Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

ADOPTION OF CONTINUOUS DELIVERY IN DEVOPS: FUTURE CHALLENGES

MARIA AFZAL
Department of Software Engineering, Superior University, Lahore, Pakistan.
Email: maria.afzal202@gmail.com

USMAN HAMEED
Department of Computer Science, Superior University, Lahore, Pakistan.
Email: usman.hameed4477@gmail.com

SALEEM ZUBAIR AHMED
Department of Software Engineering, Superior University, Lahore, Pakistan.
Email: saleem.zubair@supeior.edu.pk

MUHAMMAD WASEEM IQBAL
Department of Software Engineering, Superior University, Lahore, Pakistan.
Email: waseem.iqbal@superior.edu.pk

SABA ARIF
Department of Computer Science, Superior University, Lahore, Pakistan.
Email: saba.arif@superior.edu.pk

USAMA HASEEB
Department of Software Engineering, Superior University, Lahore, Pakistan.
Email: mspm-s22-005@superior.edu.pk

Abstract

Product may be released more frequently and consistently with continuous delivery, requiring less- Continuous delivery (CD), a crucial component of DevOps, will remain a significant challenge in future manual work and give developers more assurance that the software is of high quality. Additionally, it necessitates the use of more advanced configuration management and deployment technologies, as well as a greater emphasis on automation. Additionally, continuous delivery necessitates a change in company culture and philosophy, which some firms may find challenging to implement. Additionally, organizations will need more advanced monitoring and logging systems to make sure the software is functioning properly and that any problems can be found promptly because of the complexity and rapidity of software development. Careful preparation and the application of the appropriate DevOps techniques and tools can be used to overcome challenges. Investment in training and the acquisition of DevOps specialists with continuous delivery experience can both be advantageous for organizations.

Keywords: Devops; Continuous Delivery; Adoption; Future Challenges.

1. INTRODUCTION

A crucial DevOps technology, continuous delivery helps developers to deploy code more quickly and with fewer failures. It makes it possible for teams to create, test, and deploy software fast and effectively. Teams can test new features fast and roll them out to customers thanks to continuous delivery. Nevertheless, there are several difficulties with continuous delivery in DevOps. The demand for automation is one of the most difficult issues. Although automation is necessary for effective continuous delivery, it can be challenging to set up and manage. The requirement for accurate testing and

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

monitoring presents another difficulty. It is challenging to guarantee that the code is of great quality before it is delivered without trustworthy testing and monitoring [1]. A DevOps technique called continuous delivery (CD) enables businesses to swiftly release software updates to their clients. It entails automating the development, testing, and deployment of programs to enable on-demand software release to production. Although CD has completely changed how software is produced and distributed, it also presents a unique set of difficulties. Addressing the complexities of software deployments is the first difficulty. Various teams and services must be coordinated as part of CD, which can be challenging to handle [2]. In addition, managing numerous environments, like development, production, and staging, can make the software release process complicated. Due to its intricacy, delivery times may be prolonged and errors are more likely to occur. Making sure applications are safe and adhere to industry standards is the second problem [3]. To guarantee that their apps are safe and consistent with industry standards, CD mandates that businesses constantly review and update them. As firms must be capable of swiftly discovering security flaws and fixing them before they are exploited, this can be a challenging process. Scalability is the third difficulty. Organizations must be able to expand their delivery method to satisfy client expectations to implement CD. Implementing this can be challenging, particularly for large firms with intricate systems. Finally, continuous integration is necessary for continuous delivery [4]. To create, test, and deploy apps, several developers will need to collaborate, which can be challenging to organize. Additionally, Delivery Pipeline is thought of as the final link in the production chain for the program that gathers parts of the finished product from various packages of potentially susceptible software build, testing, and staging. The Delivery Pipeline is seen in Fig. 1 and begins with the Staging step following Code Integration. The item will next be put to the test through an automated test [5, 6]. The code will be made public once the artifact has successfully undergone the Automated Test procedure to get user feedback.

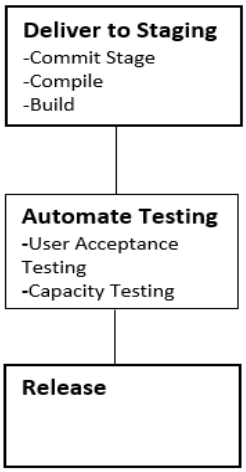

Figure 1: Pipeline of delivery

A combination of skills called continuous delivery enables us to swiftly, safely, and sustainably introduce new features, configuration changes, bug fixes, and experiments

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

into production or the hands of customers [7]. Continuous delivery is based on these fundamental ideas:

Integrate quality. In continuous delivery, we invest in creating a culture that is supported by technologies and people so that we can identify issues as soon as they arise and repair them as soon as they are affordable to identify and fix.

Work in small chunks. When creating new services or products or making investments in organizational change, organizations frequently plan their work in large chunks. We get crucial feedback on the tasks we are doing so that we may make course corrections by breaking work up into so much smaller pieces that provide measurable business outcomes rapidly for a tiny portion of our target market [8]. Working in tiny batches adds some overhead, but it has huge benefits since it helps us avoid tasks that have no negative value for our businesses. Improving the finances of the software delivery process so that the cost of pushing out individual updates is very low is one of the main objectives of continuous delivery.

Computers perform repetitive tasks; people solve problems. Software deployments and regression testing are two examples of repetitive tasks that require a lot of time and effort that can be reduced by investing in automation and simplification. As a result, we free up personnel for higher-value problem-solving tasks, such as responding to criticism by redesigning our systems and procedures.

Relentlessly pursue continuous improvement. The most crucial quality of high-performing organizations is that they're never content and constantly try to improve [9]. High performers incorporate improvement into every employee's everyday tasks.

Here are some ways that high-performing organizations achieve this:

1. Encouraging a culture of continuous improvement: High-performing organizations foster a culture where everyone is encouraged to identify and act on opportunities for improvement. This can involve creating forums for employees to share ideas and feedback, as well as recognizing and rewarding employees who contribute to continuous improvement.
2. Prioritizing quality: Quality is a key aspect of continuous delivery. High-performing organizations prioritize quality by incorporating automated testing, code reviews, and other quality assurance measures into their development processes. This ensures that bugs and issues are caught early, reducing the risk of delays and downtime.
3. Empowering teams: High-performing organizations allow their teams to take proprietorship of their work and make decisions that impact the product. This includes providing teams with the tools, resources, and training they need to be successful. Empowered teams are more likely to take responsibility for delivering high-quality code and identifying opportunities for improvement.
4. Providing regular feedback: Continuous delivery relies on frequent feedback to identify and address issues quickly. High-performing organizations provide regular

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

feedback to their teams, including metrics and performance data. This allows teams to monitor their progress and identify areas where they can improve.

5. Embracing experimentation: High-performing organizations are willing to experiment with new ideas and approaches. This can involve running small-scale experiments to test new features or changes, or conducting A/B testing to compare the effectiveness of different strategies. By embracing experimentation, organizations can quickly identify what works and what doesn't, and make changes accordingly.

Overall, high-performing organizations that use continuous delivery incorporate improvement into every employee's everyday tasks by fostering a culture of continuous improvement, prioritizing quality, empowering teams, providing regular feedback, and embracing experimentation.

Everyone is responsible. Teams in bureaucratic companies frequently prioritize departmental goals over organizational objectives. Development thus emphasizes throughput, testing emphasizes quality, and operations emphasize stability [10]. These are all system-level objectives, though, and they can only be accomplished through tight cooperation amongst all parties who are involved in the delivery process. Making these system-level results transparent, engaging with the rest of the company to set quantifiable, doable, and time-bound goals for these results, and then assisting their teams in achieving those goals are essential management objectives.

According to our research, changes in CD had a positive impact on how rewarding work felt [11, 7]. This indicates that investing in technology also involves investing in people and that doing so will increase the sustainability of our technological process (figure 2).

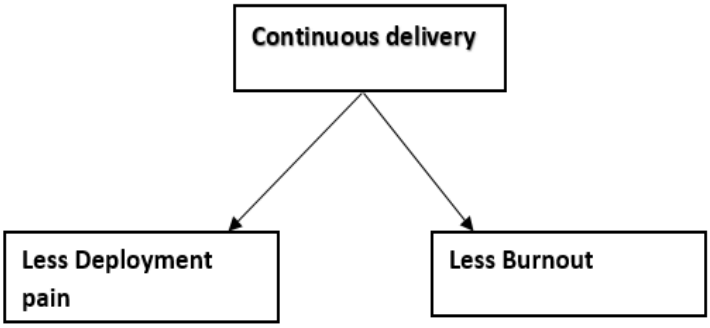

Figure 2: Work becomes more sustainable through continuous delivery

2. LITERATURE REVIEW

A summary of the extant literature is provided in this section. The difficulties that an IT organization has implementing DevOps principles have been extensively studied. We discovered that the organization's issues with adopting CD are plugins and CI. Additionally, they noted that firms may have difficulties in acquiring the necessary knowledge and abilities to embrace DevOps principles. The implementation of continuous delivery is hampered by bad infrastructure, manual testing, and resistance to

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

change; it has been found [12]. Because of tool restrictions, continuous delivery is halted. The adoption of current tools raises security issues, and they also provide an insufficient testing response.

Continuous delivery is one of the DevOps techniques that must be used, which necessitates changes in corporate culture, roles, and responsibilities surrounding the delivery process. It might be difficult to develop appropriate procedures for various firms; thus, interviews should be done to identify the advantages and difficulties. Many writers conducted an SLR and discovered that the main obstacles to adopting continuous delivery under DevOps principles are the design phase, testing, and integration tools [13]. They propose that when firms transition to DevOps and CD, they must alter their strategy. Another study found that moving forward into continuous delivery while dealing with complicated infrastructure might be quite difficult. A semi-structured technique was used by some authors to identify security threats to a continuous delivery pipeline. They draw attention to the danger posed by the internet by snatching or hacking data from the CD pipeline [14]. To meet this difficulty, they offered container solutions like Docker. On rapid delivery and software testing, several writers conducted a case study and semi-systematic review of the literature. They claimed that poor test coverage performance, time restrictions, and customer satisfaction are obstacles to testing and CD. The majority of developers, according to a survey conducted by the author, are unaware of important industry threats that the continuous delivery pipeline mandates in development [15]. Continuous delivery is complicated by a large number of infrastructure systems and quantities of resources. In this article, the author discusses the value of collaboration and communication between the development and operations teams when it comes to software quality assurance. He said that DevOps offers the best testing and delivery options. When doing testing, DevOps concepts and features use different methods than those used for ordinary testing. Delivery is not late because of the prioritization of tests and the usage of analytics [16]. Organizations may overcome the difficulty of adopting DevOps with the aid of testing groups. Testing teams can set up an environment for continuous testing to support automated deployment, delivery, and monitoring. This facilitates immediate input on the software's quality. In agile, continuous testing is carried out in the early phases, while in DevOps, testing is a continual activity. It provides teams with precise and lucid test results. More systematic research in the academic, scientific literature is needed to support DevOps testing efforts [17]. DevOps presents several ways to view testing. A close link between the development and operations teams may be formed through DevOps. Numerous test method DevOps organization approaches have been discussed by the author, including feature toggles, infrastructural testing, removing, aggressive testing, etc.

Feature toggles, also known as feature flags or feature switches, are a technique used in software development to turn on or off specific features or functionality in an application. Feature toggles can be used in conjunction with continuous delivery to allow for the continuous deployment of code to production while also controlling which features are visible to end users [18].

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

Here are some ways that feature toggles can be used in continuous delivery:

1. Gradual rollout: Feature toggles can be used to gradually roll out new features to a subset of users, allowing for testing and monitoring of the feature before releasing it to all users.
2. A/B testing: Feature toggles can also be used for A/B testing, where two or more versions of a feature are released to different user groups to determine which version performs better.
3. Canary releases: Feature toggles can be used for canary releases, where a new version of an application is released to a small group of users to ensure that it is stable before rolling it out to all users.
4. Emergency rollbacks: In the event of a problem with a new feature or release, feature toggles can be used to quickly disable the feature or roll back to a previous version of the application.

Overall, feature toggles provide flexibility and control in continuous delivery by allowing developers to release new features to production gradually and monitor their impact before releasing them to all users.

Infrastructural testing is an essential part of continuous delivery (CD) because it ensures that the system infrastructure is functioning correctly before the new code is deployed [19]. In CD, the software development process is automated, and changes to the code are frequently released to production [20]. The goal of infrastructural testing is to detect any potential issues in the infrastructure that could cause problems for the application once it is deployed.

Some common types of infrastructural testing in CD include:

1. Configuration testing: This involves testing the configuration of the infrastructure, including hardware, software, and network settings, to ensure that it is correctly set up to support the application.
2. Integration testing: This involves testing the interaction between different components of the infrastructure, such as databases, servers, and applications, to ensure that they are working together correctly.
3. Performance testing: This involves testing the performance of the infrastructure under different levels of load and stress, to ensure that it can handle the expected traffic levels and workloads.
4. Security testing: This involves testing the security of the infrastructure to identify potential vulnerabilities that could be exploited by attackers.

In addition to these types of testing, it is also essential to monitor the infrastructure continuously to identify any issues that arise after the application is deployed [21]. This can include monitoring server logs, application logs, and other metrics to detect any anomalies or errors that could impact the performance or security of the application.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

Overall, infrastructural testing is critical for ensuring the reliability, performance, and security of the infrastructure that supports continuous delivery. By detecting and resolving issues early in the development process, teams can improve the quality of their applications and reduce the risk of downtime or other issues in production.

Removing: In Continuous Delivery, removing refers to the process of deleting or disabling a feature, component, or code from a software application or system.

Removing is an important part of Continuous Delivery because it helps ensure that the software application or system is always in a releasable state [20]. By removing any unnecessary or outdated features, components, or code, the development team can reduce the risk of introducing new bugs or other issues into the system.

To implement removing as part of Continuous Delivery, the development team should have a clearly defined process in place for removing code or features. This process should include proper testing to ensure that the removal does not adversely impact the application or system [22]. Additionally, the team should have a plan for a rollback in case any issues are discovered after the removal is completed.

Overall, removing in Continuous Delivery is an important practice that helps ensure that the software application or system is always up-to-date, secure, and reliable.

Aggressive testing is a crucial aspect of ensuring that high-quality software is delivered to customers quickly and reliably [23]. Aggressive testing involves a comprehensive testing strategy that covers all aspects of the software development process, including unit testing, integration testing, performance testing, security testing, and more.

The goal of aggressive testing is to catch bugs and other issues as early as possible in the development process [24]. By identifying and fixing problems early, teams can avoid costly delays and reduce the risk of introducing bugs into production environments.

To implement aggressive testing in Continuous Delivery, it is important to automate testing wherever possible [25]. This can involve using tools such as automated testing frameworks and continuous integration systems to run tests automatically every time new code is committed. Additionally, teams can use monitoring and logging tools to keep track of issues that arise in production environments and quickly resolve them.

Overall, aggressive testing is a key component of successful Continuous Delivery practices, and teams should aim to integrate testing into every stage of the development process. By doing so, they can ensure that they are delivering high-quality software that meets customer needs and expectations.

The author says that company development and operations teams might choose these techniques based on their requirements. In DevOps, testing in production provides the development team with constant input or response.

The author lists A/B testing, beta testing, and being able to monitor as testing for production testing as basically three techniques. Real-time monitoring of the production environment and prompt handling of anomalies are both benefits of DevOps. We explain how testing tools used in a DevOps approach aid in the development of efficient

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

test creation techniques [26, 13]. Finding precise tools for continuous practice activities presents issues for DevOps professionals. Additionally, they struggle to choose the right testing instrument, which wastes time and effort. According to Fredrickson, communication difficulties rather than technological difficulties constitute a key obstacle [27]. DevOps includes testing as a key component. Processes for testing services like development-driven testing, unit testing, and behavioral testing in DevOps are used by many enterprises. Code is sent straight to the test pipeline for additional quality improvement. DevOps should incorporate testing from the very start of the software development process [27, 16]. The project's quality is impacted by testing towards the end. Without first testing the project, continuous delivery might also be stopped. To improve performance during DevOps' continuous testing, the author recognized the need of choosing the appropriate tools and technologies. During testing, the tools immediately identify the issue, saving time and effort. We do a case study regarding the introduction of DevOps in a firm in New Zealand. The performance of the tester is improved, according to the writers, by learning new technologies and tools [28]. They observed that cooperation between the testing and development teams boosts output. The adoption of DevOps also reduces knowledge gaps and improves code quality [29]. A study, interviews, and a workshop with IT specialists were all part of the multi-perspective research strategy we used. They listed several competencies for productive DevOps teams [30]. During testing, automation was prioritized. Writing and executing test cases manually requires more time to locate and address the primary cause of an issue in testing, therefore knowledge of automation is crucial. The author explained how testing teams may get through obstacles in the way of a DevOps transformation [31]. They may set up a framework for continuous testing and receive a prompt response from the software development team. A quick release and greater test coverage are made possible by the continuous automation framework and coordination between development and operation. Both test environment performance and management testing as a code are topics of blogs. However, there isn't a lot of information on metrics that assist teams in enhancing their testing job. DevOps testing impacts/covers all services from development to deployment. Performance and security testing are encouraged by DevOps [32]. Monitoring, according to the author, might be useful in testing to acquire quick responses on technical services [33]. DevOps testing was developed to address the problems of agile development. DevOps testing enhances communication between all stakeholders and refines quality. However, there isn't a lot of systematic study on DevOps test results in scientific or academic journals, including case studies. Reliability and rapid deployment are made possible by solid and reliable architecture. It helps in obtaining prompt input from the operation and development teams [34]. The research on DevOps effects on software architecture is sparse, nevertheless. The goal of DevOps methods is to achieve high quality by expediting the implementation of change in the production environment. These procedures eliminate the structural barriers to transformation. Many software experts stated that monolithic architecture, in particular, does not applicable to DevOps and CD [28]. They claimed that DevOps is incompatible with highly coupled architecture.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

By discussing with software companies, the writers discovered several problems. They discovered that the adoption of DevOps might be impacted by legacy or outdated architecture. Due to integration issues in these outdated infrastructures, data collecting from various tools might be challenging [31, 29]. We analyzed the IT staff as a case study. They said that reference design simplifies the organization of employees and business-related problems in a DevOps setting [32, 27]. They found that cooperation between the development and operations teams reduced operational effort by 50%. DevOps also decreases faults by 3% while adding new modules to the production pipeline. The author claimed that teams are motivated to create new apps rather than concentrate on defect remediation when there are fewer faults. To determine which strategies may be useful for continuous infrastructure and DevOps, we carried out a methodical mapping research. Continuous code reworking in distributed software applications, according to them, slows down the testing and deployment process [33, 25]. The majority of businesses are switching to DevOps because of its quick feedback and delivery capabilities. There have been some DevOps transformation study studies. However not in the areas of architecture influenced by the shift to DevOps and continuous testing. The influence of architecture on production settings concerning Continuous Delivery (CD) and Continuous Testing (CT) is the subject of very little research [34, 39]. Additionally, few studies explain continuous delivery and testing in the DevOps context. The authors highlight the advantages and difficulties of embracing DevOps as well as the difficulties DevOps teams have when it comes to continuous testing []. However, it did not offer a solution to issues like adoption issues for continuous delivery or how architecture affects environments in DevOps.

H1- Continuous delivery (CD) is a DevOps practice that enables organizations to quickly deliver software changes to their customers.

H2- Work in small chunks for quick outcomes in the market.

H3- Continuous delivery allows developers to deploy code faster and with fewer errors. It enables teams to quickly and efficiently develop, test, and deploy applications.

H4- It ensures the software is running smoothly and issues are identified and addressed.

3. METHODOLOGY

We will motivate our work by argument and framework, and how we can overcome barriers and challenges through continuous delivery that we are facing in other software models. In this article, we describe the continuous delivery problems that are lack of automation, security, infrastructure, monitoring, and integration. Now here we are going to work on the lack of automation in continuous delivery [41].

Problems

1. Lack of Automation: Automation is a key part of any successful Continuous Delivery process. Without proper automation, manual processes will add extra time, effort, and cost to the process.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

2. Security: Security is often overlooked when implementing Continuous Delivery, but it's a critical factor. Security issues can create problems throughout the entire process, from source control to deployment.
3. Infrastructure: Implementing Continuous Delivery requires a reliable, secure, and scalable infrastructure. Without this, your deployments will be unreliable and prone to failure.
4. Monitoring: Continuous Delivery requires detailed monitoring of your systems to ensure that everything is running smoothly. Without this, you won't be able to recognize and address any problems that arise.
5. Integration: Continuous Delivery requires the integration of multiple tools and processes. This can be difficult to implement, and if done incorrectly, can lead to issues with the process.

Challenges and Barriers

1. Lack of Automation: Automating the entire DevOps process can be a challenge. Without automation, manual steps can add delays, introduce errors, and increase overall costs.
2. Security: Security is a major challenge for continuous delivery. As code moves quickly from development to production, it is important to ensure secure code and configurations are deployed.
3. Testing: Testing is a critical part of the continuous delivery process. Without proper testing, bugs and other issues can be introduced into production.
4. Integration: Continuous delivery requires the integration of multiple tools and processes. This can be a challenge, as teams must build and maintain reliable integrations between their tools.
5. Infrastructure: Infrastructure is an important part of the continuous delivery process. Teams must ensure their infrastructure is reliable, scalable, and secure.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

Table 1: Way to Mitigate Continuous Delivery Challenges

Continuous delivery challenges	Way to Mitigation
Lack of Automation	Conduct a thorough analysis of the software development lifecycle to identify processes that can be automated.
	Select the appropriate tools for automating different processes.
	Adopt a CI/CD approach to streamline software development and deployment.
	Encourage collaboration, communication, and teamwork between different teams involved in the software development process.
Security	Implement automated security testing throughout the entire development process, from code development to deployment.
	Conduct regular security assessments to identify any vulnerabilities in the system.
	Implement a secure deployment pipeline that ensures that only authorized code changes are deployed to production.
Testing	Implementing automated testing is one of the key solutions for testing in DevOps CD.
	Managing test environments can be a complex and time-consuming task. With the help of tools like Docker and Kubernetes, teams can create and manage test environments more easily.
	Test data is critical to the testing process, and managing it can be challenging. Tools like data masking and synthetic data generation can help teams create and manage test data more efficiently.
Integration	The use of a version control system (such as Git) allows developers to work collaboratively on the same codebase and track changes made to the code over time.
	Code reviews involve a process where developers review each other's code changes to ensure that they meet the organization's standards and best practices.
Infrastructure	Containerization using tools like Docker or Kubernetes enables the creation and management of isolated environments that can be easily deployed and scaled, ensuring consistency and reducing the risk of configuration issues.
	Cloud infrastructure like AWS, Azure, or Google Cloud provides infrastructure services on demand and allows for easy scaling and management of resources, making it an ideal solution for DevOps and continuous delivery.
	Implementing a CI/CD pipeline can help automate the testing, building, and deployment of infrastructure and applications, ensuring rapid and reliable delivery.

To investigate how automation affects skills and cooperation, we studied 10 app development teams within the setting of a big European IT services organization (more than 100 000 people) with 14000 IT staff. This business has been around for a while [34]. The organization is hierarchically structured with seven layers, and its highly bureaucratic culture is in opposition to its adaptability [16, 18]. The development and exploitation departments make up the Information Systems Division's two divisions. This organization was one of the first to adopt the DevOps methodology; staff has been utilizing agile methodologies for more than 14 years and DevOps for 7 years.

Five job responsibilities are examined as the units of analysis for the 11 teams, which are all immersed in a company with certain comparable conditions (all adopt agile management approaches) [35, 17]. However, because these teams employ various forms of rapid engineering, their test automation levels and infrastructure vary. We believe that when they transition to DevOps, automation will have an impact on the potential need for human skills and cooperation patterns.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

It must be observed that there is not necessarily consensus on what constitutes a large project in terms of team size [36]. The number of development teams, workers, lines of code, and project time are some of the recommended metrics [37]. There is no consensus, even when the measure refers to the number of developers. More than 20 developers, according to authors while 50–100 developers, according to authors constitute a sizeable project [38]. In our scenario, the management responsible mentioned that the variations resulting from largeness become obvious when teams have more than 12 to 15 people. As a result, we settled on 14 as the study's cutoff point. Projects that are contracted require more soft skills (such as communication), which are related to contracting.

We recognize three stages of automation:

1. Automation Level 1 Agile (Aut.1): Agile may be viewed as automation level 1 in the shift to DevOps compared to the conventional V-model since it allows for more regular updates as development becomes more flexible. Dev and Ops continue to operate in isolation with no collaboration or release automation; therefore DevOps is not fully achieved.

2. Automation Level 2 Continuous Integration (Aut.2): The Operation function must be in line with the Development function and both must start carrying out various tests [39, 23] that are coordinated with code development and are also automated to the greatest extent feasible.

3. Automation Level 3 Continuous delivery (Aut.3): which involves co-designing, performing, and ideally automating integration tests with other modules, end-to-end testing, performance tests, and user acceptance tests by Operation working with the Development function [40, 24]. The more automation there is, the more exchange and talk about monitoring there has to be so that the appropriate indicators may be appropriately built to track the activity of software development.

We were able to investigate 11 different sorts of teams in all feasible configurations. We were able to investigate the impact of the amount of automation while noticing a variety of groups in terms of the other two criteria thanks to the equal division of groups for each of the criteria.

These main collaborations are listed in Table 2 by role or profession; for instance, four out of the eleven RM indicated the developer as a primary collaborator. In line with expectations, the DPM reported the most instances of cross-functional cooperation among the four other jobs. The crucial function of the PO is a further noteworthy insight.

Table 2 shows the primary partnerships between Dev and Ops in more detail. We emphasize partnerships between RM and PO, AR and workers, or DPM with PE and workers from the development perspective. From the perspective of operations, we observe close cooperation between developers and PE, RM, and PO, as well as with creators.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

Table 2: Main Collaboration

OPS ← Main Collaborations cited → DEV		PO (N=12)	PE (N=12)	AR (N=11)	DPM (N=13)	RM (N=11)
	Product Owner, N	3	11	11	13	10
	Department & Project Manager, N	10	12	11	13	11
	Architect, N	3	4	1	1	2
	Scrum Master, N				3	
	Developer, N	9	12	8	8	4
	DevOps animator, N				1	
	Technical expert, N		3	1	1	2
	Quality expert, N	1				
	Multiple collaborators, N			1	2	
	Qualifier, N				1	
	Production Engineer, N	1	1	2	4	8
	Administrator Infrastructure Director, N			1		
	System Engineer, N		1			
	Operators, N		5	2	3	4
	Release Manager, N	2	6	6	1	

Furthermore, every responder has emphasized how better coworker cooperation is in an agile environment. This conclusion could be explained in part by the usage of collaboration tools, daily meetings, or sprint reviews. The organization of these regular encounters will formally schedule the interactions. However, many meetings may be viewed as an excessive workload. We also discuss the various collaboration styles and actor types that are engaged for each job.

Release Manager (RM): The RM serves as the focal point for many company-wide partnerships, particularly those with the manufacturing staff. RMs emphasized their connections with the project manager, which happened typically once a week. This cooperation is ongoing and essential to the success of initiatives to give particular tasks top priority. For instance, an RM clarified (Aut.3): "We set up a location dubbed the "Common Work Plan of Exploitability" where we communicate once a week. We review all actions and ongoing projects to determine what needs to be prioritized in the case of new activities ". These partnerships frequently act as real catalysts for generating a dynamic among all teams engaged in the same project. Nearly all RMs also mentioned working with production engineers (see Table 2). They converse frequently (RM Aut.2): "Before seeing the project manager, we have a conversation about a few things. One

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

visits the other for assistance with even the smallest issue, and vice versa." It should be highlighted that in this situation, the organization of project group meetings gave RMs, developers, and operators the chance to come together. The degree of automation in the shift from agile to DevOps may have an impact on how frequently these partnerships occur. In reality "Behind the project manager are the developers for me. However, this changes within the most sophisticated initiatives using the DevOps methodology" (RM Aut.3). When a result, additional interactions with developers may come organically as agile and DevOps approaches and concepts are applied at a higher application level.

Product Owner (PO): The majority of POs reported working with developers and DPM, as was to be expected. Some of these engagements are necessary for the demonstrations' sprint validation. The frequency and caliber of these interactions directly depend on the degree of automation in the shift from agile to DevOps. To feel like a member of an embedded staff, a PO recommended the following (Aut.3): "Today we are truly an embedded staff with marketing, developers." When numerous POs are a part of the same team, they communicate with one another every day by phone or email and during the weekly "sales meeting." In addition, there will be a discussion between the line managers and product managers during this meeting. Additionally, a few POs recalled having several conversations with functional architects. Most of the POs also described working with corporate sponsors or users, however, these relationships were not thought to be key ones, in keeping with the diverse functions inherent to PO duties. It should be noted that numerous POs emphasized the sparse collaborative contacts with supervisor-operators (Aut. 2): "Those with whom I communicate the least, are exploiting"; "I have difficulty expressing myself on this topic, they (supervisor-operators) are more associated to developers."

Architect (AR): The three primary teammates named by the architects were DPMs, POs, and developers, similar to earlier studies (see Table 2). The majority of architects reported speaking with DPMs to exchange knowledge and opinions about a project. Information-sharing partnerships with the development team was also highlighted, particularly during sprint evaluations and demos. This was emphasized by an architect (Aut.3): "we have to be specific on how to operate, and if practicable, build a better, more thorough partnership with the technical manager." These discussions frequently take occur. To establish norms, standards, and best practices, some architects proposed sharing information with the RM profession (Art. 2): "he (RM) is supposed to give over, to examine the technical architectural file." An AR (Aut.3) underlined how the switch to DevOps affected his job and his ability to collaborate: "It is a method that enables a deeper understanding of each participant's position in the project. In the end, everyone is still responsible, but problems with architecture will no longer exist since decision-making will be more evenly distributed."

Production engineer (PE): The principal partnerships mentioned by PEs are those with DPMs, POs, and developers (see Table 2). Therefore, the majority of PEs concur that there is strong reciprocal interaction with developers (Aut. 2 and 3). We

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

communicate with one another naturally, and as developers join our team, we are evolving into the DevOps team.

Additionally, the DPMs must submit applications to PEs as part of their daily tasks, resulting in extensive collaboration with shared duties. However, there have been some reported issues with cooperation, as one PE noted: "I was the person who stated what I anticipated in terms of paperwork, albeit it is their duty." PEs also talked about their partnerships with RMs. The frequency of this cooperation might, however, differ significantly amongst teams. Collaborations do occur frequently when the RM is fully committed to the team. According to a PE's description of this partnership (Art. 2), "In our organization, the function of RM was formed since administrative support for our project is important; it helps to lighten our work." To ensure that we do not overlook anything, the RM's job is to anticipate complicated processes. The jobs of RM and DPM no longer be available in a DevOps team, according to a PE (Aut.3). The Scrum Master handles the role of DPM at the DevOps team level, while the team as a whole is responsible for project management at the planning level. Additionally, production engineers, operators, technical experts, and PO all perform the duty of the RM.

Department and Project Managers (DPM): DPMs are at the heart of several partnerships, both inside and outside of teams. The manager profile works with several performers (see Table 2). The majority also acknowledged working together with other DPMs and POs, particularly to talk about budget, tracking, and item prioritizing. Several Managers also mentioned having a tight working connection with operational architects (Aut.2): "I work early with our operational architects to identify and schedule future development." Furthermore, despite not being a functional expert, the DPM accepts the result's conformance.

In conclusion, it is evident that the key collaborators' collaboration scope has altered and is more evenly distributed. The perceived higher depth of collaboration is a reflection of improved member comprehension. When groups are in automation 3, this more extensive richness seems to stand out more. As a result, taking into account our research on cooperation and skill sets, we can divide the agile to DevOps transformation into the following three stages. The limited range of collaborations at automation level 1 may be described by the DevOps concept itself, which expands collaboration among Dev and Ops while restricting it within Dev. COM (Communication) skills are crucial, and RES (Responsibility) skills are highly developed as IPS (Interpersonal Skills), as agile techniques need them. Unexpectedly, FLX (Flexibility Skills) and TWK are less common in their discourses. Thus, collaboration extends farther at automation level 2 than it does at automation level 1. Except for RES, all SSk (Soft Skills) are more prevalent at this level. The degree of collaboration is comparable to automation level 2 or may even significantly decline at automation level 3. This conclusion is explained by the fact that teams at automation level 3 are more aware of which stakeholders to limit or stop working with to increase efficiency. Except for RES, SSk rise relative to Aut.2 and this is especially true for FLX, TWK, and IPS.

DevOps with Continuous: Delivery DevOps with continuous delivery is a methodology that combines software development, operations, and quality assurance processes to

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

enable the continuous delivery of software applications. It emphasizes automation, collaboration, and communication between development and operations teams to ensure that applications are released in a timely, reliable, and repeatable manner. This approach enables teams to quickly and easily deploy new versions of their applications, while also reducing the risk of errors or downtime. DevOps without Continuous: Delivery DevOps without continuous delivery is a methodology that combines software development and operations processes but does not emphasize automation or collaboration between development and operations teams. This approach requires manual processes and lacks the automation, feedback loops, and feedback cycles that are essential for effective DevOps. As a result, it is more prone to errors, delays, and downtime. Additionally, it does not support the rapid and reliable deployment of applications, making it a less efficient and less reliable option for software development.

Table 3: Software with and without Continuous delivery

Software with DevOps continuous delivery	Software without DevOps continuous delivery
DevOps with continuous delivery is a methodology that combines software development, operations, and quality assurance processes to enable the continuous delivery of software applications.	DevOps without continuous delivery is a methodology that combines software development and operations processes but does not emphasize automation or collaboration between development and operations teams.
It emphasizes automation, collaboration, and communication between development and operations teams to ensure that applications are released in a timely, reliable, and repeatable manner.	This approach requires manual processes and lacks the automation, feedback loops, and feedback cycles that are essential for effective DevOps.
This approach enables teams to quickly and easily deploy new versions of their applications, while also reducing the risk of errors or downtime.	As a result, it is more prone to errors, delays, and downtime. Additionally, it does not support the rapid and reliable deployment of applications, making it a less efficient and less reliable option for software development.

4. DISCUSSION

Table 4 provides a summary of our key conclusions and empirical contributions. We talk about these results and create theoretical explanations. We examine the consequences of identifying several phases in the agile to DevOps transition and how doing so may result in the creation of a DevOps maturity model. Finally, we suggest looking at how our findings relate to intelligence.

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

Table 4: Findings and empirical contributions

Topic	Related Work	Findings
Skills	Non-technical skills are especially significant in agile software development [25, 14].	The greater the level of automation in DevOps, the more soft skills are required.
	Flexibility and interpersonal skills are important for agile teams [26, 27].	Flexibility, Interpersonal Skills, and Teamwork are more important in DevOps than in agile.
Collaborations	DevOps is something that is present in every organization that employs development and operations personnel. However, the amount of interaction differs and depends on the particular organization [28].	Collaboration patterns among the main players have drastically changed and are now more balanced between Dev and Ops.
	The adoption of more disciplined agile and collaborative DevOps leads to successful enterprise implementation [29].	The greater level of automation in DevOps is associated with an increased frequency of collaborations and with an improvement in the efficiency of these collaborations.

5. CONCLUSION

This study examined the degree to which the move toward DevOps represents increased intelligence. The move from Agile to DevOps may be broken down into three stages: Agile (Aut. 1), Continuous Integration (Aut. 2), and Continuous Delivery (Aut.3). We looked at the significance of cooperation styles and skill sets in the development of DevOps.

We discovered a fundamental interruption in the cooperation patterns inside teams as well as the soft skills that software companies are required to possess. At automation level 3, we saw that the primary contributors' collaboration scope had obviously changed and was now more evenly distributed across Dev and Ops. Richer cooperation reflecting improved member understanding is how we explained this outcome. Additionally, we discussed three stages that are distinguished by increased collaboration in conjunction with an increase in automation as well as a change in how each stage's talents are evaluated. At automation level 1, we demonstrated the significance of duties and communication skills, which was predicted given that Agile Methodologies encourage and promote these abilities then, at the following level, we discovered an expansion of these partnerships except for knowledge associated with duties. Even at automation level 3, they were falling off. We showed how DevOps encouraged automation, enhanced flexibility, and created observable outcomes like quicker and better delivery. According to Alter's notion of smartness, DevOps may therefore result in increased smartness for the Information System function.

FUTURE WORK

The future of Continuous Delivery in DevOps will include the further automation of the software development and deployment pipelines. This will enable faster and more frequent releases, reducing the manual work needed to deploy applications and services. Additionally, the use of DevOps practices such as Infrastructure-as-Code will allow for more consistent and repeatable deployments. To improve the workflow and

Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)
ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

efficiency of Continuous Delivery, DevOps teams will need to focus on creating more flexible and automated testing strategies and processes. Additionally, the adoption of artificial intelligence and machine learning tools to automate processes and improve the accuracy of deployments will become increasingly important. Finally, there will be an increased focus on security and compliance to ensure the safety and integrity of the applications and services being deployed.

References

1. R. K. Gupta, M. Venkatachalapathy, and F. K. Jeberla, “Challenges in Adopting Continuous Delivery and DevOps in a Globally Distributed Product Team: A Case Study of a Healthcare Organization,” in *2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)*, Montreal, QC, Canada, May 2019, pp. 30–34. doi: 10.1109/ICGSE.2019.00020.
2. M. Gokarna and R. Singh, “DevOps: A Historical Review and Future Works,” in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Greater Noida, India, Feb. 2021, pp. 366–371. doi: 10.1109/ICCCIS51004.2021.9397235.
3. M. Z. Toh, S. Sahibuddin, and M. N. Mahrin, “Adoption Issues in DevOps from the Perspective of Continuous Delivery Pipeline,” in *Proceedings of the 2019 8th International Conference on Software and Computer Applications*, Penang Malaysia, Feb. 2019, pp. 173–177. doi: 10.1145/3316615.3316619.
4. M. Shahin, M. A. Babar, M. Zahedi, and L. Zhu, “Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges,” in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Toronto, ON, Nov. 2017, pp. 111–120. doi: 10.1109/ESEM.2017.18.
5. R. Singh, “DevOPS Now and Then,” other, preprint, Nov. 2020. doi: 10.20944/preprints202011.0410.v1.
6. K. Beck, “Praise for Continuous Delivery”.
7. K. Beck, “Accelerate_ The Science of DevOps”.
8. L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, “A Survey of DevOps Concepts and Challenges,” *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–35, Nov. 2020, doi: 10.1145/3359981.
9. F. Erich, Chintan Amrit, and M. Daneva, “Report: DevOps Literature Review,” 2014, doi: 10.13140/2.1.5125.1201.
10. A. Mishra and Z. Otaiwi, “DevOps and software quality: A systematic mapping,” *Comput. Sci. Rev.*, vol. 38, p. 100308, Nov. 2020, doi: 10.1016/j.cosrev.2020.100308.
11. M. Shahin, M. Ali 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.
12. S. Zaib and P. K. Lakshmisetty, “A systematical literature review and industrial survey in addressing the possible impacts with the continuous testing and delivery during DevOps transformation”.
13. D. H. Salameh, “The Impact of DevOps Automation, Controls, and Visibility Practices on Software Continuous Deployment and Delivery”.
14. Wong, S., von Hellens, L., & Orr, J. (2006). Non-technical skills and personal attributes: the Soft Skills Matter Most. In *Proceedings of the 6th Australiasian Women in Computing Workshop* (pp. 27-33).
15. Yin, R. K. (1994). Discovering the future of the case study. Method in evaluation research. *Evaluation practice*, *15*(3), 283-290.

ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

16. Strode, D. E. (2016). A dependency taxonomy for agile software development projects. *Information Systems Frontiers*, *18*(1), 23-46.
17. Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*, 131-164.
18. Strode, D. E., Huff, S. L., & Tretiakov, A. (2009, January). The impact of organizational culture on agile method use. In *2009 42nd Hawaii International Conference on System Sciences* (pp. 1-9). IEEE.
19. Rolland, K., Dingsoyr, T., Fitzgerald, B., & Stol, K. J. (2016). Problematizing agile in the large: alternative assumptions for large-scale agile development. In *39th International Conference on Information Systems* (pp. 1-21). Association for Information Systems (AIS).
20. Dingsøyr, T., Fægri, T. E., & Itkonen, J. (2014). What is large in large-scale? A taxonomy of scale for agile software development. In *Product-Focused Software Process Improvement: 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014. Proceedings 15* (pp. 273-276). Springer International Publishing.
21. Hannay, J. E., & Benestad, H. C. (2010, September). Perceived productivity threats in large agile development projects. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10).
22. Elshamy, A., & Elssamadisy, A. (2007). Applying agile to large projects: new agile software development practices for large projects. In *Agile Processes in Software Engineering and Extreme Programming: 8th International Conference, XP 2007, Como, Italy, June 18-22, 2007. Proceedings 8* (pp. 46-53). Springer Berlin Heidelberg.
23. Ståhl, D., & Bosch, J. (2014). Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, *87*, 48-59.
24. Chen, L. (2017). Continuous delivery: overcoming adoption challenges. *Journal of Systems and Software*, *128*, 72-86.
25. Gren, L., Knauss, A., & Stettina, C. J. (2018). Non-technical individual skills are weakly connected to the maturity of agile practices. *Information and Software Technology*, *99*, 11-20.
26. Vivian, R., Tarmazdi, H., Falkner, K., Falkner, N., & Szabo, C. (2015, May). The development of a dashboard tool for visualising online teamwork discussions. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (Vol. 2, pp. 380-388). IEEE.
27. Wiedemann, A. M., & Schulz, T. (2017). Key capabilities of DevOps teams and their influence on software process innovation: a resource-based view.
28. Erich, F. M., Amrit, C., & Daneva, M. (2017). A qualitative study of DevOps usage in practice. *Journal of software: Evolution and Process*, *29*(6), e1885.
29. Ambler, S. W., & Lines, M. (2012). *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise*. IBM press.
30. Ghazi, A. N., Petersen, K., Reddy, S. S. V. R., & Nekkanti, H. (2018). Survey research in software engineering: Problems and mitigation strategies. *IEEE Access*, *7*, 24703-24718.
31. Fitzgerald, B., & Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176-189.
32. Pietrantuono, R., Bertolino, A., De Angelis, G., Miranda, B., & Russo, S. (2019, May). Towards continuous software reliability testing in DevOps. In *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)* (pp. 21-27). IEEE.
33. Laukkanen, E., Itkonen, J., & Lassenius, C. (2017). Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology*, *82*, 55-79.

ISSN: 1671-5497
E-Publication: Online Open Access
Vol: 42 Issue: 04-2023
DOI 10.17605/OSF.IO/6NYPX

34. Alter, S. (2019). Making sense of smart living, working, and organizing enhanced by supposedly smart objects and systems. In *Smart Working, Living and Organising: IFIP WG 8.6 International Conference on Transfer and Diffusion of IT, TDIT 2018, Portsmouth, UK, June 25, 2018, Proceedings* (pp. 247-260). Springer International Publishing.
35. Krey, M., Kabbout, A., Osmani, L., & Saliji, A. (2022). Devops adoption: challenges & barriers. In *55th Hawaii International Conference on System Sciences (HICSS), virtual, 3-7 January 2022* (pp. 7297-7309). University of Hawai'i at Manoa.
36. Khan, M. S., Khan, A. W., Khan, F., Khan, M. A., & Whangbo, T. K. (2022). Critical challenges to adopt DevOps culture in software organizations: a systematic review. *IEEE Access*, *10*, 14339-14349.
37. Hamunen, J. (2016). Challenges in adopting a Devops approach to software development and operations.
38. Silva, M. A., Faustino, J. P., Pereira, R., & Mira da Silva, M. (2018). Productivity gains of DevOps adoption in an IT team: a case study.
39. Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. (2016). DevOps adoption benefits and challenges in practice: A case study. In *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17* (pp. 590-597). Springer International Publishing.
40. Cois, C. A., Yankel, J., & Connell, A. (2014, October). Modern DevOps: Optimizing software development through effective system interactions. In *2014 IEEE international professional communication conference (IPCC)* (pp. 1-7). IEEE.
41. H. Hira, Khan, M. Afzal, S. Zubair, M. Atif, and K. Hamid, “DEVOPS METHODOLOGY IMPACT ON SOFTWARE PROJECTS TO LEAD SUCCESSES AND FAILURE THROUGH KUBERNETES MUHAMMAD WASEEM IQBAL,” *Jilin Daxue Xuebao GongxuebanJournal Jilin Univ. Eng. Technol. Ed.*, vol. 41, p. 11, Mar. 2023, doi: 10.17605/OSF.IO/D8YPH