cliff's Delta acts as the overlap evaluation metric. These formulas will take the relevant data across the five life-cycles of Hyrax and output a value that describes these relations. Both of these formulas elaborate on the relationship between two values. Bug occurrence will act as the parameter whose relationship with the other three is examined in both instances.

II. Background and Motivation

In the field of software development, code reviews are a crucial quality process where developers review each other's code for errors, improvements, and adhere to the best practices. The goal is to reduce defects and enhance the maintainability of software. However, there remains an ongoing challenge to determine the optimal frequency and depth of these reviews to maximize software quality, as well as the ideal number of individuals handling reviews. Specifically, it is unclear when the frequency and depth of reviews, along with number of reviewers, begin to offer minimal returns in terms of reducing bugs in software. For example, when and if the frequency and depth of reviews stop making a significant increase in quality. Overly thorough reviews might not yield enough additional benefit to justify the time spent on code review, as one study suggests that 54% of the reviewed changes introduced bugs in the code. The findings also showed that both personal metrics, such as reviewer workload and experience, and participation metrics, such as the number of developers involved, are associated with the quality of the code review process [2].

To address this problem, we analyzed the relationship between the number of bug occurrences, within a particular software life-cycle, to the number of code reviews, the number of reviewers, and the depth-or detail-of those reviews in the same life-cycle. Frequency in this context refers to the number of individual code reviews, tracked per life-cycle of a particular release, or version, of software. Reviewers are the distinct individuals who write code reviews. The depth of a code review is the total number of review comments, divided by commits approved within a pull request. Our approach is to extract data from an open-source project, and use to analyze the correlation and overlap that review patterns have with number of bugs found in a release cycle per release cycle basis. In statistical terms, correlation is the extent to which two variables are related, indicating a relationship or dependence. Overlap is the measure of whether boundaries for two or more variables coincide to a significant extent. By recording these, we aimed to provide support on whether there was a key point at which increasing the frequency and depth of reviews, as well as the number of reviewers, resulted in a significant reduction or increase in bugs.

One prominent case which serves as motivation for understanding the extent code reviews can curb bugs is found in the 2012 case of the Knight Capital Group. The financial services firm lost $440 million in 45 minutes due to a bug in its trading software during an update rollout [3]. In 2005, they had moved a section of computer code to an earlier point in the code sequence within an automated equity router, breaking functionality. Despite the broken function, it was kept in the program, which proved to be an issue in 2012, when they were rolling out a new update, the accidentally made use of the outdated code, causing the program to start making bad equity orders, leading to the aforementioned losses. One of the leading causes of the disaster specified was the lack of rigorous code reviews or double checks to locate potential flaws in the code, leading to an old problem lying in wait for years before it blew up. Stronger code reviews could have prevented the fiasco. That makes highlighting what makes those reviews strong important too. What we hope to do with this study is to highlight what kind of review depth, frequency, and reviewer count makes for strong reviews.

III. Related Work

By evaluating the relationship review frequency, reviewer count, and review depth have with bug occurrences in software, this study seeks to measure code review quality. Literature regarding code review quality can be put into several broad categories. There are direct surveys with participant feedback that makes up the dataset, there are developmental studies that track the results of an iterated system, as well as direct data extraction from repositories. Despite the larger differences, they mostly aim to either improve understanding of code reviews or their practices.

Oussama Ben Sghaie and Houari Sahraoui's 2024 study primarily intersects with research in three key areas of code review automation: quality estimation, comment generation, and code refinement [4]. Each of these tasks has been addressed separately in the literature, with recent efforts focusing on leveraging pre-trained language models. However, existing approaches often treat these tasks in isolation, ignoring the connections between them. Quality estimation methods have been used to assess code quality, comment generation techniques focus on generating helpful review comments, and code refinement strategies aim to improve code quality. Our work differentiates itself by proposing the art of looking into the correlation and overlap that review frequency, reviewer count, and review depth have with bug occurrences. Even though one is looking into improving the technique of code review,

this work gives us insight, as an example of how accurate code reviews increase can decrease bugginess for a software launch.

The 2023 study by Asif Kamal Turzo and Amiangshu Bosu contributes to the field of code review effectiveness, focusing specifically on the factors influencing the usefulness of code review comments (CRCs) in Open Source Software (OSS) development [5]. Previous research has explored aspects of code review automation, defect prediction, and reviewer recommendations. This work intersects with these areas by exploring how both technical and linguistic elements of a CRC influence its perceived usefulness, while also considering contextual and participant-related factors. The findings suggest a need to reconsider reviewer recommendation models, urging a more context-sensitive approach to improve code review effectiveness. This work can help provide a good contrast with our use of correlation and overlap, exploring the extent that different aspects of reviews affect their usefulness.

M. Nejati, M. Alfadel and S. McIntosh's 2023 study addresses the intersection of build system maintenance and code review practices, particularly in relation to build specifications [6]. Build systems play a critical role in integrating source code into executables, and maintaining these systems is known to be a challenge due to the complexities they introduce. Code review has been widely studied in the context of production and test code, but its application to build specifications remains under-explored. This research expands that area of study, finding that comments on changes for build specifications are more likely to point out defects than rates reported in the literature for production and test code, and evolvability and dependency-related issues are the most frequently raised patterns of issues. This study indicates to our work that the relevancy of a code review has a significant effect on bugs present in code, and the more reviews there are, the less likely bugs are to shown up in a software launch.

A 2021 survey by Wang, Dong, et al. sought to answer three research questions: what the extent of link sharing in review discussion was, if the number of links in review discussion correlate with review time, and what the common intentions of link sharing are [7]. Focusing on reviews in the Openstack and Qt projects, the study limits its dataset to closed reviews with a comment by the reviewer. To answer the research questions, quantitative analysis is applied for each. Qualitative analysis is also applied to questions involving reviewer intention. In the case of link number, the number of reviews was measured against number links, whether it was the reviewer or author who shared the link, and where the link leads (if it's related to internal project sites or not, and if it links to a GitHub activity, communication channel, a video or photo, a memo, or a review). Review time correlation was analyzed by measuring line adds, deletes, patch sizes (as in quantity of adds and deletes), patch purpose, number of files, number of review iterations, number of prior patches made by the author, number of comments, number of author comments, number if reviewer comments, number of reviewers, number of external links, internal links, and total links. These variables were fitted into an Ordinary Least Squares multiple regression model. With regards to link sharing intention, seven categories were identified, including providing context, elaboration, clarification, explanation of necessity, proposals, expert suggestions, and informing split patches. The three most common of the categories, context, necessity explanations, and elaborations, were directly measured as an overall percentage. Additional developer feedback was also sought out as a part of the research. This included the degree to which developers agreed with the conclusions of the study, as a survey on what of the seven link intention categories do they most commonly have. Similar to our study, the paper seeks to analyze correlation of code review performance with certain practices, though it focusing on the specific topic of link sharing, as opposed to a more broad one like code frequency and depth.

In the 2021 research paper by AlOmar, Eman Abdullah, et al., a case study using the work of 24 developers at Xerox examines refactoring practices in the context of modern code reviews [8]. Five research questions are covered: what motivates developers in the context of modern code reviews, how developers document refactoring for code reviews, what are the challenges that reviewers face with regards to refactoring changes, what mechanisms do developers and reviewers use to ensure correctness after refactoring, and how developers and reviewers assess and perceive the impact of refactoring on code quality. The study's dataset for qualitative analysis used surveys given out to the developers asking about their background, motivation, documentation practices, challenges as a reviewer, verification practices, and the implications of refactoring. The data for qualitative analysis was metadata extracted from review requests, using the Mann-Whitney U test to compare data. Evaluation with regards to direct survey results considers Xerox's own internal guidelines, as well as percentages cataloging response quantities. This paper heavily deviates from evaluation practices of our own, directly asking input from code reviewers. Still, both papers share a similar approach to metadata analysis.

A 2025 study (Oliveira, Delano, et al.) tries to understand how code review comments improve code understandability [9]. The research questions are as follows: How often do reviewers ask for code understandability improvements in code review comments, what are the main issues pertaining to code understandability in code review, how likely are understandability improvement comments to be accepted, what code changes are found in understandability improvements, to what extent are accepted code understandability improvements reverted, and if linters contain rules to detect the identified code understandability smells. Data is collected from the code reviews of open source Java projects, which are checked for references to understandability. The reviews are grouped by their topic, such as if the suggestions pertain to understandability, rate of acceptance for improvements, or what changes to understandability exist. This information is separated into different categories, and tallied into percentages. Further analysis takes the data and puts it into the Kappa coefficient. The direct taking of data from code reviews mirrors our own approach. However, the study requires the team to analyze

and categorize their data before it can be quantified, whereas research regarding depth and frequently can be immediately put into numbers.

The 2022 study by Gonçalves, Pavlína Wurzel, et al. tests whether giving explicit review strategies improves code review performance and reduces cognitive load. There are two review questions, each comprising two smaller subproblems [10]. The first: if guidance in reviews leads to higher review effectiveness (proportion of defects found), as well as higher efficiency (functional defects found over time). The second asks if the effectiveness and efficiency of guidance is mediated by lower cognitive load. The study took the form of an experiment, hiring 70 developers and tracking their background (education, prior experience, and age). Developers were split into three groups and given either nothing, a checklist that lists what to look out for, or a guided checklist that specifies the scope of what to look out for. Cognitive load is measured via a questionnaire. The results of the three groups are measured against the developer's individual backgrounds and the answers from the cognitive load surveys. Both our own paper and theirs seeks to answer the same question of improving code performance, but instead of extracting metadata from repositories and comparing it, this paper receives data via direct behavioral experiments.

The 2024 document by Frommgen et al. is an internal Google study looking into the capability of machine learning to automate and streamline the code review process [11]. The research question is the same as the premise: the extent to which a machine learning model can aid in all aspects of the code review process. Data took the form of the code review suggestions the studied ML model gave across iterations, and how participants responded to them. Participant responses (as in what they did with the suggestions) were categorized and calculated as a percentage. The suggestions by the model were separated into different key categories, such as suggestions to complicated code reviews and open ended reviews, and were analyzed qualitatively. Verbal feedback from participants was also analyzed on a qualitative basis. The Google study seeks to improve the code review process, similar to our own, but the avenue of its research revolves around how specific technology (ML models) can improve review practices, rather than analyzing the performance of review practices directly.

The 2022 paper by Gonçalves, Pavlína Wurzel, et al. investigates how developers perceive code review conflicts and addresses interpersonal conflicts during code reviews as a theoretical construct [12]. The study seeks to answer five research questions: how developers perceive and experience conflicts during code review, what are conflicts during code review like, what the positive and negative consequences of conflicts during code review are, what factors intervene in the conflict dynamics, and what strategies developers use to prevent and manage conflicts. Data from this study revolved around a series of interviews asking participating developers inquiries derived from the research questions themselves. Answers were compiled in broad categories and analyzed for implications on the basis of the developers' individual answers. As a direct feedback survey, this paper heavily deviates in terms of dataset origin from our own. Despite that, it still is trying to examine exact trends in code review behavior.

M. B. Ada and M. U. Majid's 2022 paper on student code peer reviews attempts to develop a system to increase the engagement of programming students during peer code reviews [13]. The research question is the same as the premise, but more specifically it means to develop a system "that aims to provide course instructors with the capability to create and conduct formative and summative peer code review exercises for their students... improve student engagement with the review process and motivation to achieve better learning." This takes the form of an experiment that sees participating students provide feedback to reviewers, as well as giving rewards for both completing reviews and handing out high quality ones. Results from the experiment were taken directly from participants in the shape of feedback forms. Answers to these forms were then quantified into percentages. This is another direct feedback survey, and one aimed at resolving educational needs rather the broadly professional ones. Our own paper analyzes information from repositories, comparing different quantities from that set, to answer research questions regarding code review practices. However, this study is relevant in how it both qualitatively and quantitatively analyzes its data, which showcases avenues of approach for our own study.

IV. Approach

The approach to answering our research question involves extracting, comparing, and analyzing data from the open-source software repository Hyrax. Given that Hyrax has been actively developed since 2016, we expect it to have a rich dataset of user-reported issues, particularly related to bug fixes. We collect the data with four quantitative parameters in mind: the code review depth, code review frequency, distinct reviewer count, and bug occurrences. The data is gathered via the GitHub API communicating with a series of Python scripts that tally the data and organize it into five date ranges corresponding to Hyrax release deployments. Once the data is compiled, it is organized into a series of tables and graphs tracking each distinct time period.

Following that, we will input the extracted data into the statistical formulas Cliff's Delta and Pearson Correlation Coefficient to determine the overlap and correlation between the bug occurrence compared to the three other parameters. These formulas will produce values that are used to interpret the relationship of the parameters. Figure 1 showcases the flow of data.

V. Evaluation

A. Dataset

This study will examine metadata for code reviews and bug tickets from the open-source Hyrax repository, which is extracted using the GitHub API [1]. The GitHub API will look into the closed pull requests, issues, and ticket data of the repository. Python scripts communicate with the API to organize the extracted data in five time periods based

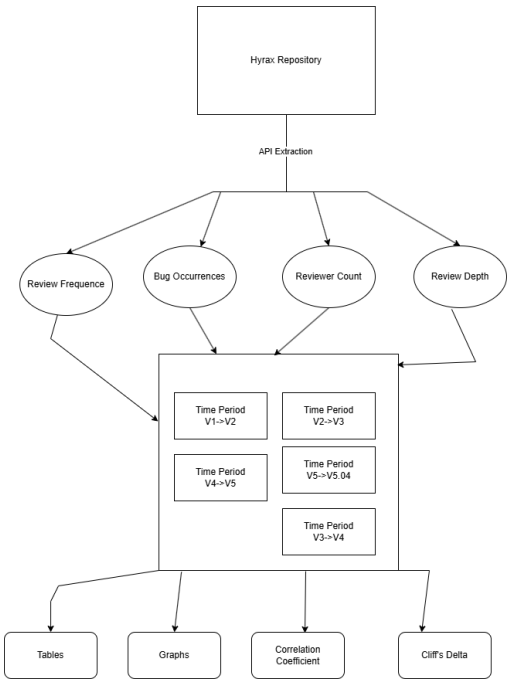

Fig. 1. Flow of Data

off the creation date of the data. Additionally, these scripts will plug the data into the evaluation formulas to create the corresponding metrics. The API will specifically extract the frequency of code reviews, the number of code reviewers, the depth of code reviews, and the number of bug occurrences.

- Frequency of Code Reviews: Refers to the number of extracted tickets labeled 'review' within pull requests.
- Code Reviewers: Refers to the amount of distinct people performing code reviews, which is fetched by counting the amount of unique user IDs amount people who posted reviews.
- Code Review Depth: a measurement of the average detail of code reviews. The number of changes requested by the reviewer, divided by the number of changes made by the PR owner. Approved changes are tracked by counting the number of approved commits in a PR.
- Bug Occurrences: Tracks the quantity of closed tickets labeled as 'bug'.

Additionally, the equation for depth is defined as such:

$$Depth = \frac{No.\ of\ changes\ requested\ by\ the\ reviewer}{No.\ of\ changes\ made\ by\ the\ owner\ of\ PR} \quad (1)$$

These four quantitative parameters are organized based on if they were created in five particular time periods. These five time periods correspond to the start of one version release of Hyrax, to the start of the next version's release. This is a means of tracking and evaluating Hyrax's metadata on a per-version basis.

- V1 $\rightarrow$ V2: Corresponds from May 23 2017 to Nov. 9 2017.
- V2 $\rightarrow$ V3: Corresponds from Nov. 9 2017 to Mar. 24 2021.
- V3 $\rightarrow$ V4: Corresponds from Mar. 24 2021 to May 30 2023.
- V4 $\rightarrow$ V5: Corresponds from May 30 2023 to Mar. 1 2024.
- V5 $\rightarrow$ V5.04: Corresponds from Mar. 1 2024 to Feb. 10 2025.

B. Metrics

To evaluate the trend in correlation and overlap that review frequency, amount of reviews, and review depth have the number of bug occurrences, we use the following metrics:

1) **Pearson Correlation Coefficient:** This metric, measured as the r-value, is meant to measure the strength of a linear relationship between two variables. The r-value is a measure between $-1 \leq r \leq +1$. The closer r is to zero, the weaker the relationship. As r gets closer to positive 1, it showcases a stronger positive correlation, while an r that's closer to -1 indicates a stronger negative correlation.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

2) **Cliff's Delta:** This metric is meant to compare two samples of ordinal data. It tracks overlap, or the tendency of one value to being larger than another. It is measures between $-1 \leq \delta \leq +1$. As the delta gets closer to a positive or negative 1, it indicates a larger absence of overlap between the two samples, while a value closer to 0 indicates a greater degree of overlap.

$$\delta = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \delta(i, j)$$

In terms of how the parameters fit into the formula variables, bug occurrence will be used to calculate every evaluation metric. Specifically, with the correlation coefficient, it will be the Y value in every instance, while each of remaining parameters are the x value for their unique r-value. In the case of Cliff's Delta, j = bug occurrence, whereas the parameters are the i values in their own corresponding delta values.

These metrics were selected based on their relevance in understanding the relationship that the review parameters have with bug occurrences. The values given by these formulas provide a direct measure of that relationship.

C. Experiment Procedure

The experiment starts with the extraction of the qualitative parameters in the metadata of the Hyrax repository. This is done by communicating with the GitHub API via a series of Python scripts, which focuses data collection exclusively on closed issues and linked pull requests (PRs). The scripts tally up the parameter data, organizing them into five time periods, based on their extracted date of creation. This outputs the data, which is then inputted into a further series of

Release Version	V1 -> V2	V2 -> V3	V3 -> V4	V4 -> V5	V5 -> V5.04
Bug Occurrences	278	909	337	124	75
Review Frequency	1545	3920	1728	738	202
Review Depth	177%	99%	107%	65%	35%
Reviewer Count	49	74	41	25	13

Fig. 2. Data Table

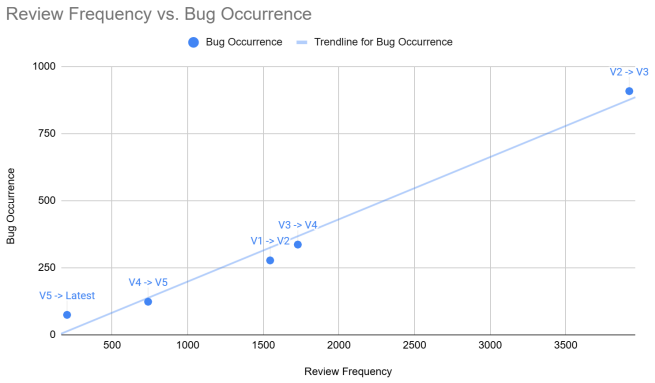

Fig. 3. Review Frequency vs. Bug Occurrence

Python scripts, which plugs the data into the correlation and overlap formulas that calculate the evaluation metrics. The data regarding the four parameters and their time periods, as well as the correlation coefficient and overlap delta are recorded along the way.

D. Results

After collecting the data, as seen in figure 2, we found that through each distinct version (e.g. $v1 \rightarrow v2, v2 \rightarrow v3, v3 \rightarrow v4, etc...$), increased occurrences of bugs actually trended with the number of reviews, distinct reviewers, and degree of review depth. This can also be seen in the trend-line of figures 3, 4, and 5, which all feature a positive trend-line around the values even before taking into account evaluation metrics.

Calculating the correlation coefficient largely saw the same trends, with bug occurrences featuring large positive correlation values for the three other parameters. This is especially true for review frequency and review count, which both nearly have the full r-value of 1.

1) **Review Frequency vs. Bug Occurrence:** $r = 0.996$
2) **Review Depth vs. Bug Occurrence:** $r = 0.758$
3) **Reviewer Count vs. Bug Occurrence:** $r = 0.953$

Calculating Cliff's Delta results in a slightly more varied picture. Review frequency has a moderate to large absence of overlap with bug occurrences. Review depth and reviewer count have overwhelming absences of overlap, both being stuck at the far end of the value range at -1.

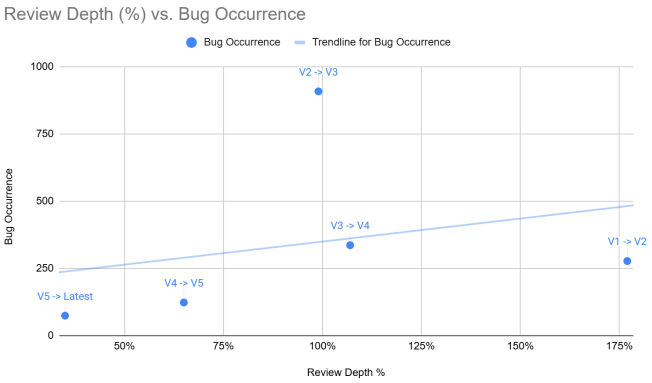

Fig. 4. Review Depth vs. Bug Occurrence

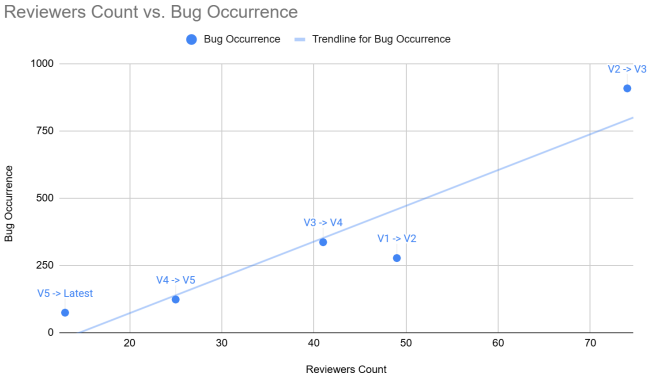

Fig. 5. Number of Reviewers vs. Bug Occurrence

1) **Review Frequency vs. Bug Occurrence:** $\delta = 0.68$
2) **Review Depth vs. Bug Occurrence:** $\delta = -1.0$
3) **Reviewer Count vs. Bug Occurrence:** $\delta = -1.0$

In terms of answering the research question, if bug occurrences correlate to the other three quantitative factors, the result is a resounding yes. The three factors all showed high positive correlation. But beyond that, it can be difficult to draw definitive conclusions. While we know there is a strong positive correlation between bugs and the review factors, that does not get at the logic as to why. There are many possibilities as to why. While the data could be interpreted as showing an increase in bugs with a higher amount of code reviews, the data could equally be interpreted as just showing how buggier code requires a higher amount of reviews to work through. Examining overlap also gives mixed messages. The delta value for review frequency shows that there isn't much overlap, that really just says that reviews will always be higher than bug occurrences, with depth and review count always being smaller. That could support either end of the interpretation. There just isn't enough data to come to a stronger conclusion. Further research would need to be done, examining the qualitative features of code review relative to the amount of bugs in software.

VI. Discussion and Threats to Validity

Several limitations and threats to the validity of this paper exist. In terms of limitations, the decision to limit the scope to examining one repository, which was taken to manage time constraints, blunted the effectiveness of the data. The lack of diversified data meant that there was a diminished ability to evaluate the repository data in terms of statistical functions (which required a wider data pool), comparisons (which would need contrasting data points), and especially for hypothesis testing functions, where the lack of randomized samples in the dataset meant that few of them could be used.

For threats to validity, construct validity, which concerns the relation between the theory behind the experiment and the observed findings, has one regarding the definition of code depth. Code depth found the percentage of review comments, relative to the amount of commits, as a means of quantifying the detail of a code review. That means that more comments equals a larger depth value, which makes the assumption that a review is more detailed in tandem with the amount of comments. While generally true, that isn't always the case, some reviewers will format their reviews in the form of large block comments. So in certain instances, a small number of review comments would hide very detailed reviews.

Another threat to validity is external validity, that concern whether the results can be generalized outside the scope of the study. This also has to do with the decision to study only one repository. That repository, Hyrax, is a very specific type of program, a large-scale web application engine. That ensures that code reviews may be more likely to concern large-scale issues in the code. There's a risk that the evaluations made might not be true for smaller projects, where the potentially smaller reviews have a different relationship to the number of bugs and review length.

VII. Contributions

In the interest of establishing the habits of bug occurrences vis-à-vis code reviews, the Hyrax repository was measured for its bugs, review frequency, reviewer count, and review depth. The extracted data was then put into the correlation coefficient and cliff's delta to ascertain the relationship between the amount of bug occurrences and the level of the other three factors. All three other factors were shown to have a strong positive correlation with bugs, while all values also lacked overlap with bug occurrences. As a result, the review parameters show strong correlation to the observed amount of bugs, but by itself, without qualitative analysis, the information can taken several different ways. The high correlation value could be seen as representative of buggy code requiring more reviews to fix it, or that higher review totals results in worse code. Further research examining either a broader set of data, or the qualitative aspects of code reviews, would need to be done to come to a stronger conclusion.

References

[1] samvera, "Hyrax," https://github.com/charlespwd/project-title, 2025.

[2] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, and M. W. Godfrey, "Investigating code review quality: Do people and participation matter?" in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 111–120.

[3] C. Saltapidas and R. Maghsood, "Financial risk the fall of knight capital group," 2018.

[4] O. Ben Sghaier and H. Sahraoui, "Improving the learning of code review successive tasks with cross-task knowledge distillation," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3643775

[5] A. K. Turzo and A. Bosu, "What makes a code review useful to opendev developers? an empirical investigation," *Empirical Software Engineering*, vol. 29, no. 1, p. 6, Nov 2023. [Online]. Available: https://doi.org/10.1007/s10664-023-10411-x

[6] M. Nejati, M. Alfadel, and S. McIntosh, "Code review of build system specifications: Prevalence, purposes, patterns, and perceptions," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 1213–1224.

[7] D. Wang, T. Xiao, T. Patanamon, R. G. Kula, and K. Matsumoto, "Understanding shared links and their intentions to meet information needs in modern code review:," *Empirical Software Engineering*, vol. 26, no. 5, Sep Sep 2021, copyright - © The Author(s) 2021. This work is published under http://creativecommons.org/licenses/by/4.0/ (the "License"). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License. [Online]. Available: https://oregonstate.idm.oclc.org/login?url=https://www.proquest.com/scholarly-journals/understanding-shared-links-their-intentions-meet/docview/2549481919/se-2?accountid=13013

[8] E. A. AlOmar, H. AlRubaye, M. W. Mkaouer, A. Ouni, and M. Kessentini, "Refactoring practices in the context of modern code review: An industrial case study at xerox," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021, pp. 348–357.

[9] D. Oliveira, R. Santos, B. de Oliveira, M. Monperrus, F. Castor, and F. Madeiral, "Understanding code understandability improvements in code reviews," *IEEE Transactions on Software Engineering*, vol. 51, no. 1, pp. 14–37, 2025.

[10] P. W. Gonçalves, E. Fregnan, T. Baum, K. Schneider, and A. Bacchelli, "Do explicit review strategies improve code review performance? towards understanding the role of cognitive load," *Empirical Software Engineering*, vol. 27, no. 4, Jul Jul 2022, copyright - © The Author(s) 2022. This work is published under http://creativecommons.org/licenses/by/4.0/ (the"License"). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License. [Online]. Available: https://oregonstate.idm.oclc.org/login?url=https://www.proquest.com/scholarly-journals/do-explicit-review-strategies-improve-code/docview/2660495235/se-2?accountid=13013

[11] A. Froemmgen, J. Austin, P. Choy, N. Ghelani, L. Kharatyan, G. Surita, E. Khrapko, P. Lamblin, P.-A. Manzagol, M. Revaj, M. Tabachnyk, D. Tarlow, K. Villela, D. Zheng, S. Chandra, and P. Maniatis, "Resolving code review comments with machine learning," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 204–215. [Online]. Available: https://doi.org/10.1145/3639477.3639746

[12] P. Wurzel Gonçalves, G. Çalikli, and A. Bacchelli, "Interpersonal conflicts during code review: Developers' experience and practices," *Proc. ACM Hum.-Comput. Interact.*, vol. 6, no. CSCW1, Apr. 2022. [Online]. Available: https://doi.org/10.1145/3512945

[13] M. B. Ada and M. U. Majid, "Developing a system to increase motivation and engagement in student code peer review," in *2022 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, 2022, pp. 93–98.