Exploring the Impact of DevOps and Agile Practices from the Perspective of Source Code Analysis

Jalles Daniel Barros[1], **Luiz Carlos Jesus**[1] **Johnatan Oliveira**[2] , **Juliana Padilha**[1]

[1]University Única
Ipatinga – MG – Brazil

[2]Federal University of Lavras (UFLA/ICTIN)
São Sebastião do Paraíso – MG – Brazil

jallesdaniel@gmail.com, luiz04nl@gmail.com
johnatan.oliveira@ufla.br, analise.sistemas@unicaead.com.br

Abstract. *Implementing agile practices in software development processes promises improvements in product quality and process productivity. However, there are few reports of failures from which to learn, and it is not always clear which practices are effective in specific contexts. This paper investigates how the use of DevOps and agile methodologies affects code quality in open-source projects hosted on GitHub. A total of 57 Java repositories, obtained through queries to the GitHub GraphQL API, were analyzed and categorized into four groups: i) projects that do not use agile nor DevOps, ii) projects that use agile but not DevOps, iii) projects that use both agile and DevOps, and (iv) projects that use DevOps but not agile. The results indicate that the majority of analyzed projects belong to group iii. Furthermore, the absolute number of code smells and their proportion relative to the number of lines of code are highest in group iii, suggesting a positive correlation between the use of both agile and DevOps practices and an increase in code smells.*

Keywords: *DevOps, Agile, GitHub, Quality*

1. Introduction

In recent years, the field of software development has benefited from a significant paradigm shift with the large-scale adoption of Agile and DevOps methodologies [3]. These practices are highly regarded for their potential to maximize product quality and productivity, thereby addressing the increasing demand for efficient and robust software delivery processes [4]. Agile methodologies, characterized by iterative development, continuous feedback, and adaptive planning, aim to deliver functional software incrementally [10]. DevOps, meanwhile, emphasizes the collaboration between development and operations teams, seeking to automate and streamline the software delivery pipeline from code development to production deployment [6].

The motivation for this study is derived from the mixed results reported in the literature regarding the impact of Agile and DevOps practices on code quality. While some studies [1, 14] suggest that these methodologies can lead to improved code quality and security, others indicate an increase in technical debt [16, 2] and code smells [12]. For instance, Almeida et al. found that combining DevOps and Agile practices can enhance software delivery speed and product quality, but also noted potential increases in code smells and technical debt due to the pressures of rapid iterations and continuous deployment [1]. Similarly, Ruk et al. highlighted that while Agile practices improve responsiveness and customer satisfaction, they often require extensive training and cultural adjustments to mitigate potential quality issues [14].

Given these conflicting findings, this study aims to provide a deeper understanding of how Agile and DevOps practices impact code quality in open-source projects hosted on GitHub. By analyzing 57 Java repositories, we categorize them into four distinct groups: projects that do not use Agile nor DevOps, projects that use Agile but not DevOps, projects that use both Agile and DevOps, and projects that use DevOps but not Agile. This categorization enables us to draw comparisons and identify trends in code quality metrics across different development practices.

Our analysis reveals that projects employing both Agile and DevOps practices tend to exhibit higher numbers of code smells and bugs compared to those using either methodology alone or neither. Specifically, the mean number of bugs for projects using both methodologies is approximately 650, compared to 300 for Agile only, 100 for DevOps only, and 20 for projects using neither. Similarly, the number of code smells is highest in the DevOps only group, followed by the Agile and DevOps group, the Agile only group, and the neither group. These findings suggest that while the integration of Agile and DevOps can introduce certain quality issues, it may also enhance security ratings and overall responsiveness to change.

Additionally, our findings indicate that the integration of Agile and DevOps practices contributes to a collaborative environment that improves team efficiency and adaptability. The results suggest a strong association between the adoption of DevOps practices and the implementation of Agile methodologies, highlighting the synergistic relationship between these two approaches. Furthermore, larger teams, despite introducing more total code smells, appear to maintain a lower density of code smells relative to the size of the project, suggesting that effective collaboration and task distribution can mitigate some quality issues.

The paper is organized as follows: Section 2 covers the study design, including the goal, evaluation steps, and dataset. Section 3 discusses the applicability evaluation. Section 4 presents the analysis results, focusing on the impact of Agile and DevOps on code quality, Agile's integration in DevOps projects, and the influence of contributors on code quality. Section 5 reviews related work. Section 6 addresses threats to validity, and Section 7 concludes with a summary of findings and implications for future research.

2. Study Design

This section describes the study design to address RQ1, RQ2, and RQ3. Section 2.1 shows the central goal of this study. Section 2.2 presents the Evaluation Steps. Finally, Section 2.3 presents the dataset.

2.1. Goal and Research Question

The central goal of this study is to investigate the impact of Agile and DevOps practices on code quality in open-source software projects. We aim to understand how these methodologies, both individually and in combination, influence various code quality metrics, including code smells, bugs, vulnerabilities, and security ratings. By conducting this analysis, we seek to provide empirical evidence that can guide practitioners in optimizing their software development processes. To achieve this goal, we formulated the following research questions (RQs):

RQ1: –How do Agile methodologies and DevOps influence code quality in software projects?

RQ2:– How do projects that utilize DevOps integrate Agile methodologies?

RQ3:–How does the number of contributors affect code quality in software projects?

2.2. Evaluation Steps

The evaluation process was structured into five steps. In the following, we describe the steps. The initial step in was the selection of the programming language. We opted for Java [1], given its widespread use and ranking among the top ten most popular programming languages on GitHub [5]. This choice was strategic, aimed at maximizing the relevance and applicability of our study within the vast landscape of software development. Subsequently, the second step involved the cloning of the top one thousand projects from GitHub, prioritized by their popularity, which was quantified through the number of stars each project received. This approach was necessitated by the inherent limitations associated with the direct cloning capabilities of GitHub repositories. In this step, we delete projects non source code (see Section 2.3).

The third phase of our evaluation focused on the identification of DevOps practices within these projects. This was achieved by examining the presence of specific configuration files indicative of DevOps integration. We searched for a set of key files, including .github/workflows/*, .cicleci/config.yaml, Jenkinsfile, .gitlab-ci.yml, azure-pipelines.yml, .travis.yml, .harness/ya, and bitbucket-pipelines.yml. The presence of these files served as a primary indicator of a project's alignment with contemporary software development methodologies.

In the fourth step, we analyzed the frequency of commits integrated into the default branch of each project using the PyDriller tool. This analysis was pivotal in determining the agility of the projects. We defined the criteria for agility based on the average interval between commits. Specifically, repositories that demonstrated an average integration interval exceeding fifteen days were classified as non-agile. This threshold was chosen because agile methodologies typically emphasize rapid iteration and frequent integration of changes, which are fundamental for accelerating development cycles and enhancing responsiveness to changing requirements. Consequently, a longer interval suggests a deviation from these agile principles. Thus, repositories with more frequent commits were classified as agile, reflecting their alignment with the agile methodology's core practices of continuous integration and regular updates.

The final step involved the construction and data extraction from the projects using SonarQube and Docker. Each repository marked as cloned underwent a building process wherein SonarQube was deployed to extract and analyze project metrics. This phase proved to be the most challenging due to its technical complexity and the time investment required. Out of the initial sample, 57 repositories were successfully built using Maven, Gradle, or Ant and were analyzed with SonarQube.

[1]https://www.tiobe.com/tiobe-index/

2.3. Data Set

To conduct this study, we selected Java as the programming language for analyzing projects on GitHub. Java is a popular language with a favorable range of tools for detecting code smells and other code vulnerabilities. This choice was made to ensure the reliability and relevance of our findings. We start by searching for the top 1,000 most popular Java projects on GitHub, ranked by the number of stars. This popularity metric was chosen to ensure that the selected projects had a significant level of community engagement and usage.

At this point, we filtered out projects that were not software-related, such as documentation repositories or educational examples. Projects with only one developer or only one active developer were excluded, as such projects may not provide a comprehensive view of collaborative development practices. We also removed projects that had not been updated in the last 12 months to ensure that the analyzed projects were actively maintained. Finally, we excluded projects where Java was not the predominant programming language to maintain consistency in our analysis. After applying these filters, we identified 57 projects that were suitable for analysis.

3. Results

This section aims to answer RQ1, RQ2 and RQ3. Section 3.1 presents results about Impact of Agile and DevOps on Code Quality to answer RQ1. Section 3.2 presents the results about Agile Integration in DevOps Projects to answer RQ2. Finally, Section 3.3 discusses Contributors' Influence on Code Quality to answer RQ3.

3.1. Impact of Agile and DevOps on Code Quality

This section presents the results of our first research question (RQ1), defined as follows.

RQ1: –How do Agile methodologies and DevOps influence code quality in software projects?

To address RQ1, we categorized the data into four groups: i) Neither Agile nor DevOps, ii) Agile Only, iii) DevOps Only, and iv) Both Agile and DevOps. The first group, Neither Agile nor DevOps, consists of projects that do not employ Agile methodologies or DevOps practices. The second group, Agile Only, includes projects that utilize Agile methodologies but do not incorporate DevOps practices. The third group, DevOps Only, comprises projects that implement DevOps practices without employing Agile methodologies. The final group, Both Agile and DevOps, represents projects that integrate both Agile methodologies and DevOps practices. Descriptive statistics for code quality metrics were calculated for each group using SonarQube. SonarQube checks for code compliance against a set of coding rules and considers violations as Technical Debt (TD) items [2][16]. Metrics analyzed include code smells, bugs, vulnerabilities, and security ratings [2]. Code smells indicate potential maintainability issues, bugs are defects causing failures, and vulnerabilities are weaknesses that compromise security [2].

Figure 1 presents the average number of bugs for each group. The Both Agile and DevOps group has the highest mean number of bugs (≈ 650), followed by the Agile Only group (≈ 300), the DevOps Only group (≈ 100), and the Neither Agile nor DevOps group (≈ 20). These differences highlight the varying impact of Agile and DevOps practices on bugs. Figure 2 shows the average number of vulnerabilities for each group. The Both Agile and DevOps group has the highest mean number of vulnerabilities ($\approx$ 55), followed by the Neither Agile nor DevOps group ($\approx$ 25), the DevOps Only group ($\approx$ 5), and the Agile Only group ($\approx$ 2). These differences highlight the varying impact of Agile and DevOps practices on vulnerabilities.

Figure 3 presents the mean security rating for each group. The Both Agile and DevOps group has the highest mean security rating ($\approx$ 1250), indicating better security practices. The DevOps Only group follows ($\approx$ 400), then the Agile Only group ($\approx$ 250), and the Neither Agile nor DevOps group has the lowest mean security rating ($\approx$ 150). These differences highlight the impact of Agile and DevOps practices on security ratings. Figure 4 presents the average number of code smells for each group. The DevOps Only group has the highest mean number of code smells ($\approx$ 275), followed by the Both Agile and DevOps group ($\approx$ 225), the Agile Only group ($\approx$ 125), and the Neither Agile nor DevOps group ($\approx$ 50). These differences highlight the varying impact of Agile and DevOps practices on the presence of code smells.

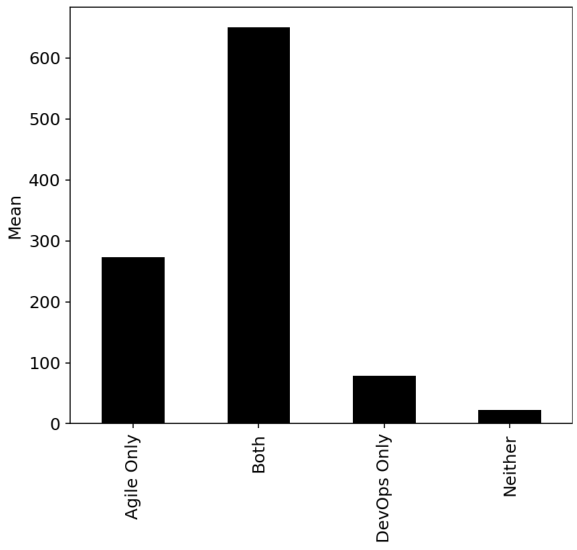

Figure 1. Mean of Bugs by Group

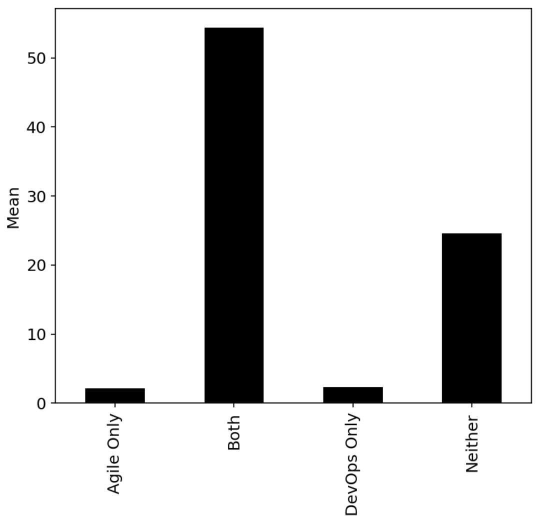

Figure 2. Mean of Vulnerabilities by Group

To identify statistical differences in the data, we applied an ANOVA test to evaluate the impact of Agile methodologies, DevOps practices, and their interaction on metrics such as code smells, bugs, vulnerabilities, and security ratings, all obtained from SonarQube. The ANOVA results for code smells indicated that neither the use of Agile methodologies (F=1.46, p=0.232) nor DevOps practices (F=0.82, p=0.370) had a statistically significant impact on the number of code smells. Similarly, the interaction between Agile and DevOps practices did not show a significant effect (F=0.03, p=0.867).

For the bugs metric, ANOVA results showed that neither Agile methodologies (F=1.34, p=0.252) nor DevOps practices (F=0.79, p=0.379) significantly impacted the number of bugs. The interaction between Agile and DevOps also did not present a significant effect (F=0.32, p=0.571). In the case of vulnerabilities, ANOVA results indicated that neither Agile methodologies (F=0.13, p=0.722) nor DevOps practices (F=1.28, p=0.262) had a statistically significant impact. The interaction between Agile and DevOps

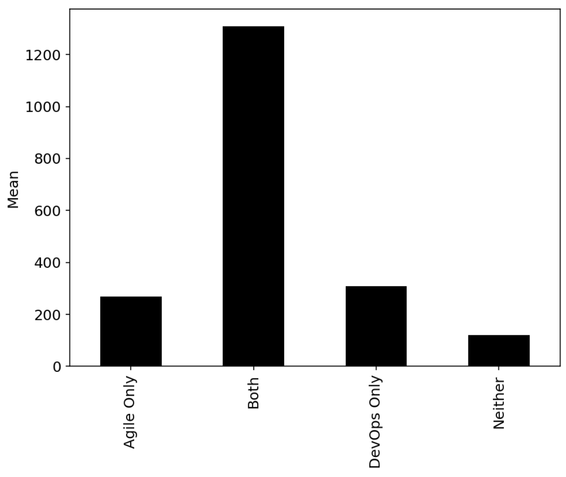

Figure 3. Mean of Security Rating by Group

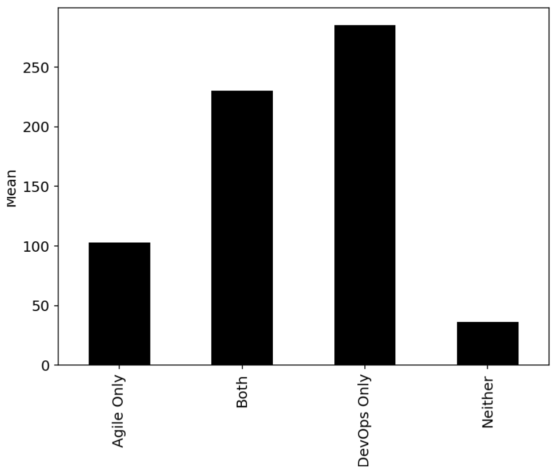

Figure 4. Mean of Code Smells by Group

showed no significant effect (F=0.52, p=0.475). Regression analysis further investigated the relationships between the use of Agile methodologies and DevOps practices on code quality metrics. For code smells, the model R-squared value was 0.071, indicating that only 7.1% of the variability could be explained by the independent variables. The coefficients for Agile (t=1.220, p=0.228) and DevOps (t=0.912, p=0.366) were not statistically significant.

For bugs, the R-squared value was 0.066, with coefficients for Agile (t=1.166, p=0.249) and DevOps (t=0.892, p=0.376) also not significant. Regarding vulnerabilities, the model had an R-squared value of 0.039, with non-significant coefficients for Agile (t=-0.359, p=0.721) and DevOps (t=-1.139, p=0.260). For the security rating, the R-squared value was 0.107. The coefficient for Agile (t=1.795, p=0.078) showed a marginally significant effect, while the coefficient for DevOps (t=0.826, p=0.413) was not significant. Overall, the results suggest that neither Agile methodologies nor DevOps practices significantly impact code smells, bugs, and vulnerabilities. However, Agile methodologies may have a marginal effect on improving security ratings. Further investigation with more data is necessary to confirm these findings.

3.2. Agile Integration in DevOps Projects

This section presents the results of our first research question (RQ2), defined as follows.

RQ2:– How do projects that utilize DevOps integrate Agile methodologies?

To address the second research question, which investigates whether projects using DevOps also include Agile methodologies, we conducted an analysis using contingency tables and the chi-square test for independence. The chi-square test evaluates whether there is a significant association between the use of DevOps and Agile methodologies. Based on the p-value, we determined the statistical significance of this association. Out of the projects analyzed, 15 projects did not use either DevOps or Agile methodologies, 3 projects used Agile methodologies but not DevOps, 14 projects used DevOps but not Agile methodologies, and 25 projects used both DevOps and Agile methodologies. The chi-square test results showed a value of 9.2716 and a p-value of 0.0023 with

1 degree of freedom, indicating a statistically significant association between the use of DevOps and Agile methodologies, as the p-value is less than 0.05.

The expected frequencies under the null hypothesis of independence were as follows: 9.16 projects using neither DevOps nor Agile, 8.84 projects using Agile but not DevOps, 19.84 projects using DevOps but not Agile, and 19.16 projects using both DevOps and Agile. Given the significant p-value, we reject the null hypothesis and conclude that there is a significant association between the use of DevOps and Agile methodologies in the analyzed projects. The results from the contingency analysis and chi-square test suggest that the implementation of DevOps is often accompanied by the use of Agile methodologies. This outcome indicates that teams adopting DevOps practices are likely to also implement Agile methodologies, highlighting a pattern of combined adoption in software projects.

3.3. The Influence of Contributors on Code Quality

This section presents the results of our first research question (RQ3), defined as follows.

RQ3:–How does the number of contributors affect code quality in software projects?

To address the third research question, we conducted a series of statistical analyses. Specifically, we examined the relationship between the number of contributors and code quality metrics, including the absolute number of code smells and the number of code smells per line of code (LOC). We performed linear regression to evaluate the relationship between the number of contributors and the absolute number of code smells, as well as the number of code smells per LOC. This helped us understand the trend and quantify the effect of the number of contributors on code quality metrics. We calculated the Pearson correlation coefficients to assess the strength and direction of the linear relationship between the number of contributors and the code quality metrics.

Figure 5 presents the relationship between the number of contributors and the absolute number of code smells in the analyzed software projects. The scatter plot shows individual data points, while the regression line provides an overall trend. The shaded area around the regression line represents the confidence interval. We observe a positive trend, indicating that as the number of contributors increases, the absolute number of code smells also tends to increase. This suggests that larger teams may introduce more code smells into the project, possibly due to the complexity and challenges of coordinating work among many contributors. However, it is important to consider other factors that might contribute to this trend, such as the size of the project and the processes in place for code review and quality assurance.

Figure 6 presents the relationship between the number of contributors and the number of code smells per line of code (LOC) in the analyzed software projects. Similar to Figure 1, the scatter plot shows individual data points, while the regression line provides an overall trend with a shaded confidence interval. We observe a slight negative trend, indicating that as the number of contributors increases, the number of code smells per LOC tends to decrease. This suggests that larger teams might be more effective in maintaining code quality relative to the size of the project, potentially due to better distribution of tasks, more thorough code reviews, or other quality control measures. However, this trend is less pronounced, and further investigation would be needed to draw definitive conclusions.

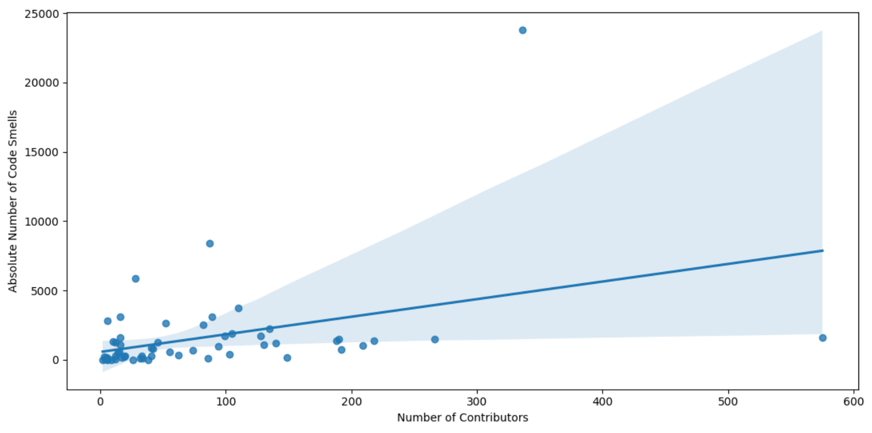

Figure 5. Number of Contributors vs Absolute Number of Code Smells

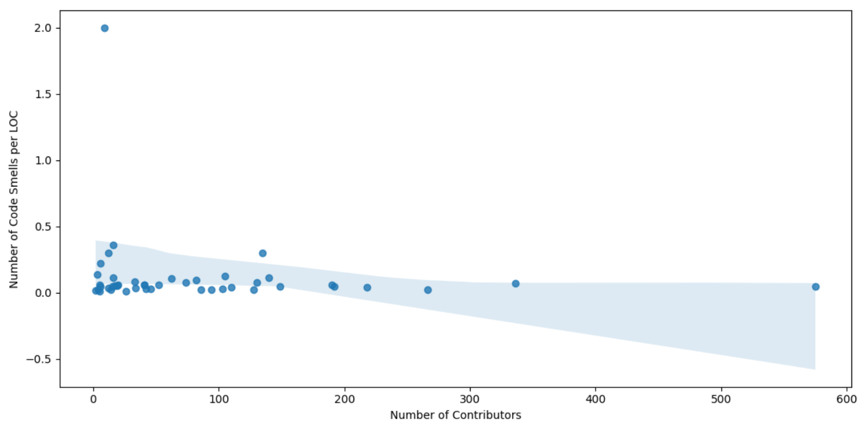

Figure 6. Number of Contributors vs Number of Code Smells per LOC

Note that Figures 5 and 6 provide insights into how the number of contributors impacts code quality in software projects. While the absolute number of code smells tends to increase with more contributors, the code smells per LOC shows a slight decreasing trend. This could imply that while larger teams introduce more total code smells, they may also be more efficient in maintaining a lower density of code smells relative to the size of the project.

4. Related Work

Understanding how the combination of agile methodologies and DevOps impacts code quality has become a recurring theme in software engineering studies [1, 14]. The premise is that agile development tends to sacrifice software quality due to its focus on functionality and rapid product delivery, as stated by the authors [9].

Almeida et al. (2022) conducted an in-depth analysis of the synergy between DevOps and Agile practices, revealing significant improvements in software delivery speed and product quality when these methodologies are combined. Their study highlights the potential for reduced development cycles and enhanced customer satisfaction [1]. Ruk et al. (2019) surveyed the adoption of Agile methodologies and their impact on software quality. They identified key issues related to Agile adoption, including resistance to change and the need for extensive training to fully leverage Agile benefits [**?**]. Their findings underscore the importance of addressing cultural and organizational challenges to optimize Agile implementation.

Karhapää et al. (2024) explored the evidence-based quality-aware Agile software development process, emphasizing the need for a balanced approach that integrates quality considerations into Agile workflows. Their research suggests that while Agile methodologies can accelerate delivery, they must be coupled with rigorous quality assurance practices to prevent technical debt and maintain high software standards [9]. Several studies aim to understand the challenges faced in adopting the DevOps approach in software projects. Jayakody and Wijayanayake (2021) and Hemon et al. (2020) highlight the difficulties in integrating development and operations teams. The collaboration between historically separate teams is essential for the success of DevOps, yet it presents significant challenges [8, 7].

Perera et al. (2017) conducted a comprehensive study on how DevOps practices impact software quality, utilizing the CAMS (Culture, Automation, Measurement, Sharing) framework. Their findings indicate that automation is a critical factor in improving software quality, and that a strong DevOps culture can significantly enhance team collaboration and efficiency. The study also emphasizes the importance of continuous measurement and knowledge sharing for sustaining high-quality outcomes in DevOps environments [13]. Mishra and Otaiwi (2020) performed a systematic mapping study to investigate the relationship between DevOps practices and software quality. Their research demonstrated a positive correlation between the implementation of DevOps and improvements in various quality metrics, including reliability, maintainability, and performance. The study suggests that DevOps practices contribute to a more stable and resilient software development process [11].

Our research approach explicitly addresses the limitations expressed in previous studies, related to the lack of research on how the use of DevOps and agile methodologies affects the quality of open-source code in projects hosted on GitHub. By analyzing a diverse set of repositories, we aim to provide a comprehensive understanding of the impact of these methodologies on code quality metrics such as code smells, bugs, vulnerabilities, and security ratings.

5. Threats to Validity

In this section, we discuss the potential threats to the validity of our study, following the guidelines proposed by Wohlin et al.. We address threats to internal, external, conclusion, and construct validity [15].

Internal Validity – A key threat to internal validity is the potential for confounding variables, such as the varying experience levels of contributors or differences in project management practices. Additionally, the use of SonarQube for code quality measurement, while robust, may have inherent biases or limitations in detecting certain types of code smells or vulnerabilities. We mitigated these threats by using a diverse sample of projects and ensuring consistent application of SonarQube metrics. External Validity – We focus on Java projects hosted on GitHub may limit the generalizability of our results to projects using different programming languages or hosted on other platforms. The popularity-based selection criteria may bias our sample towards well-maintained projects, potentially overlooking less popular but equally relevant projects.

Conclusion Validity – Potential threats to conclusion validity include the accuracy and precision of the statistical methods employed. While we used ANOVA and regression analysis, there is a risk of Type I or Type II errors due to inherent variability in software project data. To mitigate this, we ensured comprehensive data preprocessing and applied statistical techniques. Construct Validity – The primary constructs are Agile practices, DevOps practices, and code quality. A threat to construct validity is the operationalization of these constructs, particularly the identification and categorization of Agile and DevOps practices. The reliance on project documentation and commit messages to infer the use of these practices may not fully capture their depth and consistency. We addressed this by using multiple sources of evidence and cross-referencing project documentation with actual practices observed in the codebase.

6. Conclusion

This study aimed to investigate the impact of Agile and DevOps practices on code quality in open-source software projects. By analyzing 57 Java repositories on GitHub, we categorized the projects into four groups based on their use of Agile and DevOps practices and examined code quality metrics using SonarQube. Our findings indicate that projects utilizing both Agile and DevOps practices tend to have a higher incidence of code smells, bugs, and vulnerabilities compared to those employing only one or neither of these methodologies. This suggests a positive correlation between the use of both Agile and DevOps practices and an increase in these code quality issues. However, it is important to note that these methodologies also contribute to higher security ratings, highlighting their potential in enhancing security aspects of software projects. The study also revealed a significant association between the use of DevOps and Agile methodologies, suggesting that teams adopting DevOps practices are likely to integrate Agile methodologies as well. This combined adoption pattern underscores the synergy between these two approaches in modern software development practices. Furthermore, our analysis of the number of contributors showed that larger teams tend to introduce more code smells. However, the density of code smells relative to the size of the project decreases with more contributors, indicating that larger teams may be more effective in maintaining code quality proportional to project size.

Future research should expand the dataset to include projects from various programming languages and ecosystems, explore the specific Agile and DevOps practices that influence code quality, and investigate the long-term effects of these practices through longitudinal studies.

Replication Package

We provide spreadsheets with GitHub projects, processed data (code smells, source code metrics), and analysis scripts to facilitate replication and validation. The replication package for this study is available at: GitHub.

References

[1] F. Almeida, J. Simões, and S. Lopes. Exploring the benefits of combining devops and agile. *Future Internet*, 14(2):63, 2022.

[2] Maria Teresa Baldassarre, Valentina Lenarduzzi, Simone Romano, and Nyyti Saarimäki. On the diffuseness of technical debt items and accuracy of remediation time when using sonarqube. *Information and Software Technology*, 128:106377, 2020. ISSN 0950-5849.

[3] María Cecilia Bastarrica, Germán Espinoza, and Jacqueline Marín. Implementing agile practices: the experience of tsol. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Woubshet Behutiye, Pilar Rodríguez, and Markku Oivo. Quality requirement documentation guidelines for agile software development. *IEEE*, 10, 2022.

[5] Hudson Silva Borges, André C. Hora, and Marco Túlio Valente. Understanding the factors that impact the popularity of github repositories. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344, 2016.

[6] Samuel Ferino, Marcelo Fernandes, Anny Fernandes, Uirá Kulesza, Eduardo Aranha, and Christoph Treude. Analyzing devops teaching strategies: An initial study. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering*, SBES '21, page 180–185, New York, NY, USA, 2021. Association for Computing Machinery.

[7] A. Hemon, B. Lyonnet, F. Rowe, et al. From agile to devops: Smart skills and collaborations. *Information Systems Frontiers*, 22(4):927–945, 2020.

[8] J. A. V. M. K. Jayakody and W.M.J.I. Wijayanayake. Challenges for adopting devops in information technology projects. In *2021 International Research Conference on Smart Computing and Systems Engineering (SCSE)*, volume 4, pages 203–210, 2021.

[9] Pertti Karhapää, Woubshet Behutiye, Pertti Seppänen, Pilar Rodríguez, Markku Oivo, Xavier Franch, Silverio Martínez-Fernández, Lidia López, Michał Choraś, Alessandra Bagnato, Sanja Aaramaa, and Jari Partanen. Evidence-based quality-aware agile software development process: Design and evaluation. *IEEE Access*, 12:86487–86512, 2024.

[10] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. Relationship of devops to agile, lean and continuous deployment - a multivocal literature review study. In *International Conference on Product Focused Software Process Improvement*, 2016.

[11] Alok Mishra and Ziadoon Otaiwi. Devops and software quality: A systematic mapping. *Computer Science Review*, 38:100308, 2020.

[12] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Fausto Fasano, Rocco Oliveto, and Andrea De Lucia. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering*, 23:1188 – 1221, 2017.

[13] Pulasthi Perera, Roshali Silva, and Indika Perera. Improve software quality through practicing devops. In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 1–6, 2017.

[14] Sadaquat Ali Ruk, Muhammad Faizan Khan, Sehar Gul Khan, and Syed Maqsood Zia. A survey on adopting agile software development: Issues its impact on software

quality. In *2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, pages 1–5, 2019.

[15] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Computer Science. Springer Berlin Heidelberg, 2012.

[16] Nico Zazworka, Antonio Vetrò, Clemente Izurieta, Sunny Wong, Yuanfang Cai, Carolyn Budinger Seaman, and Forrest Shull. Comparing four approaches for technical debt identification. *Software Quality Journal*, 22:403–426, 2014.