Proceedings of the 8th International Conference on Communication and Electronics Systems (ICCES 2023)
IEEE Xplore Part Number: CFP23AWO-ART; ISBN: 979-8-3503-9663-8

Unifying Governance, Risk and Controls Framework Using SDLC, CICD and DevOps

1st Sai Alekhya Ganugapati
Symbiosis Institute of Digital and Telecom Management
(of Aff.) Symbiosis International (Deemed University)
Pune, India
esai.ganugapati2123@sidtm.edu.in

2nd Sandeep Prabhu
Symbiosis Institute of Digital and Telecom Management
of Aff.) Symbiosis International (Deemed University)
Pune, India
sprabhu@sidtm.edu.in

2023 8th International Conference on Communication and Electronics Systems (ICCES) | 979-8-3503-9663-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCES57224.2023.10192730

***Abstract*—This paper aims to study different software development frameworks and propose an efficient and comprehensive framework for handling Software Development Life Cycle (SDLC) in an IT Project. Risks and controls, work products and IT Audit risk parameters for each phase are also analysed. Furthermore, it covers Continuous Integration Continuous Deployment/Deliver (CICD) during support to the project along with management of code, branching strategies, storage of code, and CICD Pipelining. The paper also introduces Development-Operations (DevOps) and teaming structures to orchestrate project's success. It also depicts the importance of cross functional teams in a DevOps environment.**

***Index Terms*—SDLC, IT Project, CICD, DevOps, Risks, Governance**

I. Introduction

In the current competitive world, the need for delivering quality software promptly has led organizations to strive to adopt better frameworks. These frameworks enhance better practices for software development. SDLC is the process of developing IT Systems with analysis, design, implementation and maintenance; it is crucial as it makes the programmers work and evaluate concurrently by breaking down the entire life cycle into phases. Planning & Scoping, Design, Development, Test, Deploy, and Support are the phases in SDLC. Agile SDLC is an amalgamation of incremental and iterative processes. It emphasizes process flexibility and customer satisfaction through quick delivery of the working product. Agile SDLC breaks down this product into small incremental builds. These builds are provided in iterations. There are five phases within Scrum: Product backlog creation, Sprint Planning, Working on the Sprint, Testing/Demonstrating, and Retrospective. One of the other objectives of this paper is to present how CICD provides ongoing support once the product is developed. Continuous Integration, Delivery and Deployment are three phases of an automated software release pipeline. These phases take software from idea to delivery to the end-user. CICD sits at the heart of a DevOps methodology. By embracing agile concepts and practices, such as enhanced automation and improved coordination between development and operations teams, DevOps enables teams to build, test, and deliver software more quickly and reliably. In agile and DevOps, development, testing, and deployment take place. However, conventional agile ignores operations, a crucial component of DevOps.

The main contributions of this paper are threefold. First, insights are provided into SDLC, Agile, CICD, and DevOps concepts. Second, detailed descriptions of the process flows and methodologies for branching strategies are provided. Third, the paper details the risks and mitigations of implementing DevOps. The rest of the paper covers the concepts of Version Controlling, branching strategies, and Git Branching strategies. This paper also suggests an enhanced branching strategy to accommodate specific obstacles regarding fixing production issues and code alignment for various features in coordinated development.

II. Literature Review

The ”Waterfall” software development approach is a traditional downward-flowing stage model where feedback is limited between the stages of the SDLC. After-the-fact changes are too expensive to afford. Even though the waterfall approach grew to be a dominant software development methodology, the proponents of Agile SDLC argue that it is flawed as no project can finish in a pre-specified format [1]. Risk Management (RM) is based on the hypothesis of probability. Integrating security at the requirement and design stage generates maximum benefits. SDLC gives a basic understanding of starting a project, including the phases that provide the sequencing of activities. These activities are accomplished during the implementation of the software. RM offers a structured mechanism for giving visibility into threats. The potential impact of each risk is considered, and controls can be implemented accordingly [2]. Researchers have proposed many RM models in recent years, yet software projects cannot achieve a risk-free implementation. One of the significant reasons is the assessment of the risk during software development in an improper way. The IT Project teams will be able to analyse the occurrences and impacts of the risks in every phase of SDLC and apply relevant, meaningful controls when necessary [3]. Taking the peculiarities of IT Projects into account, determining the selection and usage of the RM methods and models must be in tandem with the development of the features in software development [4]. To enable fast delivery, minimize system interruption, and improve productivity, Agile practices with CICD pipeline approach have significantly amplified the projects' efficiency. The organization has to practice CI in order to adopt CD. Manual processes will tremendously reduce once CICD is completely adopted [5]. To help the software developers manage the source codes and enable them to keep all the versions of the project, Version Control Systems (VCS) are being used. VCS is a way to manage, organize and coordinate the development of the project modules. VCS makes it easy for the developers to work together effectively as it supports a collaborative framework. Collaboration becomes challenging if Version Control System is not adopted [6]. VCS supports multiple branching strategies which are an integral part of SDLC. A distributed VCS is adaptable to agile and it becomes easier for the developer to follow why the changes are made in the code [7]. CICD has become a popular practice for the development teams to ensure rapid delivery of new features and fixes. CICD is truly achieved if the testing gets automated and then releases the software versions multiple times a day. Security testing and CICD practices are discussed frequently, but combining both topics will create a unifying approach for Continuous Testing [8]. Considering the rapid change in customer demands and the complexity in managing the increasingly complicated IT architectures, many organizations have started implementing DevOps –joint, cross-functional teams. This integrates the knowledge, skills, and tasks concerning planning, developing, and running software product activities [9]. To reliably and often update a system that is already in operation, DevOps is very crucial. The automation between software development, operations and cross-functional collaboration is presumed in DevOps. Its adoption and implantation are a long-term activity requiring a supportive culture and mindset and technical practices [10]. Companies often struggle with demonstrating control over their software delivery processes due to high degree of automation and decentralized decision-making structures. Risk appetite and DevOps Maturity Model influence how organizations design their DevOps environment. While adopting DevOps, organizations must ensure that the adoption does not impede the agility of the DevOps methodology. The anticipated risk and mitigation activities must be practical and create validity in the audit [11]. DevOps, an emerging methodology that reduces the friction between development and operations team, offers continuous fast delivery and enables quick responses for changes within SDLC. Coupling the DevOps stages with its relevant tools and methods to improve the quality of the software and delivering methodology has become quite significant [12]. Cross-functional teams provide new initiatives in expeditious and backbreaking environments. DevOps collaborates development and operational teams into one team, resulting in highly collaborative cross-functional teams. These bring about the software delivery life cycle responsibilities efficiently [13].

III. Research Methodology

This study uses phenomenological research using grounded theory approach. The purpose of the current study is to understand the fundamental concepts of SDLC and its collaboration with Agile, CICD and how fast delivery and minimizing system interruption, and productivity improvement can be achieved through CICD, DevOps and how it became an evolution of agile practices or as a missing piece of agile. Furthermore, the study aims to provide a unifying framework that compiles the methodologies and practices along with risk management and IT governance. To gain a general understanding of SDLC, CICD and DevOps frameworks, we have started our analysis by reading through the industry journals. Moreover, we studied the concept by contacting IT consulting firm and meeting with IT Consultants and managers who are experienced in consulting in CICD and DevOps teams. Furthermore, we utilized secondary data such as company reports, blog articles and conference presentations for data triangulation.

IV. Framework and Discussion

This section of the paper covers the concepts of SDLC and its phases, Mapping of Agile Software development with SDLC phases, Phases of Agile SDLC and risks and controls of each phase. Furthermore, this section also covers CICD, cost and gain of CICD, Continuous Deployment, Version Controlling – Branching strategies, Git Branching Strategies, Enhancement in the branching strategy, Continuous Testing, Secure CICD, CICD Stages and Activities, DevOps and its cross-functional team structure.

A. Software Development Life Cycle (SDLC)

Agile SDLC consists of both iterative and incremental process models. It emphasizes process flexibility and customer satisfaction through quick delivery of functional software. Small incremental builds are used in Agile SDLC while creating the product. There are five phases in Scrum - Product backlog creation, sprint planning, working on the sprint, testing and demonstrating, and retrospection. The phases of SDLC are Planning and Scoping, Design, Development, Test, Deploy, and Support. Some risks associated with the planning and scoping stage are an unrealistic budget, unrealistic schedule, unclear project scope, incomplete, inaccurate and ambiguous requirements, ignoring non-functional requirements, conflicting user requirements, gold plating, misunderstood domain-specific terminology and unrealistic or lack of detailed budgeting. Similarly, the risk associated with the design phase is unclear tasks for the development team, incorrect choice of technical architecture and system standards, large component size or less reusable components, inconsistent, incomplete extensive specification of the design document, accidental omission of data processing functions, unclear establishment of sprint goals and improper refinement of product backlogs. The development phase contains poorly documented design document, lack of independent working environment, incompatibility of technical architecture and programming language, inconsistent, complex and ambiguous component development, insecure code, misunderstanding of code snippets, poor documentation of test cases, inadequate regression testing after adding new code snippets, integrating wrong versions of components, disabling access for code review, and unintentional code omission. Some of the common risks associated with the test phase are bugs during integration, not being able to achieve the desired functionality, difficulty in locating and replacing defects, limited testing resources like time, budget and tools, unrecognized metrics, lack of recognition of the loss of data, duplicated or corrupted data, manual intervention or lack of automation in the connectivity of systems, infrequent updating of Requirement Traceability Matrix (RTM), and lack of remediation plan or fallbacks if RTM fails. Premature deployment, wrong code deployment, difficulty in using the system, software faults, too many user scenarios, and sponsor denial due to lack of clarity are some of the common risks associated with the deployment phase and budget overrun, limited understanding of project flow, failure to conduct proper project closure, inefficient planning of hyper-care, and inconsistency in the collection of data to measure the metrics in terms of costs and efforts are some of the common risks associated with the support and closure phase.

B. *CICD - DevOps*

CICD is the best way to provide ongoing support to the project and how the code needs to be managed, how continuous testing needs to be adapted, how to secure CICD, the stages and activities of CICD, DevOps and its teaming structure.

1) CICD: CI shrouds the process of numerous developers attempting to merge their code changes repeatedly with the central code repository of the project. They also trigger automatic testing and validating processes. This supports in minimizing the code conflict. Continuous Delivery (CD) is an extension of CI. It manages the packaging of the code in the form of a build and delivers the same to the end-users. Tools are run automatically to generate builds. It requires human intervention to physically and judiciously trigger the deployment of changes or new add-on features. Through CD, automatically launching and distributing the software to users is achieved. Using CD, the code releases are set ahead of time by the teams responsible. The code mechanically gets distributed and deployed into the production environment. Automatically moving the builds is done through scripts or tools.

2) Version Control System: Version controlling provides teams and organizations an analytical means to manage their files and regulate their creation, controlled access, updating, and deletion.

a) Branching strategies: Branching strategies is one of the most emotive topic that triggers heated debates online and offline. A branching strategy is the team's agreement on how and when to create and merge branches in VCS. The branching strategy must be chosen according to the team needs and system needs.

Trunk-based development

TABLE I
COSTS AND GAINS OF CICD

CICD	Cost	Gain
Continuous Integration	Automated test for every new feature, bug fix-improvement. Cost of the server that keeps track of the main repository and automatically runs tests when new commits are pushed. Merging of changes as often as possible.	Shipment of less bugs. Easy build of releases. Less context switching. Drastic reduce in testing costs. Reduction in time spent by QA on testing
Continuous Deployment	Best testing culture – Determines the quality of releases. Fast-paced documentation. Feature flags turn out to be an intrinsic part of the release process - Coordination with other departments like Marketing, PR etc.	Faster Development. Less risky releases and easy fix of issues. Continuous stream of improvements. Increase in quality

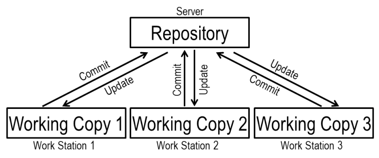

Fig. 1. Working explaination of Version control

- Division of work into small batches and merges into the trunk at least once daily.
- Branches last less than a few hours, with many developers frequently merging their changes to trunk.
- Developers must fix the problems occurred immediately after pushing into the trunk

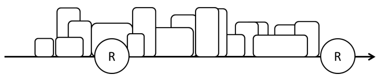

Fig. 2. Trunk based development

Feature Branch

- Keeps individual features separate from the rest of the codebase.
- Keeps work in progress out of the master branch – less prone to unfinished functionality deployment.
- Delay in the integration of changes until the feature is ”complete” – loses the benefit of CI.
- Complex bug fixing.

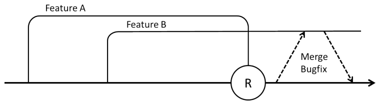

Fig. 3. Feature Branching

Release Branch

- Updates are delivered at intervals rather than as soon as ready. Easier to support multiple versions in production.

- Release branch is created once the changes are planned and no further features are merged in.
- Easier to deploy fixes to old versions. Emergency updates can be done either in master or release branch.

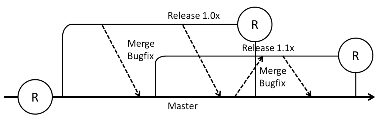

Fig. 4. Release Branching

Story or Task Branch

- Lowest level of branching, and each issue implemented has its own branch
- Best mix of continuous merging and safeguards.
- Mitigates risk of long-running branches. Easier for code review, QA and tests.

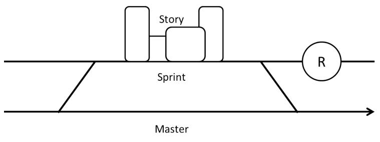

Fig. 5. Story or Task Branching

b) Git Branching Strategies: Several source control techniques are available in the market to enable the branching approach. However, Git is one of the most popular systems available in the industry. It provides several functionalities. These functionalities are intended to facilitate work completion by being compatible with the teams that adopt Agile Software Development. Compared to other strategies, three methods work better and are more relevant.

GitFlow

- A simpler alternative to GitFlow for smaller teams.
- No Release branch
- Short-lived feature branches
- Main idea – Keeping the master code in the deployable state and hence support Continuous Integration and continuous deployment Process
- Each commit contains a completed and isolated change.
- Pros – Parallel development; Organized code and work; Multiple versions of the production
- Cons – Merge difficulty; Difficult to detect issues; Slower development and release process

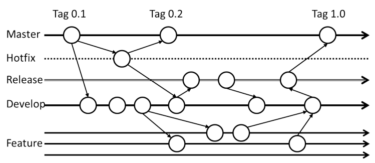

Fig. 6. GitFlow

GitHub Flow

- A simpler alternative to GitFlow for smaller teams.
- No release branch
- Short-lived feature branches
- Main idea – Keeping the master code in the deployable state and hence support Continuous Integration and continuous deployment Process
- Each commit contains a completed and isolated change.
- Pros – Fast and streamlined; Quick feedback loops; Continuous deployment
- Cons – Multiple version handling; Susceptible to bugs; Small teams sustainability

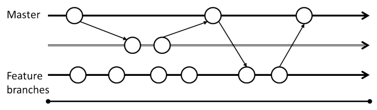

Fig. 7. – GitHub Flow

GitLab Flow

- Feature-driven development and feature branching with issue tracking.
- The staging environment and production environment are different.
- Allows the code to pass through internal environments before reaching production.
- No control over the timing of the release.
- Pros – Issue fix in production; Transparency code issue; Safe premature codes
- Cons – Alignment of production code; Difficult to implement CICD; Delay in release

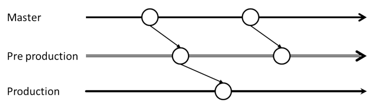

Fig. 8. GitLab Flow

V. Proposed Framework: Enhanced Branching Strategy

There is no ”one size fits all” strategy that ensures the least amount of issues during implementation as every project is unique in its way. To design a suitable working flow for a team, they should consider some generic problems that might arise, such as

- Fixing production issues with the minor efforts.
- Aligning code in different deployment environments.
- Reducing overhead to align for a wide range of features in coordinated development.

In this model, GitLab is considered the base VCS. The model is divided into three phases.

1) **Continuous Integration** – the main goal is to improve trackability for a feature lifecycle and enhance the integration of the developed code snippets. This can be achieved by adding a hotfix branch that can merge directly to the master branch after the production

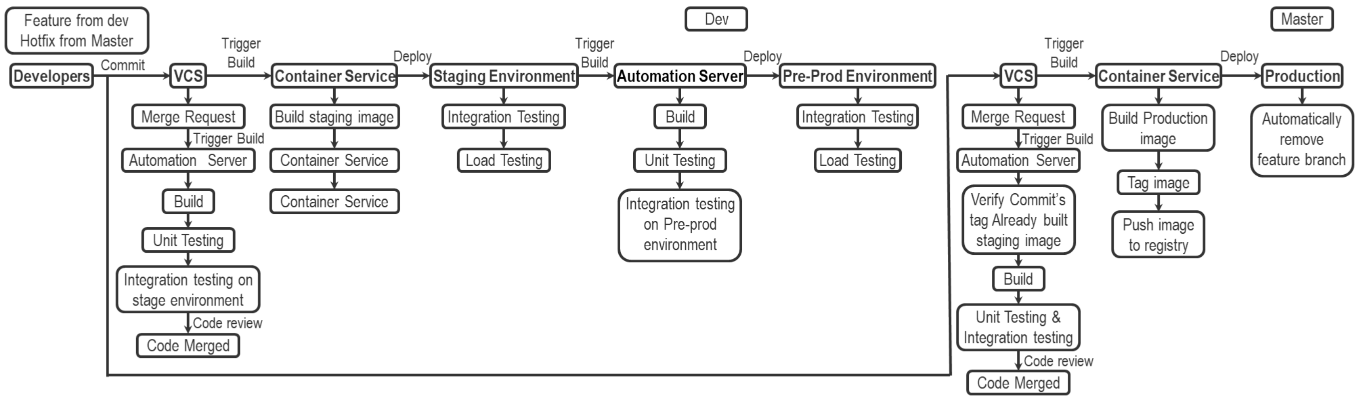

Fig. 9. Proposed Framework

issue is fixed. By continuously merging small chunks of code, CI is enhanced, and the possibility of conflicts is reduced during emergency issue resolution. To avoid the mistakes, developers make whilst merging, automation for creating the merge request has to be enabled. This reduces the complexity of maintaining the branch and automating the deletion of the feature branch after the feature development is done and launched.

2) **Continuous Delivery/Deployment** – The staging and the pre-production environment are used for a more stable testing mechanism. New features are tested in these environments before production to avoid adverse effects. Other testing features still in development do not interrupt integration testing with external teams. Continuous Delivery is enhanced when a separate default branch with enhanced stability, quality and hotfix handling is used.
3) **Fast test automation** – Unit testing and integration testing are precisely done. Automating integration testing helps detect any issue simulating real HTTP API access scenarios before deploying the same to a staging or pre-production environment. This reduces the developer's overhead to fix an already known issue and elevates code stability. Pre-build and unit testing are added while requesting for merge, reducing the developer's overhead during unit testing. Knowledge is shared amongst the team.

TABLE II
MERGE REQUEST RULES

Branch	Rules
Hotfix	Branch off from Master Require ”Open merge pull request” More than 2 reviewers
Features	Branch off from Develop Require ”Open merge pull request” More than 2 reviewers
Develop	Require ”Open merge pull request” for production deployment More than 2 developers to approve the merge

CICD Enhancements

1) Improve trackability of the issue in which commit.
   - Add Hotfix branch
   - Proper use of Naming convention – For instance, Feature + IssueID
   - After this hotfix is launched into production, automatic removal of feature branch ensures a clean state of feature and hotfix in the lifecycle.
2) Enhancing code consistency and maintainability in different deployment environments.
   - Adding deployment environment connected with development branches – Helps continuous delivery
   - Using the dev branch for staging and pre-production environments – reduces overhead maintenance.

a) Continuous Testing(CT): Feedback is necessary for the team during the testing phase. It helps them determine if any further changes are required for the developed product. Especially, faster feedback is essential. In CT, applications are continuously tested all over the SDLC process. CT evaluates the quality of the software across the SDLC, provides feedback faster and facilitates higher-quality and quicker deliveries. Repeatable tests are run through this phase whenever a code is committed into the repository. This leads the CICD pipeline to provide faster feedback, and this change would not adversely impact the product.

b) Secure CICD: Given the primacy of the CICD pipeline, security should be a priority for IT Product development teams. Inadequate security protocols result in adverse risks like malware getting inserted while the application code passes down the CICD pipeline, and exposure of sensitive data in the CICD pipeline. Threat modelling, Architecture Risk Analysis(ARA), fuzz testing, red teaming, SAST, DAST, SCA, Risk Based Testing (RBT), IAC, and penetration testing are a few security practices that the development team can use depending on the use case. Some of the best approaches to prevent security problems at a high level are as follows

- Early Identification of security issues in the development cycle - Teams should be able to perform security testing independently with minimal friction.
- Incorporation of Threat modelling while the system is in the design stage - Linters and static analysis should be

kept in place to eradicate manageable issues in advance.

- Usage of Software Composition Analysis (SCA) - Verification of dependencies in open-source platforms and clear them from vulnerabilities
- After code commit, SAST usage to locate weaknesses. Another round of SCA - Best practice to incorporate SAST tool in the automation pipeline and IDE.
- After the build is complete, leverage the tests on security integration - Execute this build in an out-of-the-way container equipped to test the validation of the input, network calling and authorization process.
- Test logging protocols and access controls - Ensure limited access to relevant users; Ensure that software logs required security and performance metrics.
- Security tests to be continued after production – Systematize configuration management and patching which results in the software having access to secure dependencies.

c) DevOps – Teaming structure: Traditional agile falls short on operations while development, testing, and deployment occur in both agile and DevOps. Operations are an integral part of DevOps. It is an evolution of agile. Typically considered as a missing piece in agile. DevOps endeavours to incorporate agile practices into operations. Agile Manifesto places a high priority on working software, customer participation, and adapting to change, which are clearly the same for DevOps. While, Agile is the methodology, philosophy, the framework that allows building changes quickly and deploying them. CICD deals with the tooling that enables it to do the same, and DevOps deals with the teaming component, which focuses on building cross-functional teams to facilitate the agile methodology. A cross functions team is formed, having the product as the focus. The team members are from different disciplines and form into a single team to carry through the common objective of the team.

d) Risks: CICD introduces several operational risks like insufficient flow control mechanisms, inadequate identity management, dependency chain abuse, poisoned pipeline execution, insufficient pipeline-based access controls, insufficient credential hygiene, insecure system configurations, ungoverned usage of 3rd party services, Improper artifact integrity validation, Insufficient logging and visibility. When a new or updated code is committed to the VCS, a sequence of tests are launched. These automated tests aid multiple roles. They act as gatekeepers that avoid risky, unstable, and insecure code to be deployed to the customers. The major risks of test automation are over-reliance on manual testing and insufficient integration tests. Release pipeline risks include building without testing, complex pipelines, wrong tool usage, human intervention, releasing the wrong build, downtimes, late lockdowns, rushed product delivery, and lengthy rollbacks. The risks associated with tooling include choosing the incompatible CICD tool set, incorrect and improper configuration and integration of tools. Container security risks include insufficient documentation, untrusted containers' usage, an insecure configuration of other components, and poor secret management. The quality of the end project depends significantly on the quality of the source code. Some of the version control risks to be kept in mind are the lack of breaking up of projects into many repositories, lack of a central server, and unlocked pipelines.

VI. Conclusion

This paper presented insights on SDLC, CICD and DevOps, additionally covering the risk factors associated with the same along with VCS and types of branching strategies associated with it. The paper also proposed an enhancement on the existing branching strategies. The paper also identified that DevOps implementation in an organization is more of an extension of agile with operations. DevOps deals with the teaming component which focuses on building cross functional teams to facilitate the agile methodology.

References

[1] [L. Mahadevan, W. J. Ketinger, and T. O. Meservy, "Running on hybrid: Control changes when introducing an agile methodology in a traditional 'waterfall' system development environment," Communications of the Association for Information Systems, vol. 36, pp. 77–103, 2015, doi: 10.17705/1cais.03605.

[2] K. Sahu Shakuntala, R. Kumar, K. Sahu, and R. Rajeev Kumar, "Risk Management Perspective," 2014. [Online]. Available: https://www.researchgate.net/publication/273063901

[3] B. Roy, R. Dasgupta, and N. Chaki, "A study on software risk management strategies and mapping with SDLC," Advances in Intelligent Systems and Computing, vol. 396, pp. 121-138, 2016. doi: 10.1007/978-81-322-2653-6_9.

[4] V. Babenko, L. Lomovskykh, A. Oriekhova, L. Korchynska, M. Krutko, and Y. Koniaieva, "Features of methods and models in risk management of IT projects Keyword: IT project risks Project risk management Software Methodology Project management Corresponding Author," vol. 7, no. 2, pp. 629–636, 2019, [Online]. Available: http://pen.ius.edu.ba

[5] S. A. I. B. S. Arachchi and I. Perera, "Continuous integration and continuous delivery pipeline automation for agile software project management," in MERCon 2018 - 4th International Multidisciplinary Moratuwa Engineering Research Conference, Jul. 2018, pp. 156–161. doi: 10.1109/MERCon.2018.8421965.

[6] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version Control System: A Review," in Procedia Computer Science, 2018, vol. 135, pp. 408–415. doi: 10.1016/j.procs.2018.08.191.

[7] D. Arve, "Branching Strategies with Distributed Version Control in Agile Projects," 2010.

[8] T. Rangnau, R. v. Buijtenen, F. Fransen, and F. Turkmen, "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines," in Proceedings - 2020 IEEE 24th International Enterprise Distributed Object Computing Conference, EDOC 2020, Oct. 2020, pp. 145–154. doi: 10.1109/EDOC49727.2020.00026.

[9] A. Wiedemann, M. Wiesche, H. Gewald, and H. Krcmar, "Understanding how DevOps aligns development and operations: a tripartite model of intra-IT alignment," European Journal of Information Systems, pp. 458–473, 2020, doi: 10.1080/0960085X.2020.1782277.

[10] L. E. Lwakatare et al., "DevOps in practice: A multiple case study of five companies," Inf Softw Technol, vol. 114, pp. 217–230, Oct. 2019, doi: 10.1016/j.infsof.2019.06.010.

[11] O. H. Plant, J. van Hillegersberg, and A. Aldea, "Rethinking IT governance: Designing a framework for mitigating risk and fostering internal control in a DevOps environment," International Journal of Accounting Information Systems, Jun. 2022, doi: 10.1016/j.accinf.2022.100560.

[12] A. Alnafessah, A. U. Gias, R. Wang, L. Zhu, G. Casale, and A. Filieri, "Quality-Aware DevOps Research: Where Do We Stand?," IEEE Access, vol. 9, pp. 44476–44489, 2021, doi: 10.1109/ACCESS.2021.3064867.

[13] A. Wiedemann, M. Wiesche, and H. Krcmar, "Integrating development and operations in cross-functional teams — Toward a DevOps competency model," in SIGMIS-CPR 2019 - Proceedings of the 2019 Computers and People Research Conference, Jun. 2019, pp. 14–19. doi: 10.1145/3322385.3322400.