International Journal of Core Engineering & Management
Volume-7, Issue-06, 2023 ISSN No: 2348-9510

A REVIEW OF CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY (CI/CD) PRACTICES IN MODERN SOFTWARE DEVELOPMENT

Rajesh Goyal
FS - Insurance
IBM, USA
Glen Mill, Pennsylvania, USA
Rajesh.nim@gmail.com

Abstract

It has already been observed that there has been a slow but sure change in the software development sector recently. Software has become a necessity, and application developers are under pressure to meet this demand by incorporating more automation in their software. Software is increasingly becoming ubiquitous in our society. Continuous integration (CI) and continuous delivery (CD) is a pipelining model that has evolved a lot due to the massive requirements of deployment and deliverability of new features and apps. CI and CD are two main approaches helping significantly to simplify the process of creating software products today and replacing the initial traditional scheme of the Software Development Life Cycle. CI focuses on regular integration of code in a master branch in order to ensure that developers integrate their code changes often and check their code for errors more often. CD, arose as the evolution of CI after the 2010, expands CI adding automated further testing and deployment procedures which accompany the fast and secure release of software. The advantages of CI/CD include; defect identification is achieved in the shortest time, assumptions are minimal hence the quality of the end product is enhanced, and overhead costs are minimized. Nevertheless, there are still some issues to discuss, including the societal change in organizations, the ways of building and deploying different models, the interoperability issue, and the problem of versions in the technology stack. As noted above, the pipeline consisting of software modules that are implemented through the CI/CD cycle involves certain tools including Jenkins, Git, and test automation frameworks. Although there are costs involved in implementing CI/CD, the ultimate benefit derived with regard to time to market and measurability of software health makes CI/CD mandatory especially for organizations that seek to deliver quality software at a faster rate.

Keywords: Continuous Integration, Continuous Delivery, DevOps, Software Development, Automation, Software Quality.

I. INTRODUCTION

The CI can be employed in software engineering to ensure that the product that a software engineer is offering is always in proper state through conducting a set of prescribed activities. Crucial methods like as CI/CD have arisen to fulfil these requirements. The primary goal of CI is to discover and fix integration problems early by regularly integrating code updates. By automating the deployment process, CD guarantees the reliable and rapid delivery of code updates to production[1].

In most cases, a few of critical procedures make up continuous delivery. In one, developers use version control to make incremental changes to either the trunk or mainline, rather than a single large update. Another important part is that automated tests provide quick feedback to the developers within minutes of each modification. When automated tests fail, developers may quickly learn from the experience and address any issues that have been identified. Collectively, these methods are called continuous integration. Upon successful completion of the build and tests, other steps are initiated, including performance testing, human exploratory testing, and complete automated acceptance testing[2].
Modern businesses must contend with security demands, performance scalability, and competitive environments that are always shifting. There has to be a connection between the steady state of operations and the quick release of new features for businesses. Software may be rapidly updated using CI/CD, which allows for secure and stable system operations. Companies in the modern era that see software development as a competitive advantage have a dilemma: how to deploy software at an ever-increasing pace while maintaining system stability and quality. In response to this need, a collection of technological procedures known collectively as CD have gained widespread use[2].
Application delivery lifecycle management is complicated due to the diversity and variety of tools, commercial or open-source technology, and applications. The DevOps methodology provides a framework for tackling such difficult and intimidating problems. DevOps is newer culture; it is the involvement of the development, testing and operation teams to continuously and efficiently deliver outcomes. New entrants to markets can undertake DevOps with confidence and enhance the quality of apps in order to support or even exceed business value, thus enhancing benefit and client satisfaction. It makes things easier for the business to envision opportunities and reduces the time required for clients to review critical bug fixes or potential advancements [3][4].
Some of these improvements include the automation of development significant phases, feedback cycles, and other phases which require minimal human interaction. CI is continually integrated and tested for code that enables the identification of errors performed by the CI team. With Continuous Delivery (CD), continuous deployments are made while the release can be controlled through human intervention. On the other hand, Continuous Deployment automates all procedures and changes could be pushed to the production environment as soon as possible after the tests. This pairing can reduce the costs of software production, increase its quality, and thus ensure its timely release. Below figure 1 depicts the control flow between continuous integration, delivery and deployment listed below:

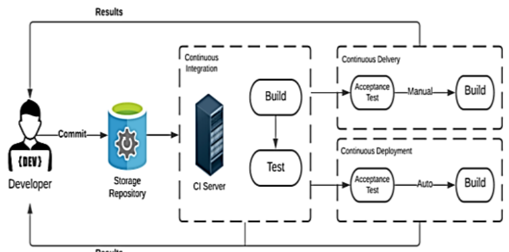

Figure 1: Control flow between Continuous Integration, Delivery and Deployment.

About CI, Continuous Delivery and Continuous Deployment and their work flow. A developer commits code into the repository, the CI server further builds and verifies the code. If successful, a code moves to the CD pipeline, where it undergoes acceptance testing. Continuous Deployment completely automates the process, while Continuous Delivery relies on a human trigger to deploy the build. The developer is continuously updated on the progress of the procedure.

1. Purpose of the Study

This research aims to examine and assess the state of the art in software development processes related to CI and CD. The goal is to learn how these approaches improve software quality and speed up release cycles by automating operations related to integration, testing, and deployment. The study also examines the benefits, challenges, and applications of CI/CD, while highlighting its impact on critical infrastructure and proposing future directions for optimizing CI/CD implementations. The key contribution of the paper is listed below:

- The study reviews key practices and benefits of CI/CD in modern software development.
- It identifies challenges in CI/CD deployment, including organizational culture shifts and version management issues.
- The paper examines tools for version control, automation, testing, and monitoring in CI/CD processes.
- It explores CI/CD applications across diverse development stacks for faster, stable software releases.
- Suggestions for optimizing CI/CD implementation and overcoming deployment challenges are proposed.

2. Structure of the Study

The paper is structured as follows: Section II introduces CI/CD practices, Section III provides an overview, Section IV covers benefits, and Section V discusses deployment challenges. Section VI highlights challenges in critical infrastructure, Section VII examines CI/CD applications, Section VIII offers a literature review, and Section IX concludes with a summary and future research directions.

II. OVERVIEW OF CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY

Current Methods for Developing Software using CI/CD. Various steps in the IT life cycle go into making a system or application that runs continuously, such as planning, designing, developing, testing, fixing bugs, testing again, and finally putting it into production. Developers, testers, and operational units are the three groups participating in these phases.

The concept of CD is a recent innovation. Interest in it began to grow around 2010. "Continuous delivery: Reliable software release through build, test and deployment automation" is the working title of the book which will be published by Humble and Farley. Google search patterns (Figure 2), on the other hand, reveal that the CD technique is gaining popularity [5].

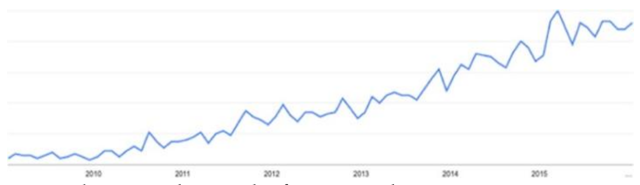

Figure 2: google search tends for searching to continuous delivery

2.1 Continuous Integration (CI)

Continuous Integration pushes engineers to integrate their code into a primary branch of a regular storage as frequently as possible. An engineer will attempt to contribute programming work items to the store a few times on any given day, rather than developing highlights in disconnection and presenting them all at the end of the cycle. The big idea here is to reduce reconciliation expenses by having designers perform it more frequently and sooner changes in the environment[6]. Figure 3 below shows the continuous integration process:

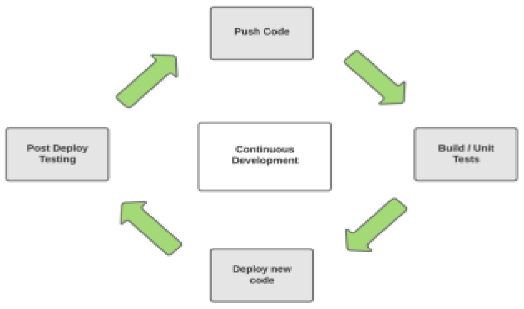

Figure 3: Continuous Integration Process

A goal of CI is to turn integration into a simple, repeatable regular advancement procedure that will help to lower overall form expenses and discover deserts straight away in the cycle. A shift in the improvement group's mentality towards things like availability motivation, sequential and iterative forms, and early defect management is crucial to CI's success[6]. The flow of modules in continuous integration are given below in the figure 4:

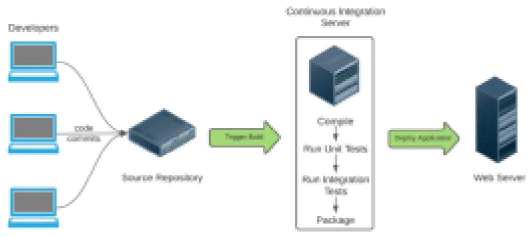

Figure 4. Flow of Modules in Continuous Integration.

2.2 Continuous Delivery (CD)

"Continuous delivery" is often only a few key processes. Throughout one, programmers make little adjustments to the version control system's trunk or mainline branch. A further essential aspect is that automated tests provide developers immediate feedback only minutes after they make a change. Developers can quickly address issues when automated tests fail, thanks to automated testing's rapid feedback and learning capabilities. Continuous integration is the collective name for various methods. Subsequent procedures, including thorough automated acceptance testing, performance testing, and human exploratory testing, are initiated after the build and tests succeed [2]. The stages of continuous delivery shown in Figure 5 are listed below.

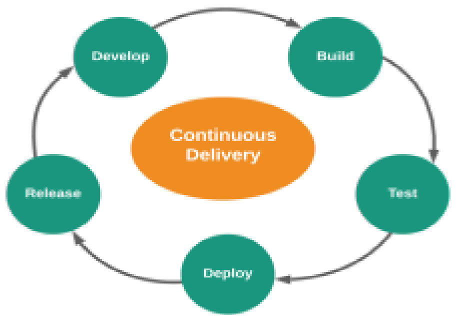

Figure 5: Continuous delivery phases.

III. BENEFITS OF CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY

The advantages of CI and CD will be covered in this section. Here, CI/CD has a few benefits:

1. **Faster Development and Deployment Cycles:**

CI/CD enables automated and frequent testing of code changes, allowing developers to integrate new code continuously. This reduces manual work, leading to faster delivery and shorter release cycles.

2. **Early Detection of Bugs:**

The frequent integration and testing of improvements make it easier to manage issues before they escalate when discovered early in the development cycle.

3. **Improved Software Quality**:

Only code that meets a predetermined quality level is supplied via CI/CD pipelines with automated testing. This results in higher-quality software with fewer defects.

4. **Increased Collaboration and Transparency**:

CI/CD encourages better collaboration between development, testing, and operations teams by providing a clear, real-time view of the development process. This transparency helps in smoother communication and efficient issue resolution.

5. **Reduced Manual Efforts**:

CI/CD practices increase automation of the testing, deployment, and integration processes and minimize the roles that human beings play. This in turn enables the various teams to dedicate their time on creative efforts and on developing new products.

6. **Consistent Deployment Processes:**

CD maintains that every release goes through the same automated process, thus guaranteeing reliability of deployments across the different environments, testing, staging, and production [1].

IV. CHALLENGES OF CI/CD DEPLOYMENT PROCESS

The following are some of the problems that are highlighted in this part about the CI/CD deployment process:

1. **Changes in Organizational Culture:** These types of firms might prefer older solutions and could have a hard time maintaining the continuous integration process. Training of employees would require a complete overhaul of current processes in place. Managers may be cautious because the seemingly unending integration does not contribute towards managers' business agendas (for example, revenue over deliverance) [6].

2. **Divergent Build and Deployment Processes:** Every technology stack may have its build tools, dependencies, and deployment standardized practices; hence, the process is complex and error-prone.

3. **Interoperability Concerns:** To integrate components that are developed in different technologies it is necessary to solve such problems as dependencies, APIs and compatibility to avoid failures after successful completion of development.

4. **Testing and Quality Assurance:** It is necessary to ensure uniformity and stability of testing across different layers of technologies in order to keep the main organic structure intact.

5. **Versioning and Release Management:** Synchronization of versioning and releases is always a challenge when working with different technologies and this requires atomic deployment and sometimes atomic rollback across components [7].

V. CI/CD IMPLEMENTATION IN SOFTWARE DEVELOPMENT

Planning, designing, developing, testing, fixing errors, continuing testing, and implementing in a production environment are all steps in the IT life cycle that contribute to the construction of a continuous system or application. The development team, the testing team, and the operational units all work separately throughout these phases. For developers to prioritize the speed and functionality of the software, the examiner will look at the number of mistakes and faults found. On the other hand, the operational unit will focus on system stability and the minimal chance of failure[8][9]. Figure 6 depicts the deployment management cycle with CI/CD are given below:

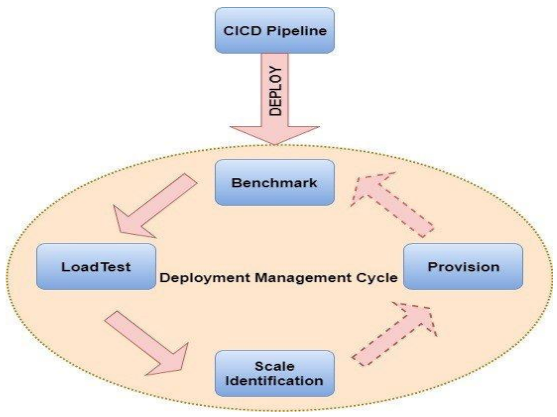

Figure 6: The deployment management cycle with CI/CD

5.1 Tools for CI/CD Implementation

There are many subcategories of CI/CD tools, including those for version control and repository management, building, automation, test automation, and monitoring.

1. **Repository and Version Control Tools:** "Version Management" means keeping track of revisions. The open-source nature and support of distributed architecture have contributed to Git's meteoric rise in popularity as a version management system for CICD workflows.

2. **Build Tools:** Several well-known build tools are open source and include Ant, Maven, and Gradle. According to Shahin, there are a few CI servers that are now in use: Bamboo, Jenkins, Cruise Control, Hudson, Sysphus, Hydra, and TeamCity. CI servers provide centralized control over build tools. However, because of its characteristics like security, cross-platform compatibility, scalability, and customization, Jenkins is utilized as the primary CI server in CI/CD techniques.

3. **Automation Tools:** The main goal of automation is configuration management (CM), which is the process of making sure that infrastructure is consistent. Cost reduction, simplified release processes, easy configuration, reliability, efficiency, and optimal use of resources are some of the benefits of CM. Proper host management is also necessary. The first contemporary open-source CM tool was CF Engine, which came out in 1993.

4. **Test Automation:** Mike Cohn first proposed the "test automation pyramid" as a tool for agile development. The pyr: The middle tier includes the Unit, Service, and UI tiers. AWS has added a new layer to the service test stack, which is termed Performance/Compliance[8].

VI. LITERATURE REVIEW

This section provides a literature review on CI/CD focusing on the Practices in Modern Software Development.

The paper Mowad, Fawareh and Hassan, (2022), examine DevOps and evaluate the methods, tactics, concerns, and procedures that have been found in order to adopt and execute ongoing practices. The case studies we conducted demonstrated the positive effects of CI/CD on software development. The article goes on to describe DevOps as a fresh approach to bridging the gap between the two departments. One potential competitive benefit of using Azure as a DevOps CI/CD platform is the ability to facilitate continuous delivery of software, which allows for quick and frequent releases and allows for rapid reactions to changing client needs. This article also assesses how well CI/CD works to reduce the time and effort needed for software development. The DevOps endeavor is also the focus of their investigation to learn how CI/CD facilitates program delivery, increases flexibility in delivering high-quality software on schedule, and what domains serve as a bridge between the two. The approach used in this article was to investigate and monitor a company-created project that was built utilizing the Azure platform for software development. An assessment and performance of the project are part of the experiment [10].

This paper Pratama and Sulistiyo Kusumo, (2021), intends to carry out the performance test with CI/CD applied. There is still a need for human testers in the current test. Our suggested method for CI/CD performance testing may be run automatically, cutting down on the need for human testers. The performance test becomes integrated, automated, and conducted frequently with the

installation of CI/CD. When the tester changes the values of the parameters, it can react fast enough to stop them from being set in a new test case [11].
In this study Yasmine Ska and Habeebullah Hussaini Syed, (2019), describes the objective of CD as the discovery of methods for the effective, rapid, and dependable delivery of high-quality, useful software. Software engineers using the CI methodology automate builds and tests after merging their work into a shared repository. The primary goals of CI are to improve software quality, speed up the process of detecting and correcting bugs, and decrease the time it takes to validate and distribute new software updates. The goal of continuous integration is to incorporate minor code changes and contributions more often. At the very least once each day, a developer will commit changes to their code. Before pushing to the build server, the developer checks the code repository to make sure the local host and the build server have merged. Now that all of the tests have been executed, the build server decides whether to accept or reject the code change [2].
In this study Soares et al. (2022), Present the findings of the literature review after doing research in six electronic libraries. They find 101 empirical studies that evaluate CI for all software development tasks, including testing, after sorting through 479 publications. They carefully review and extract data pertaining to (i) the CI setting, (ii) results about the impacts of CI, and (iii) the employed technique. They organize and summarize the results using a theme synthesis. Results: Previous studies have looked at the benefits of CI, including increased collaboration, as well as its drawbacks, like increased process and technical difficulties. Six topics emerge from our thematic synthesis: build patterns, problems & defects, software process, quality assurance, integration patterns, and development activities [12].
In this work Shahin, Ali Babar and Zhu, (2017), using the SLR approach to examine the peer-reviewed studies on continuous practices that were released between June 1, 2016, and 2004. They used the theme analysis approach to analyze the information gleaned from evaluating 69 articles that were chosen based on predetermined standards [1].
In this study S.K, Amrathesh and Raju M, (2021), Several software solutions, such as Jenkins, Bitbucket, and TeamCity, have been created to facilitate the continuous integration process. The purpose of this article is to provide a comprehensive overview of CI/CD in software development, including its standard practices, techniques, problems, and methods for implementation using Jenkins [13].
In this study Pruthviraj Deshmukh, (2021), the continuous integration, Delivery and Deployment is the best technique for rapid and continuous improvement. As a result of this strategy, organizations are more likely to provide new and enhanced features to existing ventures or products on a regular basis. This article provides a brief overview of the Continuous Integration, Delivery and Deployment process, as well as the benefits of using this methodology, some of the challenges encountered when using these techniques, and some suggestions for improving these approaches [6].

Table 1: summarizing the related works on critical Infrastructure: Secure Data Management with Edge Computing.

Authors	Focus	Key Contributions	Methodologies	Technologies
[10]	Review and analyze DevOps strategies, methodologies, and processes for adoption and continuous practices.	Benefits of CI/CD in reducing gaps between development and operations, rapid responses to customer requirements.	Case study on a project developed using Azure DevOps tools	Azure DevOps CI/CD
[11]	Implement CI/CD in performance testing to automate and reduce human involvement.	CI/CD improves efficiency by integrating, automating, and periodically executing tests.	Proposed automated CI/CD performance test solution	CI/CD automation for performance tests
[2]	Discuss Continuous Integration (CI) and its role in high-quality software delivery.	Conceptual discussion and overview of CI practices.	CI helps address bugs quickly, improves software quality, and reduces validation time with smaller, regular commits,	CI techniques
[12]	Evaluate the effects of CI on software development activities through empirical studies.	Identified six themes: development activities, software process, quality assurance, integration patterns, issues & defects, and build patterns. Both positive (better cooperation) and negative (technical and process challenges) effects found.	Systematic literature review of 101 empirical studies.	CI environments
[1]	Review continuous practices in software development from 2004 to June 2016	NIST CSF is a comprehensive framework but requires well-organized Planning for implementation in IoT and industrial environments.	Systematic Literature Review (SLR) of 69 peer-reviewed papers using thematic analysis.	Continuous practices
[6]	Overview of Continuous Integration, Delivery, and Deployment (CI/CD) benefits and challenges.	CI/CD allows rapid, continuous improvement and feature enhancement in products.	Overview article.	CI/CD
[13]	a· review on the standard practices, approaches, challenges faced while using the continuous integration/delivery	The continuous integration method makes use of a number of software technologies, such as Bitbucket, Jenkins, and TeamCity.	and using Jenkins for the implantation of continuous integration/delivery	Jenkins, Bitbucket, TeamCity.

VII. CONCLUSION & FUTURE WORK

To improve communication and coordination between operational teams, testers, and developers, contemporary software development has centered on CI and CD. An integration of CI/CD practices reduces the time to detect and resolve defects, ensures higher software quality, and decreases overall development costs. The potential advantages are clear, although difficulties such as cultural transitions, integrating various sets of technologies, and testing and deploying across platforms will have to be overcome. CI/CD helps to improve the software delivery procedures significantly and it is successfully implemented.

In general, going forward CI/CD can be further improved by using some of the emerging technologies like AI that can be used for automating using testing based on predictive models and also using AI for intelligent error detection. Increased complexity of the multi-cloud and hybrid environments will put more pressure to improve interoperability within CI/CD pipelines. Furthermore, improving security in CI/CD processes using predefined vulnerability and compliance sleuthing will play an important role as well. Another research topic will be needed on how the CI/CD practices can be extended to other new areas such as edge computing and IoT and their integration so that smooth and efficient deployments can be achieved across distributed systems.

REFERENCES

1. M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” 2017. doi: 10.1109/ACCESS.2017.2685629.
2. Yasmine Ska and Habeebullah Hussaini Syed, “a Study and Analysis of Continuous Delivery, Continuous Integration in Software Development Environment,” Researchgate, vol. 6, no. September, pp. 96–107, 2019.
3. F. Sendy, T. Kandaga, A. Widjaja, H. Toba, R. Joshua, and J. Narabel, “Continuous Integration and Continuous Delivery Platform Development of Software Engineering and Software Project Management in Higher Education,” J. Tek. Inform. dan Sist. Inf., vol. 7, no. 1, Apr. 2021, doi: 10.28932/jutisi.v7i1.3254.
4. A. P. A. Singh, “STRATEGIC APPROACHES TO MATERIALS DATA COLLECTION AND INVENTORY MANAGEMENT,” Int. J. Bus. Quant. Econ. Appl. Manag. Res., vol. 7, no. 5, pp. 13–19, 2022.
5. M. K. A. Abbass, R. I. E. Osman, A. M. H. Mohammed, and M. W. A. Alshaikh, “Adopting continuous integeration and continuous delivery for small teams,” in Proceedings of the International Conference on Computer, Control, Electrical, and Electronics Engineering 2019, ICCCEEE 2019, 2019. doi: 10.1109/ICCCEEE46830.2019.9070849.
6. Pruthviraj Deshmukh, “IRJET- Continuous Integration, Delivery and Deployment Process in Software Development,” Irjet, vol. 8, no. 6, pp. 1867–1871, 2021.
7. T. Rangnau, R. V. Buijtenen, F. Fransen, and F. Turkmen, “Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines,” in Proceedings - 2020 IEEE 24th International Enterprise Distributed Object Computing Conference, EDOC 2020, 2020. doi: 10.1109/EDOC49727.2020.00026.
8. A. Singh, “A Comparison on Continuous Integration and Continuous Deployment (CI/CD) on Cloud Based on Various Deployment and Testing Strategies,” Int. J. Res. Appl. Sci. Eng. Technol., 2021, doi: 10.22214/ijraset.2021.36038.
9. J. Thomas, “Enhancing Supply Chain Resilience Through Cloud-Based SCM and Advanced Machine Learning : A Case Study of Logistics,” J. Emerg. Technol. Innov. Res., no. September 2021, 2021.
10. 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 Proceedings - 2022 23rd International Arab Conference on Information Technology, ACIT 2022, 2022. doi: 10.1109/ACIT57182.2022.9994139.
11. M. R. Pratama and D. Sulistiyo Kusumo, “Implementation of Continuous Integration and Continuous Delivery (CI/CD) on Automatic Performance Testing,” in 2021 9th International Conference on Information and Communication Technology, ICoICT 2021, 2021. doi: 10.1109/ICoICT52021.2021.9527496.
12. E. Soares, G. Sizilio, J. Santos, D. A. da Costa, and U. Kulesza, “The effects of continuous integration on software development: a systematic literature review,” Empir. Softw. Eng., 2022, doi: 10.1007/s10664-021-10114-1.
13. A. S.K, A. Amrathesh, and D. G. Raju M, “A review on Continuous Integration, Delivery and Deployment using Jenkins,” J. Univ. Shanghai Sci. Technol., vol. 23, no. 06, pp. 919–922, Jun. 2021, doi: 10.51201/JUSST/21/05376.