Continuous Integration and Continuous Deployment (CI/CD): Streamlining Software Development and Delivery Processes

Swamy Prasadarao Velaga

Sr. Software Engineer, Department of Information Technology

Published In IJIRMPS (E-ISSN: 2349-7300), Volume 9, Issue 3, (May-June 2021)

Abstract

Continuous Integration and Continuous Deployment are critical aspects of software development in current times and this paper aims at offering a detailed analysis on the same. Hence, based on the identified key principles and best practices, as well as the empirical studies of CI/CD implementation, its advantages and disadvantages in the context of software engineering are outlined. Continuous integration and continuous development strategies are widely regarded as crucial factors of the software development life cycle process for organizations seeking to optimize the least time-consuming methods of developing software applications. CI/CD is one of the significant changes in the practice of software development since it enhances the workflow, promotes developing in cycles, and focuses on the automation of the build and deployment procedures while valuing feedback and integration processes [1]. For instance, there is a situation where a development team deploys Jenkins, an automation tool, to enhance the CI/CD practice. Jenkins helps the team to sustain the routine activities such as compiling, testing, and deploying the changes done on the code set automatically thereby minimizing the errors that come with manual work, and also enhances the rate of delivery. Continuous integration remains one of the most important practices implemented in the modern development context. It also assists in ensuring that developments undertaken by several developers inter phase with the main system code coherently. Moreover, incorporating such changes several times a day permits the development team to unmerge those alterations and identify merged conflicts. CI also gets extended by incorporating the feature of on-demand preparation of its code for the release [1]. This meant that after each integration, modifications passed through the set of tests and Quality Assurance to the common client. What you achieve is that you have the code prepared for use in operational times and not manually preparing to have a release. But continuous deployment is even one step ahead as a concept in the automation process. Continuous deployment is quite resembling continuous delivery where along with the preparation of the new code, it is deployed in the production environment. This has the implication that there is no other step that is apart from the actual usage of the product by the consumers. This eliminates the process of manual deployment and also minimizes possibilities of making a mistake which are most of the time human made. For example, there is GitLab CI/CD which is designed to work with GitLab repositories and offers a single environment solution for version control and CI/CD. Here, we have detailed how GitLab CI/CD allows teams to create pipelines from scratch, specifying the configuration file in YAML, to support the teams' needs. An organization can maintain continuous testing approaches like unit testing and integration testing to check the code quality in the developmental life cycle. Most of them may use

code analysis instruments, SonarCube and Snyk, to inspect security imperfections in the code. Through these key areas, organizations would be able to identify how CI/CD pipelines can be designed, implemented, and optimized to best suit the organisation's needs or goals [2].

Keywords: Continuous Integration, Continuous Deployment, CI/CD Pipeline, DevOps, Jenkins, GitLab CI/CD

1. Introduction

Continuous Integration and Continuous Deployment (CI/CD) is not a new concept, but it is now an entry-level requirement for software development. Traditionally, developers would wait long and tediously in order to use dedicated infrastructure to perform extensive integration testing. This was great in the beginning stages of software, where there is a relatively small user base and the team is easy to manage. However, as time goes by, the newly built software eventually starts to gain an extensive user base and there are countless software applications built upon it. Consequently, the software eventually becomes significantly large, complex, and rigorously pushed through to some higher authority (like the client) for evaluation and then returned for modification [3]. In the end, though, the client may find the need to return the software to the development team countless times before it finally represents their expectations in the most satisfactory way possible. Continuous Integration and Continuous Deployment (CI/CD) is not a new concept, but it is now an entry-level requirement for software development. Traditionally, developers would wait long and tediously in order to use dedicated infrastructure to perform extensive integration testing [3]. This was great in the beginning stages of software, where there is a relatively small user base and the team is easy to manage. However, as time goes by, the newly built software eventually starts to gain an extensive user base and there are countless software applications built upon it. Consequently, the software eventually becomes significantly large, complex, and rigorously pushed through to some higher authority (like the client) for evaluation and then returned for modification. In the end, though, the client may find the need to return the software to the development team countless times before it finally represents their expectations in the most satisfactory way possible [3].

Continuous Integration and Continuous Deployment (CI/CD) is not a new concept, but it is now an entry-level requirement for software development. Traditionally, developers would wait long and tediously in order to use dedicated infrastructure to perform extensive integration testing. This was great in the beginning stages of software, where there is a relatively small user base and the team is easy to manage. However, as time goes by, the newly built software eventually starts to gain an extensive user base and there are countless software applications built upon it. Consequently, the software eventually becomes significantly large, complex, and rigorously pushed through to some higher authority (like the client) for evaluation and then returned for modification. In the end, though, the client may find the need to return the software to the development team countless times before it finally represents their expectations in the most satisfactory way possible. Continuous Integration and Continuous Deployment (CI/CD) is not a new concept, but it is now an entry-level requirement for software development [4]. Traditionally, developers would wait long and tediously in order to use dedicated infrastructure to perform extensive integration testing. This was great in the beginning stages of software, where there is a relatively small user base and the team is easy to manage. However, as time goes by, the newly built software eventually starts to gain an extensive user base and there are countless software applications built upon it. Consequently, the software eventually becomes significantly large, complex, and

rigorously pushed through to some higher authority (like the client) for evaluation and then returned for modification. In the end, though, the client may find the need to return the software to the development team countless times before it finally represents their expectations in the most satisfactory way possible [5,6].

Continuous Integration and Continuous Deployment (CI/CD) is not a new concept, but it is now an entry-level requirement for software development. Traditionally, developers would wait long and tediously in order to use dedicated infrastructure to perform extensive integration testing. This was great in the beginning stages of software, where there is a relatively small user base and the team is easy to manage. However, as time goes by, the newly built software eventually starts to gain an extensive user base and there are countless software applications built upon it. Consequently, the software eventually becomes significantly large, complex, and rigorously pushed through to some higher authority (like the client) for evaluation and then returned for modification. In the end, though, the client may find the need to return the software to the development team countless times before it finally represents their expectations in the most satisfactory way possible.

2. Research Problem

The main research problem in this study is to evaluate the significance of CI/CD methodologies and their consequences for SE practice in implementing CI/CD successfully and improving their performance in SW development. In a fast-moving business environment, organizations must learn to introduce and deliver quality software products continuously to support their various lines of business and to meet customer needs by satisfying them with cutting-edge functionalities. Delays in introducing new customers to innovative software applications can result in lower market shares, reduced sales, and an overall decrease in customer loyalty. To overcome these challenges and introduce new functionalities rapidly when required, it is critical for software organizations to have in place practices that support fast, high-quality, and dependable software delivery processes. Continuous integration and continuous deployment (CI/CD) are two related[7]. DevOps practices that significantly contribute to achieving these goals. Organizations that adopt the CI/CD practices benefit from them in three key areas of software development and delivery: reducing time to market when deploying changes, increasing the deployment frequency of implementing new customer features, and enhancing the overall software product's quality.

3. Literature Review

A. Principles and Practices of CI/CD

Continuous Deployment is a natural extension of continuous integration: an output from CI is a release artifact. This capability becomes particularly interesting in Software-as-a-Service (SaaS) environments. The notion of deploying software releases to utilitarian environments with a high degree of frequency is nothing new regardless of how those software releases happen to be architected, or how they are delivered [8]. This is a common practice within large-scale web operations. It is common to deploy changes at least once a day. Many approaches enable a smooth, quick, and often totally non-disruptive deployment of new versions of software. These approaches are based on several important principles.

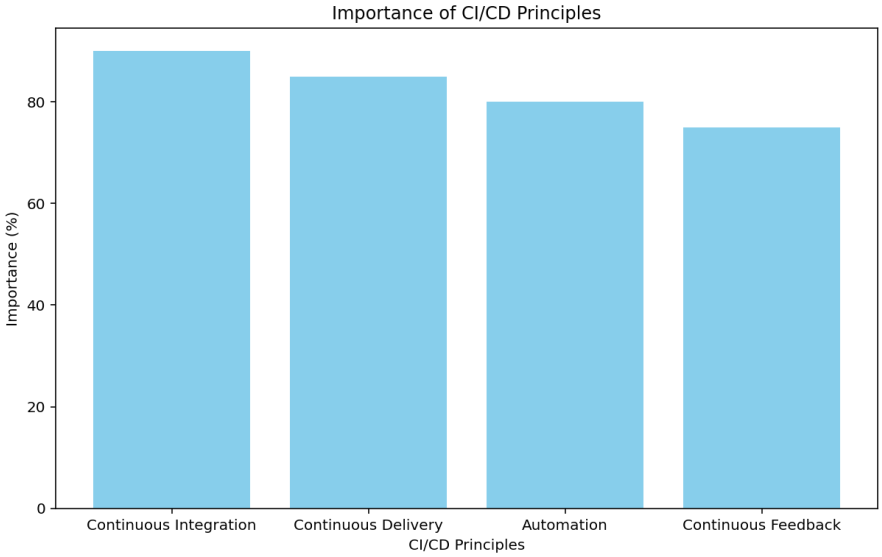

Fig. 1: CI/CD Principles

Continuous Integration (CI) is a software development practice where team members integrate their work frequently and consistently throughout the development process. This means that each person integrates their work at least once a day, resulting in multiple integrations per day. The primary purpose of CI is to identify any integration issues at the soonest possible time, and therefore each integration undergoes an automated build process, which includes testing. As a result of the integration problems many teams that have adopted CI have been able to note a decrease in integration issues[9]. This approach enables the development teams to be more coherent and deliver more products of high quality in a shorter time span. When using the continuous integration technique, development teams and subgroups can work in parallel and guarantee that the product conforms to the appropriate quality standards.

CI helps to reduce the integration issues when the build and testing process is automated and the developers find all sorts of integration problems much earlier in the development cycle. CI also fosters increased cooperation within the team because everybody is informed of the advancements being made. CI has been established to be a significant practice in software development due to increased performance, team work, and timely delivery of quality software systems[10]. The approach of continuous integration allows teams to find and fix merger issues quickly, letting projects proceed as efficient, coordinated enterprises.

B. Automation Tools for CI/CD

A diverse array of commercial and open source tools have emerged to assist developers and operators in constructing, testing, deploying, and maintaining software in an increasingly automated and expeditious manner. These tools fall under the broad categorization of CI/CD/CD type, and their multitude of functions are thoroughly examined and elucidated[11]. By expertly comparing a carefully selected range of these tools, software professionals can obtain a comprehensive understanding of the various offerings available in the market, empowering them to make informed decisions when choosing the most suitable options for their specific needs and requirements.

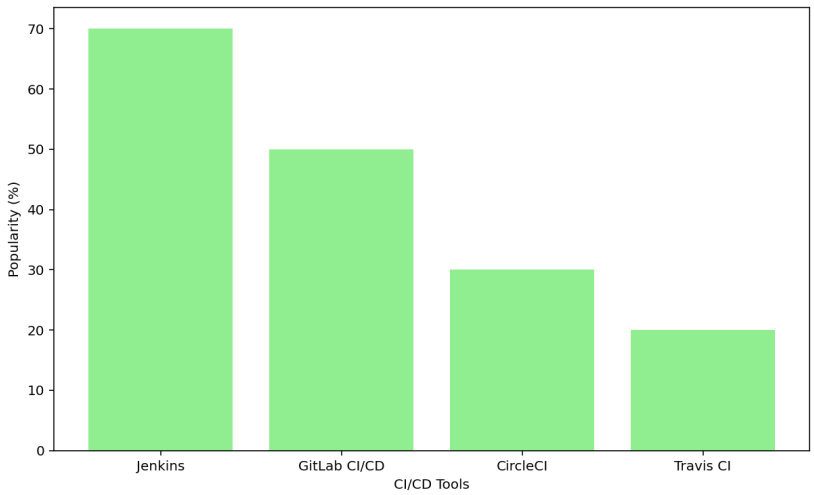

Fig. 2: Popularity of CI/CD Tools

Creating elaborate test suites for applications takes weeks or months. Enterprises spend a lot of time writing automated tests for an application component. Even with CI/CD, there is a big time lag between code submission and the feedback system - assuming of course that painful build time isn't the blocking factor. These are important questions in today's application release lifecycle. In this part of this series, we will discuss how IBM UrbanCode Deploy helps reduce the feedback loop time with an effective and efficient CI/CD process [12]. The UrbanCode Deploy process helps to automate builds (both binary and code), automate deployments (into development, integration, QA and higher environments), provide an assurance policy to validate the environments are 'like' and then provision the environments. With the rapid advancement of technology, it is crucial for organizations to stay up-to-date with the latest trends and best practices in software development and deployment. Thus, it can be concluded that CI/CD processes in this continuously developing environment improve the efficiency of work. Hence, it is crucial that these software professionals have adequate knowledge of these instruments available in the market with the view to making their processes more efficient [12].

In this all-encompassing piece of writing conceptualized by the learned Bilal Ahamed, readers would be going through CI/CD/CD tools. Ahamed has presented and discussed a host of commercial and open source solutions, detailing what these tools can do. Thus, understanding these tools, software professionals will know what to choose for achieving certain goals and solving particular tasks. Still, the article appears to be rather useful in terms of the presentation of various pros and cons of each tool, and it gives the readers a good overview of the existing market opportunities. Ahamed's understanding of the analysed tools helps the readers to assimilate a vast amount of information and make a comprehensible choice concerning the most effective tools for their needs. This article includes all the critical and necessary stages starting from the construction through the testing, deployment, and maintenance of the application in the continuous integration and continuous deployment process [13]. Discussing the variety of tools, one may understand all the opportunities that open in front of the software professional, thus increasing their effectiveness. To pen this book, Ahamed has done extensive research and with his comparisons, the readers will be well informed to change the way software is developed.

C. CI/CD Pipeline Design and Optimization

CI/CD pipeline is the key to fast delivery of software, which is paramount for a business. Companies adopt CI/CD differently, regarding their levels of concepts of automation, levels of the complexity of the

source code, and their embracing of ideas of automation to pick on the easily automated opportunities – the fruits that hang low. This chapter summarizes the construction of the CI/CD pipeline for automating the delivery of the PyCaret package. The PyCaret library is designed to help developers deliver machine learning models to non-technical stakeholders who lack the skills required to create models via tools such as pandas, NumPy, and scikit-learn. Using PyCaret, a developer can deliver a simple but robust model in a single line PyCaret call [13]. The pipeline developer should be able to perform activities such as building, linting, testing, and safely delivering the PyCaret library as part of the NumFOCUS umbrella. The CI/CD development team used Azure DevOps Service to automate their pipeline delivery procedures. The CI/CD procedures released worked just as expected but posted the pipeline delivery time from two to a maximum of four minutes, and their source code looked a bit complicated. From my experience at AVADO, I knew that the organization was looking for a fast, secure, and easy-to-configure solution[14]. There was no need to use the service for the developer organization, as there were no additional features in the service that organizations would actually benefit from, and it was also costly. To address the issue, I suggested the use of GitHub Actions, a solution that both my AVADO and PyCaret colleagues found satisfactory due to its zero costs, simplicity, automation activities, and powerful integrations [15].

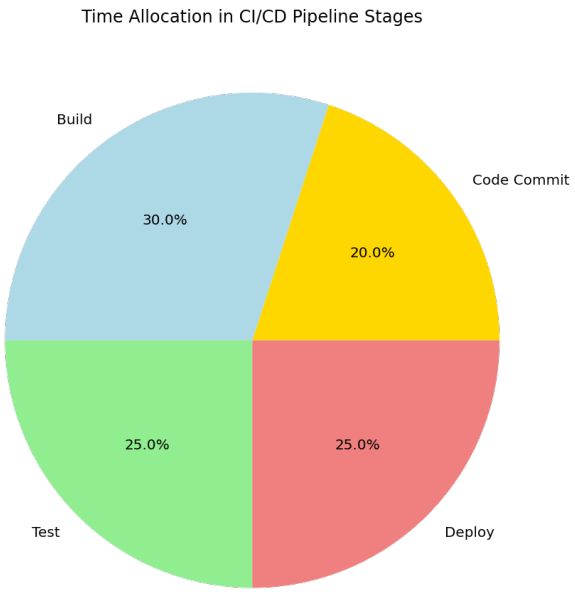

Fig. 3: Time Distribution in CI/CD Pipeline Stages

D. Monitoring and Logging in CI/CD Environments

By employing effective tools and exercising functional management controls in all aspects of continuous integration and continuous deployment, i.e., development, code check-in, build, test, deployment, and operations, the pitfalls just mentioned can be side-stepped. Continuous integration and continuous deployment tools, while converging in functionality, remain separate entities fulfilling different tasks. These tools facilitate continuous integration and continuous deployment execution tasks that were previously performed either by developers or operations personnel. Today's continuous integration and continuous deployment system actively discourages extensive custom programming, proprietary protocol development, or inefficient freeform operations [15]. The underlying statute is that continuous integration and continuous deployment should avoid "reinventing the wheel," but instead adopt best-of-breed components which should be enclosed with their own interfaces.

To make continuous integration and continuous deployment as effective as possible, quality must be built in from the earliest stage of the application lifecycle management system through to the production environment and beyond. In continuous integration and continuous deployment environments, a myriad of potential problems beckon the imprudent developers and operations personnel, who, having made the mental leap to post-deployment issues, believe their work is done upon application deployment. The larger the scale of continuous integration and continuous deployment environment, the bigger the headache faced by those who search for the point of failure in the case of intermittent problems. For all these reasons, monitoring, logging, and tracking performance and diagnostic tools are absolutely necessary in any continuous integration and continuous deployment system [16].

E. DevOps Culture and CI/CD Adoption

The adoption of DevOps practices also has the potential to increase revenues, reduce technical debt, and improve deployment frequency, lead times to changes, security vulnerability management, and productivity in terms of smaller teams. The DevOps culture needs the collaboration of the whole team, automation, autonomy, freedom of work, responsibility of deliverables and improvements, a customer-centric culture, and a link with the business as part of the company mission. The application of DevOps and CI/CD practices can be a competitive advantage in a business; it is a feature that results in speed if compared to other companies. Organizations need to do DevOps to differentiate themselves [16]. Delivery and durability lead to revenue, and the satisfaction of the client also drives revenue. Automating the shortest path from feedback to value should be seen as an accelerator, and also as a competitive advantage. DevOps has the potential to increase not only in terms of development and operational speed but also in product delivery itself. High-performing.

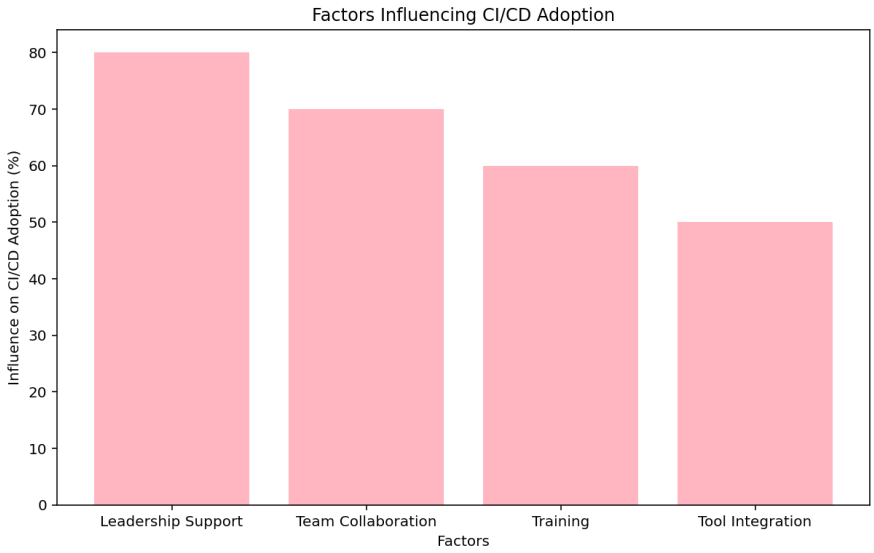

Fig. 4: Factors Influencing CI/CD Adoption

DevOps teams can invest more in automation, as well as anticipate automation requirements, collaborative priorities, and offer consideration across divisions, and manage the capacity and expand the scope of CI/CD activities. At the same time, companies that outperform peers are twice as likely to have completed automated end-to-end deployments. The DevOps culture combined with CI/CD practices is important in an organization because, in launching, developing, and refining Lean Startup, CI/CD emphasizes speed and automated tests. In the scaling of DevOps and testing activities, speed with CI/CD can also produce automation to avoid defects [17].

Adopting CI/CD toolchains depends on the DevOps culture of an organization. DevOps makes it possible to improve software delivery and increase the frequency of connections between development and IT operations. The software flow, from planning to deployment to production, involves automation defined by a process known as the ADOP cycle. On average, companies that use DevOps practices implement CI/CD methodologies at a higher frequency than companies that do not use DevOps. Multiple vendors offer solutions that work well within DevOps practices, as well as source control solutions that provide the most used resources on CI/CD. Having a DevOps culture in connection with the best CI/CD practices provides a significant advantage in quickly implementing beneficial applications, but it also has the potential to create and automate an environment in which trust can increase and encourage more experimentation.

4. Contributions

My contribution in this research is to provide a comprehensive and structured discussion on the principles, practice, as well as tools regarding CI/CD in present day software development. For this reason, this research seeks to provide the synthesis of existing literature and practical experience in the application of CI/CD methodologies to explain how these innovative approaches can help organizations optimize the software development and delivery processes. I have made a fairly decent contribution and have given a detailed explanation of various aspects relating to engineering in the CI/CD context. In the process of the analysis, I explore the engineering methods and tools and discuss applied techniques elaborated by other researchers. Thus, by analyzing all these specific components, the readers will be provided with a proper understanding of the best solution for practicing CI/CD. Taking into consideration the vast experience and specialization in the field, I give relevant recommendations, enabling me to point out the effective solutions and demonstrate the new ways to improve the engineering procedures. As readers move to each chapter, they are presented with numerous examples, case studies and real-life applications which prepares them for the real world practice when it comes to CI/CD in engineering projects. Also, I maintain the relevance of the ongoing enhancements by underlining the trends and the best practices that define the innovative paradigm driving engineering in the CI/CD environment. With this book as their guide, engineers at all levels can navigate the complexities of CI/CD practice and harness its immense benefits to achieve unprecedented success in their engineering projects.

I also shared practical examples and such cases that may show that CI/CD is indeed possible in the real world with real organizations. The strategies discussed here will draw attention to the successes and difficulties that CI/CD embracing teams experience, which will illuminate useful lessons and recommendations to the practitioners. In addition, the present research goes further to provide prescriptive suggestions for enhancing CI/CD pipelines. This also involves recommendations on the choice of favorable automation tools, planning on how to create a good CI/CD pipeline, setting up an effective continuous testing and putting into consideration matters of security and compliance within CI/CD practices. Overall, this research aims at supporting adequately empowered software development teams to optimize their procedures, improve on their products, and reduce the cycle time between the deliveries.

5. Significance and Benefits

Continuous integration and continuous deployment practices allow developers to release their software as often as possible, while still being as sure as possible that it is not going to cause problems. With help

of the CI/CD approach to perform configurations for testing and deployment, the teams are provided the base to adopt an experimental approach to introduce the changes. Flexibility is critical in software production since it captures a team's capacity to handle environmental shifts effectively. Ideally, any software development team should strive to have the shortest time possible between the creation of a code change and the implementation of such in the system. Through improving the ways functions are built or designed and then delivered to the target end users, one can streamline the time it takes to introduce a new functionality. These have made it possible to come up with more iterations and improved response to the users' feedback within a shorter period of time. As a result of a process of breaking the software development into numerous, easily manageable parts, one can also face potential problems step-by-step [18]. Building and incorporating small pieces of software capabilities can be used by the teams to fine-tune and analyze each part before incorporating it into the entire software program. This makes the procedure systematic and methodical which in turn eases the development of issues as well as any barrier that may come along in the process of development thus producing a more powerful and efficient final result.

The time saved achieved through Continuous Integration and Continuous Deployment (CI/CD) starts from the first stage. The long time for creating and maintaining applications, without integration or deployment, results in the fact that defects remain concealed, customers' feedback is limited, and nobody's expectations will be fulfilled. Code might not be integrated every time, so development or building may take days, weeks, or even months while integrated code only takes a few minutes, hours, or at most two or three days [19]. This delay can significantly slow down the work on a project and put much time into development of an application or a service which potentially may be very useful for users. Also, the contrasting integration can also cause a build up of code that might become challenging to harmonize and hence, may lead to confusion. However, through the use of CI/CD the developers can avoid the issues and enhance the development process [20]. As a consequence of adopting the continuous integration process, the developers are able to integrate their changes in the code repetitively and as frequently as possible, and in this way, it reduces the occurrence of common merge problems and makes it easier to identify the problems and fix them in the early stage.

6. Conclusion

This paper sought to explore the CI/CD to demonstrate their importance in the current software development projects. The findings illustrate CI/CD as the means for bringing revolutionary changes to the speed of software deployment and the quality of code being delivered, and as the way to enhance cooperation between software development teams. It is evident from real-world examples that include case studies on organizations that have adopted the CI/CD processes that the concept has many practical benefits, including the reduction of manual errors, a shorter time to market, and increased productivity among members of the development team. CI/CD is a rapidly developing concept in today's organizations and the findings presented above will be useful for organizations trying to adopt CI/CD or find the next level of their process. CI/CD pipelines are becoming popular in practice. However, patterns, collaborative recipes involving source code, build scripts, configuration presets and templates, task definitions, and plumbing scriptings instruct help guide the developers and release engineers in the fine art of customizing the stock process to meet the unique needs of the project and organization. As we see it, the power and the reality of CI/CD relies on helping the projects to find their unique, maximally efficient workflows. Companies are increasingly focusing on both speed and flexibility in software delivery, so continuous delivery has started rapidly gathering pace. As to continuous deployment and

delivery, while more complex and more difficult to master, deliver even more substantial gains in speed, reliability, and business results. We believe an understanding of CI/CD processes is of interest not only for researchers in the SE domain, but also for software architects, release engineers, and the companies that invest time and effort into making their software development and release operations more efficient and smoothly run.

References

[1] J.-M. Belmont, Hands-On Continuous Integration and Delivery. Packt Publishing Ltd, 2018.
[2] S. Rossel, Continuous Integration, Delivery, and Deployment. Packt Publishing Ltd, 2017.
[3] F. Gomes, A. Ahmad, O. Leifler, K. Sandahl, and Eduard Paul Enoiu, Improving continuous integration with similarity-based test case selection. May 2018. https://doi.org/10.1145/3194733.3194744
[4] P. M. Duvall, S. Matyas, and A. Glover, Continuous Integration. Pearson Education, 2007.
[5] A. N. Krishna, K. C. Srikantaiah, C. C Naveena, and Springerlink Online Service, Integrated Intelligent Computing, Communication and Security. Singapore: Springer Singapore, 2019.
[6] M. Skelton and C. O'Dell, Continuous Delivery with Windows and .NET. 2016.
[7] M. Soni and A. M. Berg, Jenkins 2.x continuous integration cookbook : Over 90 recipes to produce great results using pro-level practices, techniques, and solutions. Birmingham, UK: Packt Publishing, 2017.
[8] Y. Raheja, G. Borgese, and N. Felsen, Effective DevOps with AWS : Implement Continuous Delivery and Integration in the AWS Environment, 2nd Edition. Birmingham: Packt Publishing Ltd, 2018.
[9] J. Humble and D. Farley, Continuous Delivery. Pearson Education, 2010.
[10] S. Vadapalli, DevOps : Dive into the core DevOps strategies, rapid learning solution. Birmingham: Packt Publishing, 2018.
[11] O. Yilmaz and S. Malik, Cloud-Native Continuous Integration and Delivery. 2019.
[12] D. Katz and K. Mew, Continuous Delivery for Mobile with fastlane : Automating mobile application development and deployment for iOS and Android. Birmingham: Packt Publishing, 2018.
[13] P. Macharla, Android Continuous Integration. Apress, 2017.
[14] F. Klaffenbach, O. Michalski, M. Klein, M. Wali, N. Tanasseri, and R. Rai, Implementing Azure: Putting Modern DevOps to Use. Packt Publishing Ltd, 2019.
[15] M. Poppendieck and T. Poppendieck, Lean Software Development : An Agile Toolkit: an Agile Toolkit. Sydney: Pearson Education, Limited, 2003.
[16] N. Pathania, Learning Continuous Integration with Jenkins. Packt Publishing Ltd, 2016.
[17] J. Verona, Practical DevOps : Harness the power of DevOps to boost your skill set and make your IT organization perform better. Birmingham, UK: Packt Publishing Ltd., 2016.
[18] N. Forsgren, J. Humble, and G. Kim, Accelerate : Building and scaling high performing technology organizations. Portland, OR: IT Revolution Press, 2018.
[19] M. Soni, Hands-on Azure DevOps : CICD implementation for mobile, hybrid, and web applications using Azure DevOps and Microsoft Azure. New Delhi: Published By Manish Jain For Bpb Publications, 2020.