Asia-Pacific Journal of Information Technology and Multimedia 14(1) 2025: 339 – 350
https://doi.org/10.17576/apjitm-2025-1401-19

Exploring DevOps Metrics: A Study on Code Maintainability and DevOps Deployment Practices

Penerokaan Metrik DevOps: Kajian tentang Kebolehselenggaraan Kod dan Amalan Penyebaran DevOps

Sharifah Mashita Syed-Mohamad, Norsyazwani M. Subri,*
Masita @ Masila Abdul Jalil, Amir Ngah, Najihah Ibrahim

Faculty of Computer Science and Mathematics, Universiti Malaysia Terengganu, 21030 Kuala Nerus, Terengganu, Malaysia

**Corresponding author: s.mashita@umt.edu.my*

Received 12 September 2024
Accepted 27 May 2025, Available online 30 June 2025

ABSTRACT

This study explores the crucial role of software maintainability metrics in DevOps environments, where rapid and continuous delivery is paramount. By investigating the correlation between deployment frequency, code churn rates, and code maintainability using commonly used DORA metrics, we aim to shed light on the intersection of DevOps practices and code maintainability. Analysing these metrics provides valuable insights into the evolution and maintenance of the JUnit 5 codebase within DevOps practices. Our findings reveal a consistent deployment frequency, showcasing the team's ability to maintain a steady release pattern. The codebase demonstrates steady growth with notable additions and deletions between releases, with indications of stabilization and bug fixing before major releases. The high frequency of commits reflects an active development process. Despite the codebase's growth, stable complexity levels were maintained, emphasizing the importance of managing code quality metrics. Pearson correlation analysis reveals a strong positive correlation ($R = 0.9887$) between code complexity and codebase changes, underscoring the need to balance both for quality maintenance. The study emphasizes the project's commitment to quality and stability within DevOps, emphasizing the need for ongoing vigilance.

Keywords: DevOps, DevOps metrics, Deployment frequency, Code maintainability, Code complexity

ABSTRAK

Kajian ini meneroka peranan penting metrik penyelenggaraan perisian dalam persekitaran DevOps, di mana penghantaran yang pantas dan berterusan adalah keutamaan. Dengan menyiasat hubungan antara kekerapan penyebaran, kadar perubahan kod, dan penyelenggaraan kod menggunakan metrik DORA yang biasa digunakan, tujuan kami untuk memberi penerangan mengenai persilangan antara amalan DevOps dan kebolehselenggaraan kod.

Analisa metrik ini memberikan pandangan yang berharga tentang evolusi dan kebolehselenggaraan pangkalan kod JUnit 5 dengan amalan DevOps. Penemuan kami mendedahkan kekerapan penyebaran yang konsisten menunjukkan keupayaan pasukan untuk mengekalkan corak pengeluaran yang baik. Pangkalan kod menunjukkan pertumbuhan yang stabil dengan penambahan dan penghapusan yang ketara antara keluaran, dengan petunjuk penstabilan dan pembaikan pepijat sebelum keluaran utama. Kekerapan komit yang tinggi mencerminkan proses pembangunan yang aktif. Walaupun pangkalan kod berkembang, tahap kerumitan yang stabil dikekalkan, menekankan kepentingan pengurusan metrik kualiti kod. Analisis korelasi Pearson mendedahkan korelasi positif yang kuat (R = 0.9887) antara kerumitan kod dan perubahan pangkalan kod, menekankan keperluan untuk mengimbangi kedua-duanya bagi penyelenggaraan kualiti. Kajian ini menekankan komitmen projek terhadap kualiti dan kestabilan dalam DevOps, serta keperluan untuk kewaspadaan berterusan.

Kata kunci**:** DevOps, Metrik DevOps, Kekerapan penyebaran, Kebolehselenggaraan kod, Kerumitan kod

INTRODUCTION

DevOps is a software development and operations strategy that combines the efforts of development (Dev) and operations (Ops) teams to accelerate product creation and simplify maintenance. It emphasizes collaboration, automation, and adherence to best practices to facilitate faster and more controlled development cycles (Azad & Hyrynsalmi, 2023). Firstly, DevOps enables faster development and deployment of software through practices like continuous integration (CI) and continuous delivery (CD). Organizations can release new features and updates more frequently, reducing time-to-market and gaining a competitive edge. Secondly, DevOps promotes improved collaboration and communication among teams by breaking down silos and fostering a culture of collaboration, shared responsibilities, and alignment towards common goals (Bezemer et al., 2019), (Gasparaite et al., 2020), (Arvind, 2022).

The continuous nature of DevOps highlights the importance of maintainability, as highly maintainable code supports rapid development and deployment cycles. However, ensuring continuous maintainability poses challenges that need to be addressed to sustain the efficiency and effectiveness of DevOps practices (Azad & Hyrynsalmi, 2023), (Bezemer et al., 2019), (Riungu-Kalliosaari et al., 2016), (Lwakatare et al., 2019). Despite the emphasis on continuous improvement in DevOps, there seems to be limited information available specifically addressing code maintainability within DevOps initiatives. Understanding and addressing this gap is crucial for optimizing software development processes and ensuring long-term success in DevOps environments (A. Meier, 2021), (Mohammad Zarour et al., 2020).

This study investigates the relationship between DevOps metrics, specifically deployment frequency and key indicators of code maintainability, including code churn rates and code complexity. The primary objective is to evaluate how these metrics collectively influence software maintainability within DevOps-based projects. To this end, a widely used and actively maintained open-source project has been selected as a representative case study. By analyzing these metrics, the study aims to enhance understanding of how DevOps practices impact code maintainability and to offer empirical insights that can inform efforts to optimize software development processes in DevOps environments.
This article begins with an overview of the background and related work, setting the stage for the study's context. Following this, the methodology is detailed in Section 3, outlining the

approach taken to conduct the research. Section 4 delves into the analysis of collected data and presents key findings derived from the study. Finally, the article culminates in Section 5, where conclusions drawn from the analysis are summarized and discussed, providing insights and implications for further research and practice.

RELATED WORKS

A DevOps metric is a "quantifiable, business-relevant, trustworthy, actionable, and traceable indicator that aids organizations in making data-driven decisions to continuously improve their DevOps and software delivery processes" (Amaro et al., 2024). Amaro, Ricardo, and Pereira (2024) identify 22 key DevOps metrics through a comprehensive multivocal literature review, categorizing them as Key Performance Indicators (KPIs). These metrics primarily focus on assessing the performance and effectiveness of DevOps practices, with a particular emphasis on deployment. The metrics are largely deployment-centric, with the top four consistently highlighted in various publications being Time to Restore Service, Lead Time for Changes, Deployment Frequency, and Change Failure Rate. These core metrics are essential for evaluating and enhancing the continuous delivery and operational stability of software systems.

Furthermore, the four key metrics established by the DevOps Research and Assessment (DORA) framework, developed by Google's DORA team, serve as essential indicators of the efficiency of DevOps teams in terms of velocity and reliability (Wickramasinghe, 2023). These metrics, derived from extensive research over seven years on the principles and practical implementations of DevOps, provide a robust foundation for assessing performance (Wickramasinghe, 2023). Figure 1 illustrates the software deployment performance indicator based on the DORA metrics. Studies have shown that companies excelling in these metrics often demonstrate superior software delivery and operational performance (A. Meier, 2021). For instance, Deployment Frequency (DF) measures how frequently code is deployed to production. A higher frequency signifies more rapid delivery of value, with Elite teams deploying multiple times per day. These metrics, also known as Accelerate metrics, are highly effective for evaluating the performance of development processes in microservice-based systems (A. Meier, 2021), (Bezemer et al., 2019).

It appears that most DevOps metrics focus on end-to-end performance and deployment, emphasizing deployment speed, stability, and recovery, whereas traditional metrics like code churn and object-oriented metrics are more focused on code quality and development practices (Syed-Mohamad, S. M., Ngah, A & Ali, A.-F. M. 2025). Various studies emphasize the significance of metrics in DevOps projects, highlighting the need for automated and continuous measurement to adapt to the iterative nature of DevOps methodologies (A. Meier, 2021), (Almashhadani et al., 2023). Ultimately, the goal of DevOps metrics is to provide valuable insights into software development processes, enabling organizations to enhance quality, security, and time-to-market while delivering value to the business (Bermón-Angarita et al., 2023).

Software delivery performance metric	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one hour	Between one day and one week	Between one month and six months	More than six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day	Between one day and one week	More than six months
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0%-15%	16%-30%	16%-30%	16%-30%

FIGURE 1. Dora performance metrics (Wickramasinghe, 2023)

Of particular importance within DevOps is maintainability. By measuring and monitoring these metrics, organizations can gauge the ease with which software can be enhanced, extended, or rectified, thereby impacting the overall quality and agility of the software. However, the specific criteria employed to evaluate software maintainability in DevOps initiatives may vary depending on the organization and project requirements (Lwakatare et al., 2019), (Mohammad Zarour et al., 2020), (Bermón-Angarita et al., 2023).

The DevOps maintainability metric refers to the measure of how easily a software system can be maintained and updated within a DevOps environment (Wickramasinghe, 2023), (Giamattei et al., 2024), (Bermón-Angarita et al., 2023), (Amaro et al., 2024). Software maintainability is crucial as it can reduce a significant portion of a system's life cycle costs (Lomio et al., 2022). Metrics play a vital role in assessing and improving maintainability by helping developers diagnose issues, fix bugs, and meet new requirements. Additionally, requirement traceability aids in tracking requirements throughout the software development process, facilitating change management and preventing confusion (Amaro et al., 2024). Understanding software metrics tools as programs implementing a set of software metrics definitions further enhances the measurement and analysis of maintainability (Giamattei et al., 2024). Therefore, selecting appropriate metrics and ensuring accurate data collection are essential for effectively evaluating and enhancing DevOps maintainability.

Software maintainability is paramount in DevOps environments due to its pivotal role in supporting the rapid development and deployment cycles inherent in DevOps practices. By ensuring that code is highly maintainable, teams can make changes and updates more quickly, facilitating faster delivery and keeping organizations responsive to the ever-changing digital landscape. This adaptability is essential for thriving in dynamic market conditions and meeting evolving customer demands.

Moreover, prioritizing software maintainability leads to a reduction in technical debt, which can otherwise impede future development efforts. Code that is easy to understand, modify, and extend helps teams avoid accumulating technical debt, ensuring that the codebase remains manageable and adaptable over time. By proactively managing technical debt through maintainability, organizations can sustain productivity and prevent slowdowns in development. Additionally, maintainable code fosters improved collaboration between development and operations teams, a core principle of the DevOps philosophy. When code is easily understood and modified across the organization, teams can work together seamlessly to achieve common goals. This collaboration enhances efficiency and effectiveness in delivering high-quality software products, driving innovation and competitive advantage in DevOps environments.

Several studies have conducted systematic reviews on software maintainability prediction and metrics (Jha et al., 2019). These reviews have found that there are numerous models and metrics proposed in the literature to measure and predict software maintainability, but their consistency and ability to accurately predict maintenance effort is still an open research question (Giamattei et al., 2024), (Mohammad Zarour et al., 2020). One approach in DevOps involves harnessing metadata generated during DevOps processes, such as commit history, test coverage, code complexity metrics, and developer involvement in changes (Gunnar Kudrjavets et al., 2022). Additionally, a new model called the Delta Maintainability Model (DMM) has been proposed to assess fine-grained code changes, categorizing them into low and high-risk categories to calculate a delta score, enabling developers to compare and rank the maintainability of commits at a granular level (di Biase et al., 2019) . Furthermore, the automation and visualization of Non-Functional Requirements (NFRs) within a DevOps environment play a crucial role in enhancing code maintainability, emphasizing collaboration, communication, and automation to improve software delivery speed and quality. These combined efforts help ensure that code maintainability is continuously monitored and improved throughout the software development lifecycle (Mishra & Otaiwi, 2020).

In summary, the literature review highlights the significance of DevOps metrics in assessing and improving software maintainability, emphasizing the importance of automated and continuous measurement to adapt to DevOps methodologies. There is a significant need for more consistent normalization and validation of these metrics in practical software maintenance settings (Bermón-Angarita et al., 2023), (Suescún-Monsalve et al., 2021), (M. Gasparaite, K. Naudziunaite, et al., 2020), (Zarour et al., 2019). This analysis of software maintenance metrics holds great promise for advancing the field and improving the effectiveness of DevOps practices.

METHODOLOGY

In this article, we aim to investigate the relationship between frequently used DevOps metrics, specifically deployment frequency, and key code maintainability metrics such as code churn rates and code complexity. Our objective is to assess how these metrics collectively influence software maintainability within DevOps projects. To determine this relationship, we have collected data on deployment frequency metrics, such as the number of deployments per unit of time, and code maintainability metrics, such as code churn rate and code complexity. We have defined the studied metrics as follows:

1. Deployment frequency as the rate at which new code changes are released into production (Wickramasinghe, 2023). It measures how often an organization successfully releases code to production. The terms "frequency of code releases" and "deployment frequency" are often used interchangeably in DevOps.

2. Code maintainability as the ease of managing and updating code over time (di Biase et al., 2019).

3. Code churn rate - refers to the frequency and magnitude of changes made to source code over time(Nagappan & Ball, 2005). Specifically, relative code churn measures quantify the changes made to code files relative to their previous versions, providing insights into the volatility and stability of software components. Nagappan and Ball's research highlights the significance of code churn measures in predicting system defect density, suggesting that higher code churn rates may indicate areas of the codebase prone to defects or instability (Gunnar Kudrjavets et al., 2022).

4. Code complexity is a fundamental aspect of assessing how modifications affect the code's structure and readability. This complexity is typically measured using metrics like cyclomatic complexity, which counts the number of independent paths in the source code, and Halstead complexity, which evaluates program complexity based on the number of operators and operands (di Biase et al., 2019), (Trautsch et al., 2023).

Data Collection: We gathered collected relevant data from an open-source project spanning from February 1, 2016, to May 17, 2024, focusing on the intervals between deployments. Following the methodology outlined by S. Jha et al. (2019), JUnit 5 emerged as an ideal representative DevOps project, primarily for its robust CI/CD pipeline extensively documented in its GitHub repository (Jha et al., 2019). Employing Python, we developed a custom tool to dissect and analyze the main.yml workflow file, shedding light on the project's approach to continuous integration and deployment practices. Our tool meticulously extracted critical information from triggers, jobs, and artifact publications, leveraging GitHub repository to securely access credentials for repository authentication.

We utilized Lizard, an open-source tool, to gather insights into code complexity. Specifically, we employed Lizard to assess the cyclomatic complexity of our sample project, providing valuable metrics for evaluating the intricacy of the code sample, as follows:

1. Total Non-Commented Lines of Code (NLOC): Total lines of code excluding comments; higher values indicate larger codebases.
2. Average Non-Commented Lines of Code per Function (Avg. (NLOC)): Average size of functions in lines of code; smaller values suggest more modular functions.
3. Average Cyclomatic Complexity Number (Avg. (CCN)): Average complexity of functions; lower values indicate simpler functions with fewer branches.
4. Average number of tokens (Avg. Token) (keywords, operators) per function; indicates code complexity.
5. Function Count (Fun Cnt): Total number of functions; more functions suggest modularity but may indicate complexity if not well-defined.
6. Warning Count (Warning Cnt): Number of functions that triggered warnings, usually related to complexity or length.
7. Function Rate (Fun Rt): Rate of functions per module/file; higher rates indicate more functions per module.
8. NLOC Rate (NLOC Rt): Rate of NLOC per module/file; lower rates indicate smaller modules.

ANALYSIS AND RESULTS

This section elaborates on the metrics we have gathered. To determine deployment frequency metrics, we analysed the frequency of events triggering CI/CD jobs. This involved, firstly, examining the repository's commit history and the logs of CI/CD jobs to understand the frequency of deployments. Secondly, calculating the number of successful deployments recorded in the CI/CD system over a defined period to derive metrics such as deployments per time unit, frequency of code releases, and intervals between deployments.

CODE DEPLOYMENT FREQUENCY

The terms "frequency of code releases" and "deployment frequency" are often used interchangeably in the context of DevOps, and they generally refer to the same concept. Both terms describe how often new code changes are deployed to a production environment. Figure 2 illustrates the deployment frequency from 2016 to 2024. It shows that there have been between 1 to 4 releases per month consistently over this period, highlighting the team's ability to maintain a steady release pattern.

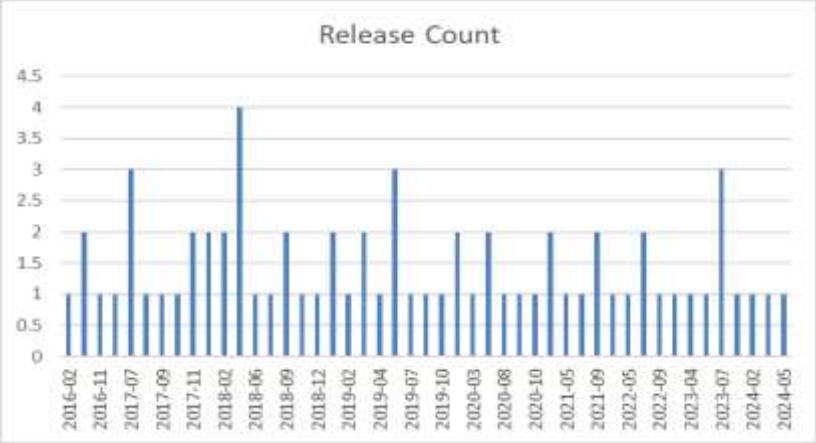

FIGURE 2. Code Deployment Frequency Per Month (2016-2024)

CODE CHURN RATE

Table 1 details the frequency of codebase changes, encompassing additions, modifications, and deletions. Analyzing the codebase changes across different releases of JUnit 5 reveals several patterns and trends in the project's development lifecycle. The following points summarize key observations and trends:

1. Growth in Codebase: Each subsequent release tends to involve significant code additions and deletions, reflecting continuous development and refactoring efforts. For example, between JUnit 5.4.0 and JUnit 5.5.0, there is a noticeable increase in code changes, suggesting major feature additions or improvements.

TABLE 1. Key metrics related to code churn rate

Release Name	File Changes	Code Added	Code Deleted	Commits
JUnit 5.5.2	476865	320376	156489	5756
JUnit 5.5.1	476583	320132	156451	5748

JUnit 5.5.0	476098	319719	156379	5744
JUnit 5.5.0-RC2	475226	319046	156180	5721
JUnit 5.5.0-RC1	473994	318230	155764	5697
JUnit 5.4.2	451828	303117	148711	5508
JUnit 5.5.0-M1	459624	307159	152465	5566
JUnit 5.4.1	451591	302905	148686	5501
JUnit 5.4.0	450838	302329	148509	5473
JUnit 5.4.0-RC2	448962	301030	147932	5427
JUnit 5.4.0-RC1	445219	299274	145945	5386
JUnit 5.4.0-M1	433122	288910	144212	5231
JUnit 5.3.2	387558	261462	126096	4834

2. Stability Before Major Releases: The release candidates (RC) and milestone (M) releases typically show a gradual increase in code stability. For instance, JUnit 5.5.0-RC1 and RC2 show slightly fewer changes compared to the final JUnit 5.5.0 release, indicating a phase of stabilization and bug fixing before the official release.

3. High Frequency of Commits: The number of commits remains relatively high across all releases, indicating an active and ongoing development process. For instance, JUnit 5.5.2 has 5756 commits, reflecting an active contribution cycle with frequent updates and iterations.

4. Consistent Codebase Growth: The consistent addition of code (e.g., JUnit 5.5.2 with 320376 lines added) suggests a robust development effort to enhance functionalities and introduce new features. Correspondingly, the code deletions (156489 lines for JUnit 5.5.2) indicate regular code clean-up and refactoring to maintain code quality.

5. Release Cadence: The release cadence shows a pattern of multiple intermediate versions (RC and M releases) leading to major versions. This structured release approach helps in gradual testing and integration of new features before the final stable release.

6. Significant Changes in Major Versions: Major version updates, such as the transition from JUnit 5.4.x to JUnit 5.5.x, involve more substantial code changes and higher commit counts. This indicates significant feature upgrades and possibly breaking changes that necessitate careful management and extensive testing.

7. Comparison Over Time: Comparing earlier versions like JUnit 5.3.2 with later ones shows an overall increase in codebase size and complexity, highlighting the project's evolution and expansion in terms of features and capabilities.

The analysis indicates a well-managed and active development process for JUnit, with continuous improvements, refactoring, and stabilization efforts.

CODE COMPLEXITY

Table 2 provides an overview of code complexity metrics, specifically highlighting software releases exclusively from the years 2023 to 2024. It seems that the average number of lines of code (Avg. NLOC) remains relatively consistent across different releases, indicating a stable codebase size over time. Similarly, the average cyclomatic complexity number (Avg. CCN) remains constant at 1.3, suggesting consistent code complexity levels across releases. The following points summarize key observations and trends:

TABLE 2. Code complexity metrics

Release Name	Total NLOC	Avg. NLOC	Avg. CCN	Avg. Token	Fun Cnt	Warning Cnt	Fun Rt	NLOC Rt
JUnit 5.5.2	73568	5.8	1.3	44.5	8202	2	0.00	0.00
JUnit 5.5.1	73464	5.8	1.3	44.5	8197	2	0.00	0.00
JUnit 5.5.0	73229	5.8	1.3	44.5	8172	2	0.00	0.00
JUnit 5.5.0-RC2	73177	5.8	1.3	44.5	8169	2	0.00	0.00
JUnit 5.5.0-RC1	73003	5.8	1.3	44.5	8153	2	0.00	0.00
JUnit 5.4.2	68219	5.8	1.3	44.3	7704	2	0.00	0.00

1. Stability in Code Complexity: The stability in cyclomatic complexity values (Avg. CCN) suggests that the overall structural complexity of the codebase has been maintained consistently across releases. This stability is further supported by the similar values of average token count (Avg. Token) across different releases.
2. Function Count and Warning Count: The function count (Fun Cnt) and warning count (Warning Cnt) also remain constant across releases, indicating a consistent number of functions and warnings in the codebase.
3. Release Comparisons: Comparing the latest release (r5.5.2) with older versions (e.g., r5.4.2), there's a noticeable increase in the total number of lines of code (Total NLOC), which is expected with software evolution and feature additions. However, the average code complexity (Avg. CCN) remains unchanged, indicating that despite code growth, efforts have been made to maintain manageable code complexity levels.
4. Rate Metrics: The function rate (Fun Rt) and NLOC rate (NLOC Rt) are consistently low (0.00), suggesting that the ratio of functions to lines of code remains stable across different releases.

RELATIONSHIP OF CODE COMPLEXITY (FUNCTION COUNT) TO CODEBASE CHANGE (CODE COMMITS)

We utilize Pearson correlation analysis to assess the relationship between code complexity (measured by Function Count) and codebase changes (measured by code commits). The calculated correlation coefficient (R) stands at 0.9887, signifying a robust positive correlation. This indicates that as code complexity increases, so does the frequency of code commit, and vice versa. Higher complexity tends to coincide with more frequent changes in the codebase, demonstrating a consistent and predictable pattern. This perfect positive correlation implies a

very strong relationship between the two variables, indicating that managing and understanding both metrics is crucial for maintaining code quality.

DISCUSSION

Upon analyzing these metrics, it's evident that the JUnit 5 codebase has experienced steady growth in size across its releases. However, amidst this expansion, the average function size and complexity have remained consistent, reflecting commendable code quality maintenance. The persistent average cyclomatic complexity number (Avg. CCN) of 1.3 indicates that functions generally maintain simplicity, enhancing both maintainability and readability. Similarly, the steady warning count of 2 suggests that while certain functions may warrant refactoring, overall complexity and quality have been upheld throughout releases.

Employing Pearson correlation analysis unveils a significant relationship between code complexity (measured by Function Count) and codebase changes (measured by code commits). The resulting correlation coefficient (R) of 0.9887 underscores a robust positive correlation, illustrating that heightened complexity aligns with increased code commits, and vice versa. Overall, the project sample has demonstrated adept codebase growth while preserving a consistent level of complexity and function size. This achievement reflects disciplined coding practices and proficient code management strategies (Trautsch et al., 2023), (Chowdhury et al., 2022). Continuous vigilance and maintenance of these metrics within acceptable thresholds will bolster the codebase's longevity and manageability.

CONCLUSION

This article delves into the intersection of DevOps practices and code maintainability, by examining the relationship between frequently used DevOps metrics and key code maintainability metrics, we aimed to provide insights into how these factors collectively influence software maintainability within DevOps projects. The analysis of deployment frequency, code churn rate, and code complexity metrics provides valuable insights into the evolution and maintenance of the JUnit 5 codebase within DevOps practices. Deployment frequency remained consistent, with 1 to 4 releases per month over the analyzed period, showcasing the team's ability to maintain a steady release pattern. The codebase exhibited steady growth with significant additions and deletions between releases, with release candidates and milestone releases indicating a phase of stabilization and bug fixing before major releases. The high frequency of commits across all releases reflects an active and ongoing development process. Additionally, the codebase maintained stable complexity levels across releases, with consistent values in cyclomatic complexity and function count, despite the growth in codebase size. Pearson correlation analysis revealed a strong positive correlation between code complexity and codebase changes, emphasizing the importance of managing both metrics for maintaining code quality. Overall, the analysis demonstrates the project's commitment to maintaining code quality and stability while adapting to the demands of continuous development and deployment within a DevOps environment, underscoring the need for continuous vigilance and maintenance of these metrics within acceptable thresholds for the codebase's longevity and manageability. In order to achieve this, regular monitoring and enhancement of automated testing frameworks are recommended to ensure high test pass percentages. Additionally, conducting regular training sessions on best practices will help maintain consistent code quality. Future research should focus on understanding the impact of code changes on continuous test failures and investigating strategies to improve deployment velocity while maintaining high code quality and stability.

ACKNOWLEDGEMENT

This research was supported by the Talent and Publications Enhancement Research Grant (TAPERG/2023/UMT/2223), titled “Empirical Analysis of Software Maintainability Metrics in DevOps Environments” awarded by Universiti Malaysia Terengganu.

REFERENCES

Almashhadani, M., Mishra, A., Yazici, A., & Younas, M. 2023. Challenges in agile software maintenance for local and global development: An empirical assessment. *Information*, 14(5), 261. https://doi.org/10.3390/info14050261.

Amaro, R., Pereira, R., & Mira, M. 2024. DevOps metrics and KPIs: A multivocal literature review. *ACM Computing Surveys*. https://doi.org/10.1145/3652508.

Arvind. 2022. DevOps lifecycle: Everything you need to know about DevOps lifecycle phases, 42.

Azad, N., & Hyrynsalmi, S. 2023. DevOps critical success factors — A systematic literature review. *Information and Software Technology*, 157, 107150. https://doi.org/10.1016/j.infsof.2023.107150.

Bermón-Angarita, L., Fernandez, A., & Osorio, A. 2023, January 2. A bibliometric analysis of DevOps metrics. *ResearchGate*. https://www.researchgate.net/publication/367622536_A_Bibliometric_Analysis_of_DevOps_Metrics.

Bezemer, C.-P., Eismann, S., Ferme, V., Grohmann, J., Heinrich, R., Jamshidi, P., Shang, W., van Hoorn, A., Villavicencio, M., Walter, J., & Willnecker, F. 2019. How is performance addressed in DevOps? *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 45–50. https://doi.org/10.1145/3297663.3309672.

Chowdhury, S., Holmes, R., Zaidman, A., & Kazman, R. 2022. Revisiting the debate: Are code metrics useful for measuring maintenance effort? *Empirical Software Engineering*, 27(6), 158. https://doi.org/10.1007/s10664-022-10193-8.

Di Biase, M., Rastogi, A., Bruntink, M., & van Deursen, A. 2019. The Delta Maintainability Model: Measuring maintainability of fine-grained code changes. *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, 113–122. https://doi.org/10.1109/TechDebt.2019.00030.

Gasparaite, M., Naudziunaite, K., & Ragaisis, S. 2020. Systematic literature review of DevOps models. *Communications in Computer and Information Science*, 184–198. https://doi.org/10.1007/978-3-030-58793-2_15.

Gasparaite, M., Naudziunaite, K., & Ragaisis, S. 2020. Systematic literature review of DevOps models. *Quality of Information and Communications Technology: 13th International Conference, QUATIC 2020*, 184–198.

Giamattei, L., Guerriero, A., Pietrantuono, R., Russo, S., Malavolta, I., Islam, T., Dînga, M., Koziolek, A., Singh, S., Armbruster, M., Gutierrez-Martinez, J. M., Caro-Alvaro, S., Rodriguez, D., Weber, S., Henss, J., Vogelin, E. F., & Panojo, F. S. 2024. Monitoring tools for DevOps and microservices: A systematic grey literature review. *Journal of Systems and Software*, 208, 111906. https://doi.org/10.1016/j.jss.2023.111906.

Jha, S., Kumar, R., Son, H., Abdel-Basset, M., Priyadarshini, I., Sharma, R., & Long, V. 2019. Deep learning approach for software maintainability metrics prediction. *IEEE Access*, 7, 61840–61855. https://doi.org/10.1109/ACCESS.2019.2913349.

Kudrjavets, G., Nagappan, N., & Rastogi, A. 2022. Do small code changes merge faster? A multi-language empirical investigation. *Proceedings of the 19th International Conference on Mining Software Repositories*, 537–548.

Lomio, F., Codabux, Z., Birtch, D., Hopkins, D., & Taibi, D. 2022. On the benefits of the Accelerate metrics: An industrial survey at Vendasta. *IEEE International Conference on Software Analysis, Evolution, and Reengineering*. https://www.semanticscholar.org/paper/On-the-Benefits-of-the-Accelerate-Metrics%3A-An-at-Lomio-Codabux/1c23ac611dad7509690251d70b24321c1ad9b68d.

Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M., & Lassenius, C. 2019. DevOps in practice: A multiple case study of five companies. *Information and Software Technology*, 114, 217–230. https://doi.org/10.1016/j.infsof.2019.06.010.

Meier, A. 2021. Measuring software delivery performance using the four key metrics of DevOps. *Agile Processes in Software Engineering and Extreme Programming: 22nd International Conference*, 103.

Mishra, A., & Otaiwi, Z. 2020. DevOps and software quality: A systematic mapping. *Computer Science Review*, 38, 100308. https://doi.org/10.1016/j.cosrev.2020.100308.

Nagappan, N., & Ball, T. 2005. Use of relative code churn measures to predict system defect density.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. 2016. DevOps adoption benefits and challenges in practice: A case study. *Product-Focused Software Process Improvement*, 590–597. https://doi.org/10.1007/978-3-319-49094-6_44.

Suescún-Monsalve, E., Pardo-Calvache, C.-J., Rojas-Muñoz, S.-A., & Velásquez-Uribe, A. 2021. DevOps in Industry 4.0: A systematic mapping. *Revista Facultad de Ingeniería*, 30(57), e13314. https://doi.org/10.19053/01211129.v30.n57.2021.13314.

Syed-Mohamad, S. M., A. N., & A.-F. M. A. 2025. Measuring software maintainability: An exploration of metrics and continuous development practices. *Journal of Advanced Research in Applied Sciences and Engineering Technology*, 1–16.

Trautsch, A., Erbel, J., Herbold, S., & Grabowski, J. 2023. What really changes when developers intend to improve their source code: a commit-level study of static metric value and static analysis warning changes. *Empirical Software Engineering*, 28(2). https://doi.org/10.1007/s10664-022-10257-9.

Wickramasinghe, S. 2023, October 27. DevOps & DORA metrics: The complete guide. *Splunk*. https://www.splunk.com/en_us/blog/learn/devops-metrics.html.

Zarour, M. I., Alhammad, N., Alenezi, M., & Alsarayrah, K. 2019. A research on DevOps maturity models. *ResearchGate*. https://doi.org/10.35940/ijrte.C6888.098319.

Zarour, M., Alhammad, N., Alenezi, M., & Alsarayrah, K. 2020. DevOps process model adoption in Saudi Arabia: An empirical study. *Jordanian Journal of Computers and Information Technology (JJCIT)*, 06(03). https://www.ejmanager.com/mnstemps/71/71-1580581874.pdf?t=1726594146.