Journal of Computer, Signal, and System Research

Article

The Impact of Continuous Integration and Continuous Delivery on Software Development Efficiency

Shuai Yang [1,*]

[1] Shandong Mingyao Information Technology Co., Ltd., Qingdao, Shandong, China
[*] Correspondence: Shuai Yang, Shandong Mingyao Information Technology Co., Ltd., Qingdao, Shandong, China

Abstract: This paper explores the impact of Continuous Integration (CI) and Continuous Delivery (CD) on software development efficiency. By examining the core principles, practices, and benefits of CI/CD, the study highlights how these practices contribute to faster development cycles, improved code quality, and enhanced collaboration across development teams. The paper discusses the challenges organizations face when implementing CI/CD, such as tool selection, cultural resistance, and security concerns, and provides practical recommendations for overcoming these obstacles. Additionally, it offers insights into future research directions, including the integration of AI in CI/CD processes, CI/CD implementation in multi-cloud environments, and enhancing security within CI/CD pipelines. The findings underscore the transformative role of CI/CD in modern software development and its potential for driving continuous improvement.

Keywords: Continuous Integration; Continuous Delivery; software development efficiency; DevOps; automation; software quality

Received: 03 March 2025
Revised: 10 March 2025
Accepted: 16 April 2025
Published: 21 April 2025

1. Introduction

In today's fast-paced software development landscape, delivering high-quality software products quickly and efficiently is paramount to maintaining a competitive edge. Among the most influential practices that have emerged to address this challenge are Continuous Integration (CI) and Continuous Delivery (CD). These practices have revolutionized the way software is developed, tested, and deployed, promising faster release cycles and improved software quality.

Continuous Integration (CI) is a development practice where code changes are integrated into a shared repository frequently — often several times a day — followed by automated builds and tests to detect issues early [1]. This approach allows developers to identify integration problems and defects at an early stage, reducing the time spent on debugging and enhancing collaboration among team members.

Continuous Delivery (CD) builds on CI by ensuring that software is always in a deployable state. With CD, the code is automatically deployed to testing or production environments after it passes all necessary tests, ensuring that software can be delivered to users faster and more reliably. This approach minimizes manual intervention, reduces errors, and accelerates the software delivery process.

The primary objective of this paper is to explore how CI and CD practices influence software development efficiency. By examining their impact on various stages of the development process — from coding and testing to deployment and maintenance — we aim to highlight the benefits and challenges associated with adopting CI/CD methodologies. Moreover, this paper will delve into the best practices for implementing CI/CD pipelines and discuss real-world examples of organizations that have successfully integrated these practices into their development workflows.

This paper is structured as follows: Chapter 2 provides an overview of CI and CD, detailing their definitions and core practices. Chapter 3 examines how CI/CD contributes to software development efficiency, focusing on key benefits such as faster development cycles, improved software quality, and enhanced team collaboration. In Chapter 4, we will discuss the challenges organizations face when implementing CI/CD, including technical, cultural, and security-related barriers. Chapter 5 outlines best practices for successfully adopting CI/CD, with a focus on designing effective pipelines and monitoring their performance. Finally, Chapter 6 concludes the paper with a summary of findings and suggestions for future research in the field.

2. Understanding CI/CD

In this chapter, we dive deeper into the principles and practices of Continuous Integration (CI) and Continuous Delivery (CD), examining their core practices, their individual value to software development, and the synergy between the two practices.

2.1. Continuous Integration (CI)

Continuous Integration refers to the practice where developers frequently commit code changes into a shared repository [2]. These changes are automatically integrated, and the system is built and tested continuously. The goal is to detect integration issues early, allowing for faster resolution.

Core Practices:

1. Frequent Commits: Developers commit code to the main repository multiple times a day.
2. Automated Builds: Each code change triggers an automated build to ensure integration consistency.
3. Automated Testing: Automated tests are run to catch defects early in the development process.
4. Version Control Systems (VCS): CI tools integrate with version control systems like Git to track changes.
5. Value to Software Development: CI increases code quality by identifying bugs early, reduces integration issues, and speeds up the development process by fostering collaboration and minimizing manual error.

2.2. Continuous Delivery (CD)

Continuous Delivery extends CI by ensuring that the software is always in a deployable state. After the code is integrated, it undergoes automated testing, and if successful, it can be automatically deployed to staging or production environments without manual intervention [3].

Core Practices:

1. Automated Deployment: Code is automatically deployed to test environments or production after passing necessary tests.
2. Frequent Releases: CD supports frequent releases by ensuring that the software is always ready to be shipped.
3. Continuous Monitoring: Monitoring systems track the performance of the deployed code, ensuring immediate identification of issues.
4. Feedback Loops: Continuous feedback is provided to developers to improve the system iteratively.
5. Value to Software Development: CD enables faster release cycles, higher deployment reliability, and more efficient software delivery by minimizing manual errors and reducing the time between development and deployment.

2.3. Relationship and Synergy between CI and CD

Although Continuous Integration and Continuous Delivery can be implemented separately, they are most effective when used together. CI is the foundation for CD; without frequent integration of code, it becomes difficult to ensure that software is always in a deployable state. CD relies on CI to ensure that code changes are tested and integrated before being deployed, facilitating rapid and reliable software delivery [4].

When CI and CD are combined, they form a comprehensive development pipeline that significantly enhances productivity, reduces lead time, and improves the overall quality of the software product. The close integration between these two practices creates a streamlined development process that minimizes errors, reduces downtime, and improves collaboration across development teams.

3. Impact of CI/CD on Software Development Efficiency

The implementation of Continuous Integration (CI) and Continuous Delivery (CD) has significantly transformed the software development process. These practices enhance efficiency across various stages of development, from coding to testing, deployment, and maintenance. In this chapter, we explore how CI/CD contributes to accelerating development cycles, improving quality and reliability, and optimizing team collaboration [5].

3.1. Accelerating Development Cycles

One of the primary advantages of CI/CD is its ability to speed up the software development and delivery cycles. By integrating code frequently and automating the testing and deployment processes, CI/CD minimizes the manual effort required at each stage and accelerates the feedback loop.

1. Faster Development Time: With CI, developers commit code frequently, and each change is automatically built and tested. This continuous feedback ensures that issues are identified and fixed quickly, rather than accumulating over time. As a result, developers spend less time fixing integration bugs and more time working on new features, leading to faster overall development.
2. Quicker Release Cycles: CD extends the benefits of CI by ensuring that software is always in a deployable state. Once the code is integrated and tested, it can be deployed to production automatically, often within hours or even minutes. This frequent deployment capability significantly reduces the time between development and delivery, allowing organizations to release updates and new features faster than ever before.

By shortening the time between development and deployment, CI/CD enables organizations to respond more swiftly to market demands, customer feedback, and emerging issues, giving them a competitive advantage [5].

3.2. Improving Quality and Reliability

Another key impact of CI/CD is its positive effect on software quality and reliability. Automated testing and continuous feedback are central to ensuring that the software meets high-quality standards throughout the development process [6].

1. Automated Testing: CI relies heavily on automated testing to catch issues early. As developers commit code changes, automated tests run to ensure that the new code does not introduce bugs or break existing functionality. This continuous testing helps maintain a high level of code quality by identifying defects early in the development cycle, reducing the likelihood of critical bugs making it to production.
2. Continuous Feedback: CI/CD pipelines provide developers with rapid feedback on their changes, allowing them to address issues quickly and efficiently. This continuous feedback loop ensures that problems are identified early, minimizing the time and effort spent on debugging and rework.

3. Reliability in Production: With CD, automated deployment processes ensure that the code released to production is tested and stable, reducing the risk of introducing errors in live environments. Additionally, the use of continuous monitoring helps detects issues in production environments quickly, enabling faster recovery and minimizing downtime.

Through automated testing, continuous feedback, and reliable deployment processes, CI/CD enhances both the quality and the reliability of software, ensuring that products are delivered with fewer defects and greater consistency.

3.3. Optimizing Team Collaboration

CI/CD also plays a crucial role in improving collaboration and communication within development teams. By automating the integration and deployment processes, CI/CD allows developers to focus on coding rather than on manual tasks, such as resolving integration conflicts or handling deployment issues. This leads to more efficient teamwork and faster decision-making.

1. Encouraging Collaborative Development: CI/CD fosters a collaborative environment where developers, testers, and operations teams work more closely together. With CI, team members continuously integrate their work into a shared repository, allowing for faster collaboration and issue resolution. This shared responsibility for code quality helps break down silos and encourages communication between team members, leading to better collaboration across the entire development lifecycle [7].
2. Reducing Bottlenecks and Miscommunication: By automating the build, test, and deployment processes, CI/CD reduces the potential for miscommunication or delays that often occur due to manual handoffs or uncoordinated workflows. With a seamless pipeline in place, teams are able to focus on solving problems rather than managing the logistics of software deployment, resulting in a more efficient and streamlined workflow.
3. Fostering a Culture of Continuous Improvement: CI/CD promotes a culture of continuous improvement by encouraging regular code commits, immediate feedback, and frequent releases. This constant iteration allows teams to refine their practices, identify inefficiencies, and adopt new tools or techniques that can improve the development process over time.

By facilitating better communication, fostering collaboration, and streamlining workflows, CI/CD enhances overall team productivity, enabling teams to deliver high-quality software faster and more efficiently.

4. Challenges in Implementing CI/CD

While Continuous Integration (CI) and Continuous Delivery (CD) offer significant benefits to software development, their implementation is not without challenges. From selecting the right tools to overcoming internal resistance and ensuring security, organizations often face several hurdles when adopting these practices. This chapter discusses the main challenges encountered during the implementation of CI/CD and explores potential solutions [8].

4.1. Tools and Infrastructure

One of the most significant challenges in implementing CI/CD is selecting the appropriate tools and setting up the necessary infrastructure. The choice of CI/CD tools depends on the specific needs of the organization, the complexity of the project, and the existing technology stack.

1. Tool Selection: The vast array of CI/CD tools available can make it difficult to choose the right one. Tools like Jenkins, GitLab CI, CircleCI, and Travis CI all

offer different features, capabilities, and integrations, which can lead to confusion when selecting the best option. The chosen tool must integrate seamlessly with version control systems, testing frameworks, and deployment environments.

2. Infrastructure Requirements: CI/CD practices require robust infrastructure to support automated builds, tests, and deployments. This often involves setting up continuous integration servers, configuring cloud environments, and ensuring scalability to handle varying workloads. The cost and complexity of setting up and maintaining this infrastructure can be a barrier for some organizations, especially smaller ones with limited resources.
3. Solution: To address these challenges, organizations should carefully assess their requirements and choose CI/CD tools that align with their existing development workflows. It's also essential to invest in scalable infrastructure and leverage cloud-based solutions that can handle the demands of automation without requiring significant upfront investments.

As seen in Table 1, selecting the right tools and infrastructure setup are some of the key challenges that organizations face when implementing CI/CD.

Table 1. Common Challenges in CI/CD Implementation.

Challenge	Description	Potential Solutions
Tool Selection	Choosing the right CI/CD tool from a variety of available options, based on the organization's needs.	Carefully assess project requirements and existing infrastructure; consider cloud-based tools for scalability.
Infrastructure Setup	Setting up and maintaining the infrastructure needed for CI/CD, such as servers and cloud environments.	Leverage cloud-based solutions for cost efficiency; ensure infrastructure is scalable.
Cultural Resistance	Resistance from team members who are accustomed to traditional development processes.	Provide training, communicate benefits, and gradually implement CI/CD to ease transition.
Security Risks	Ensuring the security of sensitive data and preventing security vulnerabilities in automated processes.	Implement encryption, access control, and integrate security testing tools into the pipeline.
Compliance and Regulation	Meeting regulatory requirements in industries like healthcare or finance during automated deployments.	Automate compliance checks and maintain thorough logs for auditing purposes.

4.2. Team and Cultural Resistance

Implementing CI/CD often requires a shift in mindset and a change in the development process. This cultural shift can be met with resistance from developers, operations teams, or management who may be reluctant to adopt new practices.

1. Resistance to Change: Teams accustomed to traditional development and release practices may be hesitant to embrace CI/CD, fearing that it will disrupt their workflows or introduce additional complexity. Developers may also feel that CI/CD imposes additional pressure to deliver code more frequently, while operations teams may be concerned about the challenges of managing frequent deployments.
2. Overcoming Resistance: Overcoming this resistance requires a clear communication strategy and strong leadership. Management should highlight the bene-

fits of CI/CD, such as faster release cycles, higher code quality, and reduced deployment risks. Providing proper training and support to team members can also help ease the transition. Gradual implementation, starting with smaller projects or teams, can reduce the perceived risk and allow teams to build confidence in the new practices.

3. Solution: Successful implementation of CI/CD often depends on fostering a culture of collaboration and continuous improvement [9]. Teams should be encouraged to share knowledge, collaborate on solving challenges, and celebrate small wins along the way. Leadership should actively support the adoption of CI/CD and be open to feedback from the team.

4.3. Security and Compliance

Incorporating CI/CD into the development process also raises important concerns around security and compliance. As CI/CD automates various stages of development and deployment, it introduces new risks that need to be managed carefully.

1. Security Challenges: One of the main security concerns in CI/CD is ensuring that sensitive data, such as API keys or passwords, is properly protected during automated processes. Additionally, frequent deployments increase the surface area for potential security vulnerabilities, and automated testing may miss certain security flaws that would be caught through manual code review.
2. Compliance Issues: Many organizations, particularly those in highly regulated industries, must comply with strict standards and regulations, such as GDPR or HIPAA. CI/CD processes must be configured to meet these compliance requirements, ensuring that sensitive data is handled appropriately, and that deployment logs are maintained for auditing purposes.
3. Solution: To mitigate security risks, organizations should implement robust security practices, such as encrypting sensitive data, ensuring that only authorized personnel have access to deployment pipelines, and performing regular security audits [10]. Integrating security tools into the CI/CD pipeline, such as static analysis tools and security testing, can help identify vulnerabilities early in the development process. Additionally, organizations must ensure that their CI/CD processes are compliant with relevant standards and regulations by implementing automated compliance checks and keeping thorough records of deployment activities.

5. Best Practices for Successful CI/CD Implementation

Implementing Continuous Integration (CI) and Continuous Delivery (CD) effectively is crucial for realizing the benefits of faster development cycles, improved quality, and efficient deployment [11]. While the technical challenges of CI/CD implementation are significant, adopting best practices can ensure a smoother integration into the development lifecycle. This chapter outlines key best practices for designing a successful CI/CD pipeline and methods for continuously monitoring and optimizing the CI/CD process.

5.1. Designing an Effective CI/CD Pipeline

Designing an effective CI/CD pipeline is critical for ensuring that the automation process integrates smoothly with the development workflow. A well-designed pipeline automates all necessary steps, from code integration to deployment, with minimal manual intervention. Below are some critical steps to designing an efficient CI/CD pipeline:

1. Automate Code Integration: The first step in CI is automating the process of integrating code into a shared repository multiple times a day. Developers should commit small increments of code frequently to avoid conflicts. This allows teams to detect issues early, which can be addressed before they become more significant problems [12].

2. Implement Automated Testing: Automated testing is vital for maintaining code quality in a CI/CD pipeline. It ensures that the new code doesn't break existing functionality. Unit tests, integration tests, and acceptance tests should be automated and run immediately after code is integrated into the repository. This immediate feedback loop helps maintain high-quality code.
3. Continuous Deployment and Delivery: After the code passes automated tests, the next step is to automate the deployment process. Continuous Deployment (CD) ensures that every change that passes automated tests is automatically deployed to the production environment, while Continuous Delivery ensures that changes can be deployed to production with minimal manual intervention.
4. Version Control Integration: Integrating version control systems like Git with your CI/CD pipeline ensures that the right versions of code are deployed consistently. This step facilitates traceability, enabling teams to track changes, revert to previous versions, and understand which changes have been implemented in production.
5. Security Practices: Security must be integrated into the CI/CD pipeline from the beginning. Automate security testing tools (e.g., static code analysis, vulnerability scanners) to identify potential risks before code reaches production. Security practices should be continuously updated in the pipeline to keep pace with evolving threats.

By following these practices, organizations can build a robust and reliable CI/CD pipeline that enhances productivity and code quality.

5.2. Monitoring and Continuous Improvement

The CI/CD pipeline should not be seen as a one-time implementation but as a continuous process that requires constant monitoring and optimization. Tracking the performance of the CI/CD pipeline and analyzing the data generated can provide valuable insights into areas of improvement.

1. Measure Pipeline Efficiency: Key performance indicators (KPIs) such as build success rate, deployment frequency, and lead time for changes are important to track. Analyzing these metrics helps teams identify bottlenecks, inefficiencies, and areas for optimization in the pipeline.
2. Identify and Address Bottlenecks: The pipeline should be regularly reviewed to ensure that no step is slowing down the overall process. Common bottlenecks in CI/CD pipelines include long-running tests, slow integration processes, and manual approval steps. These can be addressed by optimizing processes, increasing parallelization, or automating manual tasks.
3. Incremental Improvement: CI/CD is a process of continuous improvement. Teams should regularly assess the pipeline's performance and implement incremental changes to optimize workflows, reduce cycle times, and improve code quality. This can be achieved through feedback loops that allow for adjustments to be made based on real-time data and developer input.
4. Collaboration and Transparency: The success of CI/CD depends on effective communication and collaboration among developers, operations teams, and other stakeholders. Sharing data, insights, and progress openly with all teams ensures that everyone is aligned and working toward the same goals. Regular retrospectives and feedback sessions can help identify issues and drive improvements.
5. Automation of Monitoring and Alerts: To ensure that the pipeline runs smoothly, automated monitoring and alerting should be set up to catch issues early. This enables teams to respond quickly to failures or performance drops, minimizing downtime and potential impact on the development process.

By continuously monitoring and improving the CI/CD pipeline, organizations can ensure that it remains efficient, scalable, and aligned with their goals. Regular assessments, performance tracking, and ongoing refinement help teams stay agile and responsive to changing needs.

6. Conclusion and Future Directions

In this paper, we have explored the significant impact of Continuous Integration (CI) and Continuous Delivery (CD) on software development efficiency. By adopting CI/CD practices, development teams can streamline their workflows, improve code quality, and accelerate the delivery of software products. This chapter summarizes the key findings from the previous sections, offers practical recommendations for implementing CI/CD, and discusses potential areas for future research.

6.1. Summary of Key Findings

Continuous Integration and Continuous Delivery have become essential components of modern software development practices. The adoption of CI/CD enables organizations to achieve faster development cycles, higher-quality code, and more reliable software releases. Some of the key findings from this study include:

1. Accelerated Development Cycles: CI/CD practices help reduce the time it takes to integrate code and deliver software to production. Automated testing, frequent integration, and continuous deployment reduce manual intervention and the risk of errors, leading to faster and more predictable release cycles.
2. Improved Code Quality and Reliability: Automated testing and continuous feedback are integral parts of the CI/CD pipeline. They ensure that code quality is consistently maintained, errors are detected early, and security vulnerabilities are addressed promptly, contributing to the reliability of the software.
3. Enhanced Collaboration and Communication: CI/CD fosters better collaboration between development and operations teams. By automating repetitive tasks and enabling faster feedback loops, CI/CD enhances communication and cooperation, allowing teams to focus on higher-value work.
4. Challenges in Implementation: Despite the clear benefits, implementing CI/CD is not without challenges. These include selecting the appropriate tools, overcoming cultural resistance, and ensuring security and compliance. However, organizations can overcome these obstacles with careful planning, proper training, and a commitment to continuous improvement.

6.2. Recommendations for CI/CD Implementation

Based on the insights gained from this study, several recommendations can help organizations successfully implement CI/CD practices:

1. Start Small, Scale Gradually: Implementing CI/CD should begin with small, manageable projects. This allows teams to gain experience and refine their processes before scaling them to larger, more complex systems.
2. Focus on Automation: Automating key processes such as testing, deployment, and monitoring is essential for the success of CI/CD. Automation reduces manual errors, accelerates workflows, and ensures consistency across the development lifecycle.
3. Foster a Collaborative Culture: CI/CD works best in an environment where development, operations, and other teams collaborate closely. Building a culture of collaboration and transparency is critical for successfully implementing CI/CD practices.
4. Invest in Continuous Monitoring: Continuous monitoring of the CI/CD pipeline is essential to identify bottlenecks, inefficiencies, and potential issues early.

Teams should track key performance indicators (KPIs) to measure the effectiveness of their CI/CD processes and make data-driven improvements.

5. Prioritize Security and Compliance: As CI/CD pipelines handle frequent code deployments, security and compliance must be integrated into the process. Automated security testing, compliance checks, and proper access control mechanisms should be built into the pipeline to address potential risks.

6.3. Future Research Directions

While CI/CD practices have already made a significant impact on software development, there is still much to explore in this area. Some potential avenues for future research include:

1. AI and Machine Learning in CI/CD: The integration of artificial intelligence (AI) and machine learning (ML) in CI/CD processes could further optimize the pipeline. AI-driven tools for automated testing, error detection, and deployment predictions have the potential to increase the efficiency and accuracy of CI/CD workflows.
2. CI/CD in Multi-Cloud Environments: As organizations increasingly adopt multi-cloud architectures, researching how CI/CD practices can be effectively implemented across multiple cloud platforms is an important area for future study. This research could explore tools, strategies, and best practices for managing CI/CD in complex, distributed environments.
3. CI/CD for Non-Traditional Software Projects: Most current research focuses on CI/CD for traditional software projects. However, there is a need for further investigation into how CI/CD practices can be applied to other domains, such as hardware development, data science, or embedded systems.
4. Improved Security in CI/CD Pipelines: As security threats continue to evolve, further research into how to enhance security within CI/CD pipelines is necessary. Future studies could focus on developing better security testing tools, automating compliance with security regulations, and ensuring the integrity of deployment processes.

6.4. Final Thoughts

Continuous Integration and Continuous Delivery are powerful practices that have transformed the way software is developed and delivered. By enabling faster, more reliable releases and fostering collaboration, CI/CD has a profound impact on software development efficiency. However, organizations must be prepared to overcome challenges such as tool selection, cultural resistance, and security concerns to successfully implement CI/CD. With ongoing advancements and continuous improvement, CI/CD will continue to evolve, providing even greater benefits to development teams and organizations in the future.

References

1. A. S. Mohammed, V. R. Saddi, S. K. Gopal, S. Dhanasekaran, and M. S. Naruka, “AI-driven continuous integration and continuous deployment in software engineering,” in *Proc. 2nd Int. Conf. Disrupt. Technol. (ICDT)*, Mar. 2024, pp. 531–536, doi: 10.1109/ICDT61202.2024.10489475.
2. P. Liang, B. Song, X. Zhan, Z. Chen, and J. Yuan, “Automating the training and deployment of models in MLOps by integrating systems with machine learning,” *arXiv preprint*, arXiv.2405.09819, 2024, doi: 10.48550/arXiv.2405.09819.
3. M. Virmani, “Understanding DevOps & bridging the gap from continuous integration to continuous delivery,” in *Proc. 5th Int. Conf. Innov. Comput. Technol. (INTECH)*, May 2015, pp. 78–82, doi: 10.1109/INTECH.2015.7173368.
4. M. Soni, “End to end automation on cloud with build pipeline: the case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery,” in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Mark. (CCEM)*, Nov. 2015, pp. 85–89, doi: 10.1109/CCEM.2015.29.
5. S. Garg et al., “On continuous integration/continuous delivery for automated deployment of machine learning models using MLOps,” in *Proc. 4th Int. Conf. Artif. Intell. Knowl. Eng. (AIKE)*, Dec. 2021, pp. 25–28, doi: 10.1109/AIKhE52691.2021.00010.

6. S. A. I. B. S. Arachchi and I. Perera, “Continuous integration and continuous delivery pipeline automation for agile software project management,” in *Proc. Moratuwa Eng. Res. Conf. (MERCon)*, May 2018, pp. 156–161, doi: 10.1109/MERCon.2018.8421965.
7. M. Shahin, M. A. Babar, and L. Zhu, “Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.
8. M. R. Pratama and D. S. Kusumo, “Implementation of continuous integration and continuous delivery (CI/CD) on automatic performance testing,” in *Proc. 9th Int. Conf. Inf. Commun. Technol. (ICoICT)*, Aug. 2021, pp. 230–235, doi: 10.1109/ICoICT52021.2021.9527496.
9. S. Garg and S. Garg, “Automated cloud infrastructure, continuous integration and continuous delivery using docker with robust container security,” in *Proc. IEEE Conf. Multimedia Inf. Process. Retrieval (MIPR)*, Mar. 2019, pp. 467–470, doi: 10.1109/MIPR.2019.00094.
10. M. K. A. Abbass, R. I. E. Osman, A. M. H. Mohammed, and M. W. A. Alshaikh, “Adopting continuous integration and continuous delivery for small teams,” in *Proc. Int. Conf. Comput., Control, Electr. Electron. Eng. (ICCCEEE)*, Sep. 2019, pp. 1–4, doi: 10.1109/ICCCEEE46830.2019.9070849.
11. M. L. Gupta, R. Puppala, V. V. Vadapalli, H. Gundu, and C. V. S. S. Karthikeyan, “Continuous integration, delivery and deployment: A systematic review of approaches, tools, challenges and practices,” in *Proc. Int. Conf. Recent Trends AI Enabled Technol.* Cham, Switzerland: Springer, 2024, pp. 76–89, doi: 10.1007/978-3-031-59114-3_7.
12. A. M. Mowad, H. Fawareh, and M. A. Hassan, “Effect of using continuous integration (CI) and continuous delivery (CD) deployment in DevOps to reduce the gap between developer and operation,” in *Proc. Int. Arab Conf. Inf. Technol. (ACIT)*, Nov. 2022, pp. 1–8, doi: 10.1109/ACIT57182.2022.9994139.