Revista Facultad de Ingeniería (Rev. Fac. Ing.) Vol. 34, No. 72.
L-ISSN: 0121-1129, e-ISSN: 2357-5328.

DOI: 10.19053/01211129.v34.n72.2025.18997

IMPLEMENTATION GUIDE OF SOFTWARE DEVELOPMENT BEST PRACTICES BASED ON DEVOPS

Guía de implementación de buenas prácticas para desarrollo de software basada en DevOps

Manuel Pastrana
Institución Universitaria Antonio José Camacho, Cali, Colombia.
mapastrana@admon.uniajc.edu.co

Hugo-Armando Ordoñez-Erazo
Universidad del Cauca, Popayán, Colombia.
hugoordonez@unicauca.edu.co

Carlos-Alberto Cobos-Lozada
Universidad del Cauca, Popayán, Colombia.
ccobos@unicauca.edu.co

Mirna Muñoz
Centro de Investigación en Matemáticas-CIMAT, Zacatecas, Mexico.
mirna.munoz@cimat.mx

Received: 04-02-2025
Accepted: 05-05-2025

ABSTRACT

Software development processes face the constant challenge of improving quality controls within the project's construction without affecting operational efficiency and meeting customer needs. DevOps offers a potential solution by enabling software development with the best practices; however, the disadvantage of frameworks such as DevOps is that they indicate that they create the practices, but not how to implement them through precise guidelines and under specific tools, leaving this step to an experimental process of trial and error, which can sometimes be costly. To address that issue, this article proposes a guide that facilitates the step-by-step adoption of five practices: version control, change requests controlled with manual code inspection, continuous integration, static code analysis, and implementing an automated pipeline for continuous integration. The methodology involves: 1) identifying software development best practices and organizing them into a step-by-step process that allows for phased implementation; 2) detailing the steps to implement each practice with specific technologies; 3) practical application; and 4) analysis and discussion of the results. The guide was presented to students, who used it to develop a short course project. Implementing the guide's practices allowed them to recognize that the information from different tools allows for quality control as the project evolves, making the process more efficient.

Keywords: best practices for software development; DevOps; implementation guide; software engineering; software quality assurance.

How to cite: M. Pastrana, H. A. Ordoñez-Erazo, C. A. Cobos-Lozada, M. Muñoz, "Implementation Guide of Software Development Best Practices Based on DevOps". *Revista Facultad de Ingeniería*, vol. 34, no. 72, e18997, 2025. https://doi.org/10.19053/01211129.v34.n72.2025.18997

RESUMEN

Los procesos para desarrollo de software tienen el reto constante de mejorar los controles de calidad dentro de la construcción del proyecto, sin afectar a la eficiencia operativa y la capacidad de respuesta frente a las necesidades de los clientes. DevOps ofrece una solución potencial al proporcionar un conjunto de buenas prácticas de desarrollo de software; sin embargo, la desventaja que poseen los marcos de trabajo como DevOps es que indican que crean las prácticas, pero no cómo implementarlas a través de guías precisas y bajo herramientas específicas, dejando este paso a un proceso experimental de ensayo y error, que en algunos casos puede resultar costoso. Para abordar dicha situación, este artículo propone una guía que facilite la adopción paso a paso de cinco prácticas: control de versiones, solicitudes de cambio controladas con inspección manual de código, integración continua, análisis de código estático e implementación de una canalización automatizada para la integración continua. La metodología utilizada implica: 1) identificar las mejores prácticas de desarrollo de software y organizarlas en un paso a paso que permita una implementación escalonada; 2) detallar los pasos para implementar cada práctica con tecnologías especificas; 3) aplicación práctica y 4) análisis y discusión de los resultados. La guía fue presentada a estudiantes, quienes la utilizaron para desarrollar un proyecto de curso corto. La implementación de las prácticas de la guía les permitió reconocer que la información de diferentes herramientas permite el control de calidad a medida que el proyecto evoluciona haciendo más eficiente su proceso.

Palabras clave: aseguramiento de calidad de software; buenas prácticas para desarrollo de software; DevOps; guía de implementación; ingeniería de software.

GUIA DE IMPLEMENTAÇÃO DE BOAS PRÁTICAS PARA DESENVOLVIMENTO DE SOFTWARE BASEADA EM DEVOPS

RESUMO

Os processos de desenvolvimento de software enfrentam constantemente o desafio de melhorar os controles de qualidade na construção dos projetos sem comprometer a eficiência operacional e a capacidade de resposta às necessidades dos clientes. O DevOps oferece uma solução potencial ao fornecer um conjunto de boas práticas para o desenvolvimento de software. No entanto, uma das desvantagens de estruturas como o DevOps é que, embora definam as práticas, não indicam como implementá-las por meio de guias precisos e ferramentas específicas, deixando esse processo a um experimento de tentativa e erro, que pode ser custoso em certos casos. Para enfrentar essa situação, este artigo propõe um guia que facilita a adoção passo a passo de cinco práticas: controle de versão, solicitações de alteração controladas com inspeção manual de código, integração contínua, análise estática de código e implementação de um pipeline automatizado para integração contínua. A metodologia utilizada envolve: 1) identificar as melhores práticas de desenvolvimento de software e organizá-las em um passo a passo que permita uma implementação escalonada; 2) detalhar os passos para implementar cada prática com tecnologias específicas; 3) aplicação prática e 4) análise e discussão dos resultados. O guia foi apresentado a estudantes, que o utilizaram para desenvolver um projeto de curso de curta duração. A implementação das práticas do guia permitiu que reconhecessem que a informação proveniente de diferentes ferramentas possibilita o controle de qualidade à medida que o projeto evolui, tornando o processo mais eficiente.

Palavras-chave: boas práticas de desenvolvimento de software; DevOps; engenharia de software; garantia da qualidade de software; guia de implementação.

1. INTRODUCTION

Software development companies constantly face problems associated with the improvement of their production processes, especially when thinking of significantly increasing quality [1]. However, companies with the greatest economic possibility of reinvesting in a continuous improvement plan manage to make their products and services more profitable; therefore, they stand out in the industry [2].

While DevOps offers a potential solution to the challenges of modern software development by integrating development and operations through collaborative practices, automation, and continuous feedback loops, effectively implementing it in practice remains a significant challenge for both industry and educational settings [3]. Although there are academic proposals, like Grotta and Prado's DevOpsBL [4], which emphasizes project-based learning, they often remain tool-agnostic by prioritizing soft skills over concrete technical guidance; DevOpsEnvy, presented by Rong et al. [5], which creates a simulated DevOps environment but lacks the detailed implementation steps needed to translate theoretical understanding into practical expertise; other authors underscore the challenges of balancing automation with pedagogical goals [6], which highlights the need for hands-on experience without providing a structured methodology for tool adoption. Additionally, Hobeck et al. [7] focus on integrating CI/CD pipelines in graduate-level courses, thus demonstrating the value of real-world toolchains, but their approach lacks the phased, incremental model necessary for broader educational contexts. To address those limitations, this article proposes a structured implementation guide with detailed instructions for specific practices and technologies, thus enabling both students and interns to build competency incrementally while experiencing quality improvements at each stage.

This highlights that academic training programs in higher education institutions are not far from this context; therefore, they constantly seek new ways to address the needs of the industry and solve them through the courses that make up curricula [7].

Reducing the gap between academia and industry in undergraduate training programs is essential to propose exercises that confront students with real situations [8]; for instance, the latent need to implement best practices for software development in a software engineering course, where the DevOps framework stands out as an alternative with clear directions on what should be done to improve the development process [9]. However, because there is a wide variety of practices and tools that could be implemented, the trial-and-error process for adoption is slow and costly [10]. In addition, the documentation associated with carrying out step-by-step and staggered adoption for a specific set of technologies is scarce, thus making it more difficult, both in industry and academia.

Regardless of their size, software development companies face multiple challenges associated with their daily activity, such as managing technical debt, speeding up product delivery, maintaining high-quality standards that meet the market's needs, and ensuring customer loyalty [11]. To overcome them, it is key to implement best practices for software development based on DevOps because it enables setting different preventive quality filters up [12], such as control points and continuous feedback that serve the development team directly during construction, empowering those who build the product to focus on quality [13]. This scenario, according to [14], allows collaborators to know the actual status of the project in terms of quality to avoid and reduce rework resulting from the need for adjustments after the completion of one or more functionalities during a development sprint —a time box that goes from one to four weeks where commitments are agreed. Feedback is obtained from the interested parties at the end of it, as mentioned above [15].

Although frameworks such as DevOps propose a set of best practices that can be applied manually at initial stages, as suggested by [6], implementing an automated cycle can help mitigate these problems by allowing early detection of errors, efficient code construction, and continuous quality analysis. It also generates continuous information and will enable it to be always consulted, as mentioned in [16] and [17]. Hence, this context is not isolated from undergraduate training processes, where professionals are trained for the industry.

This work proposes an approach to reduce the gap between both contexts (industry and academia) with the aim of training students under real scenarios in the adoption of some of the main practices suggested by DevOps, following a clear and well-structured process with specific tools and their gradual implementation.

In recent years, several studies have addressed the challenges and opportunities of integrating DevOps practices into software engineering education. Notably, Grotta and Prado's DevOpsBL framework emphasizes project-based learning, where students engage with DevOps concepts through capstone projects that simulate real-world scenarios [4]. Their approach is primarily tool-agnostic, focusing on the development of soft skills and team collaboration rather than prescribing specific technologies or step-by-step technical procedures. While this fosters adaptability, it can leave students without clear guidance on how to operationalize DevOps best practices with concrete tools.

Similarly, Rong et al. [5] present DevOpsEnvy, an educational support system that creates a simulated DevOps environment for students to experience the workflow and collaboration typical of industry settings. This system provides a virtualized ecosystem that allows students to practice DevOps principles in a controlled setting. However, like DevOpsBL, DevOpsEnvy does not provide detailed, technology-specific implementation steps; instead, it prioritizes the simulation of roles and processes.

In both cases, there is a gap associated with the specific selection of tools that clearly guide the learning process. Furthermore, they lack gradual implementation, which could lead to some difficulties in teaching such practices, their implementation, and their interconnectedness.

Likewise, Bruel and Jiménez, in their education panel, highlight the ongoing challenges in teaching DevOps, such as the need for meaningful feedback and the difficulty of balancing automation with pedagogical goals [6]. Their analysis underscores the importance of hands-on experience but stops short of offering a granular, staged methodology for tool adoption and process automation.

Likewise, Hobeck et al. [7] contribute further by comparing DevOps teaching approaches at two universities, focusing on the integration of CI/CD pipelines and the use of industry tools like Jenkins and Ansible. Their work demonstrates the value of exposing students to real-world tool chains and deployment scenarios. Nevertheless, their methodology is largely centered on graduate-level courses. It does not provide a phased, incremental adoption model suitable for diverse educational contexts or students at earlier stages of their training. This differentiates it from the proposal of this work.

In contrast to prior works, the guide proposed in this paper offers a structured, step-by-step methodology that explicitly details the adoption of DevOps practices using a specific tool chain, including GitHub, SonarCloud, and JUnit. Each stage—from version control and manual code review to continuous integration, static code analysis, and full pipeline automation—is accompanied by concrete instructions and scripts, enabling immediate application in academic projects. This approach not only demystifies the implementation process for students but also ensures that quality assurance mechanisms are embedded in each stage of development. Furthermore, the guide has been validated through its application in ten parallel student projects. Its scalability and effectiveness in improving software quality and development efficiency in an educational setting was proven.

By bridging the gap between abstract DevOps principles and practical, tool-based implementation, this guide addresses the limitations identified in previous educational frameworks. Additionally, it provides educators and students with a reproducible model for phased DevOps adoption, supporting competency-based learning and continuous quality improvement throughout the software development lifecycle.

This article presents an implementation guide of the five best practices for software development recommended by DevOps used in a software engineering course as a learning strategy. The article is organized as follows: Section 2 describes the methodology, Section 3 shows the results, and Section 4 presents the conclusions and future work.

2. METHOD

Software development processes are made up of phases such as analysis and planning, design, development, and quality, as well as the deployment and maintenance of applications [18]. This last phase could include having to cycle again in which changes or improvements are analyzed, new elements are designed within the solution, implemented, validated, and finally deployed again [18]. When executing a software development project, this phased lifecycle specifically guides what must be done to build the desired application. However, to fully comply with each phase, it is necessary to use a methodology, for example. Rational Unified Process – RUP is a development framework like Scrum, XP, or a combination of frameworks (Scrum + DevOps, Iconix, or similar). The above aims to delimit the roles, artifacts, events, and practices to be carried out during the development process.

DevOps focuses on the development, quality, deployment, and maintenance phases [3], applying specific practices that allow controlling the evolution of the increment, managing changes, creating quality filters associated with moments such as development, preventive tests prior to the creation of the deployable unit, integration of the changes made by all developers, creation of the deployable unit, release and operation, functional test, continuous monitoring of its operation, and generation of information that allows planning how to continue improving the development process [19].

The step-by-step model proposed in this article is based on the integration of DevOps practices through an automated cycle that consists of the following key steps mentioned in [20]. Specifically in the development and quality phases.

In the *first step*, *source code versioning* works as the starting point to manage the evolutionary changes of the software under construction. It is the starting point that will allow the detection of the incorporated changes and trigger the cycle. For this, changes cannot be entered directly into a branch of integration, so it is required to take *the second step*, where a developer, through a *pull request*, will ask to include these changes. Here, another member of the team must perform a *manual inspection of the code*; if it is approved, they merge it with the integration branch, and *the third step* is triggered, which is automation *with a pipeline* that starts building the code depending on the instructions associated with the programming language [16]. To achieve this, it is necessary to run the unit tests and then try to build and create the deployable unit. If this is successful, the *fourth step* would be the *static analysis of code* using an external tool such as Sonar Cloud to measure elements such as security, possible errors (bugs), vulnerabilities, code duplicity, and odorous codes; i.e., the use of bad programming practices that increase technical debt. Finally, in *the fifth step*, *all this is implemented in a script* that allows, from the moment a change is detected, to automatically chain all practices, identifying whether the objectives of each step are met and generating useful information for continuous improvement before obtaining sprint results. The above is summarized in Fig. 1. Each step is detailed below.

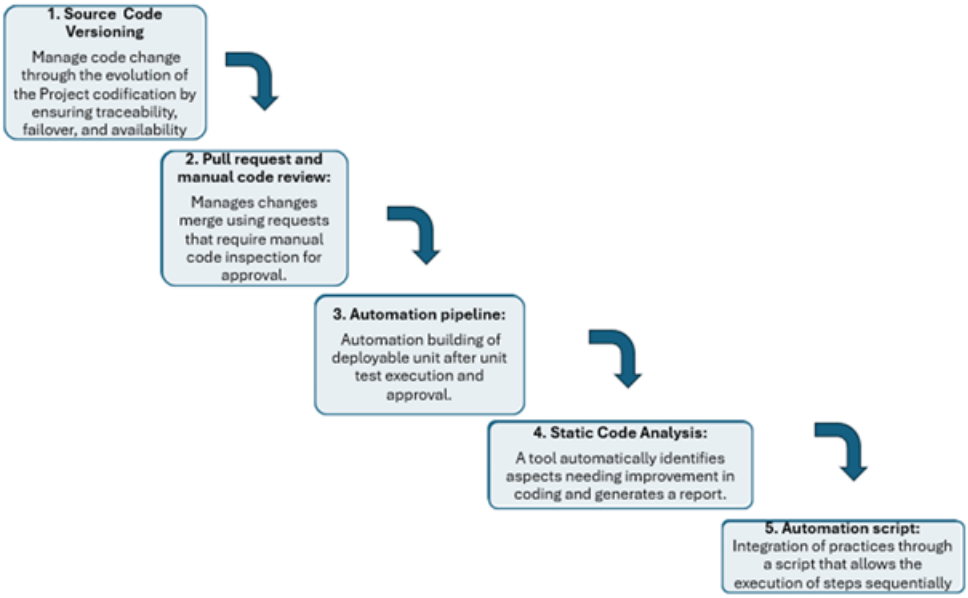

Fig. 1. *Steps to successful implementation of DevOps practices.*

2.1. Source Code Versioning

According to [20] using a version control system like Bitbucket, GitHub, and Gitlab is essential to managing software's evolutionary changes. Code versioners allow a detailed track of all modifications made to the code, identifying who made them, when, and what has been modified. They also facilitate collaboration between developers and ensure a clear and complete project history.

Steps to be taken when using the versioner:

1. Initialize the Repository: Developers must create a repository using a tool like GitHub, GitLab, or Bitbucket. This initializes the project and provides a centralized, highly available space to store the source code.
2. Branch Structure: It is necessary to define a clear branch structure that includes:

   2.1 Create a main branch (called main/master), where the most stable and final version of the code will be blocked for direct changes and where changes are only uploaded when the delivery is approved.

   2.2 Create an integration branch of development changes (it can be called develop, development, integration, etc.); it must be blocked to direct changes, and integration requests can only be made to this branch by pull request.

   2.3 Finally, developers work on their branches derived from the integration branch, where they manage their developments.
3. Commits and Branches: Developers create branches from the integration branch to implement new features or fix bugs. Each change is committed regularly, and a detailed record of the modifications is kept. The comments have a structure where the one that has been made is indicated as follows: *Addition of*, to include new functionalities; *Modification of*, to detail that something has been changed

in existing functionality; and *Correction of*, to comment that it is an adjustment either because it was detected during development or after it. Additionally, no final periods or ellipses should be used at the end of the comment because it would denote that it is incomplete, and the comment should be at most fifty (50) characters.

2.2 Pull Requests and Code Review

The work of [21] recommends using Pull Requests (PRs) to maintain control over change evolution. PRs are a fundamental practice for ensuring the quality of code. Through PRs, developers review changes before being integrated into the project's main branch.

Steps:

1. Creating the Pull Request: The developer creates a PR in the version control tool, describing the changes made and their purpose.
2. Manual Code Inspection: Another team member must review the PR to ensure it meets quality standards and does not introduce errors or vulnerabilities according to the company's development standards. This review may include comments and requests for changes if required.
3. Approval and Merging: If the revision is successful, the PR is approved, and changes will be merged into the integrations branch, triggering the next step in the automated pipeline.

2.3 Automation with Pipeline

An automated pipeline manages the creation (building) of the deployable unit, pre-executing the tests of the code as a quality control point, ensuring that the code has a first seal of quality performed by the software developer by validating correctness and handling errors in addition to determining if there are no syntactic errors that allow the construction according to [22]. This increases efficiency in the development process, allowing for a decrease in post-delivery rework. The pipeline is configured to run automatically whenever changes are merged into the integrations branch.

Steps:

1. Unit Test Execution: The pipeline automatically executes unit tests to verify that new features or modifications to existing ones do not introduce errors in the code. These tests are essential to ensure that each part of the code works correctly and maintains the system's operability in case of failures by controlling possible errors.
2. Code Construction: If unit tests are successful, the code is built using the specific instructions of the programming language with which the project is developed. Building the code converts the source code into an executable or package ready to be deployed.
3. Deployable Unit Creation: If the building is successful, a deployable unit (artifact) can be used in test or production environments. This artifact is the result of the construction process and is ready to be deployed or distributed, leaving the door open to include other practices such as continuous deployment.

2.4 Static Code Analysis - SCA

The Static Code Analysis (SCA), according to [20], uses tools that can be implemented in the development environment with measurements only on programming practices or in external tools that review the entire project, as is the case of SonarQube and SonarCloud, where many more rules are evaluated. They aim at measuring code's quality and reducing technical debt. This analysis can be integrated with the pipeline to ensure that each new code version meets the quality standards determined by the company. It can also serve as a threshold to identify whether it is ready to deliver. Steps:

1. SCA integration: Configure the pipeline to run SCA using a tool. This integration enables automatic and continuous evaluation of the code by measuring several aspects, which are consistently displayed on a dashboard accessible to the development team. This visibility encourages a culture of continuous improvement throughout the development process.
2. Execution of the Analysis: The pipeline executes static analysis, evaluating security, bugs, vulnerabilities, code duplication, and code smells. By providing this information to the team, all modifications required to increase the delivery quality will be executed before the sprint's completion.
3. Review of Results: The team reviews the reports generated by SonarCloud and takes corrective action if necessary. Detailed reports allow them to identify and correct problems before implementing improvements becomes more costly.

2.5 Automation Script

All the previous steps must be taken to a script that enables the integration and chaining of the practices above, so that whenever a change is detected, the quality is measured and reported to the development team to take immediate actions that allow the best possible result, as suggested by [23].

3. RESULTS

Following the recommendations in [24], 10 identical software development projects were implemented and conducted by students from the Antonio José Camacho University Institution as part of a software engineering course. The objective was to create the backend of an application that allows them to consult and reserve books in the institutional library, limiting it to the university community only. Professors, students and administrators can consult the books available in the library's catalog, and those that are available can be reserved for use for a period of no more than two weeks, then they must return it, or a fine will be issued, which will prevent that person to reserve any other books until it is paid. This implies the management of books (creation, updating, consultation, and inactivation of their availability), loans (creation, updating, consultation, and deletion), and fines (creation, updating, consultation, and deletion). It exposes the API's available methods and communicates with the logical layer to send a request or receive a response. Finally, once the backend is created, the presentation layer allows API consumption. For this exercise, students focused on the backend creation and the application of versioning, unit testing, Continuous Integration, and Static Code Analysis. Projects start from the source code baseline with a REST services API archetype. The project includes a multilayer

architecture based on a model-view-controller (MVC) structure as the predominant structural pattern. It was developed in Java 17 with Spring Boot 3.2.2, Swagger 2 V 2.9.2, JPA, Lombok, Junit, and Jacoco, and the dependency manager was Gradle 8.5. Postgres 11 was used as a database, and Flyway migration was used to manage all the project's database scripts. Spring Initializr was used to generate the initial structure of the project. These technologies have been selected by the training process of students in previous courses, such as web software development, to facilitate learning within the course with prior knowledge.

The project is composed of a folder to store the model classes, which are in-memory representations of the database entities; these classes use the notation language provided by Lombok to autogenerate the empty constructor, the constructor with all the attributes, and the class's accessor methods (get and set). These classes are used by DAOs or data access classes, called a registry, which allows JPAs to inherit generic methods in operations with the database, thus making coding faster. In addition, the latter classes are also organized in their own folder, different from those of the models. Together, they represent the data access layer of the system.

Registries are consumed by business logic classes and named services by the Spring Boot framework. There is no direct usage relationship; instead, the dependency injection pattern optimizes resource usage in object creation. This represents the logical layer of the system. Fig. 2 summarizes the solution's architecture.

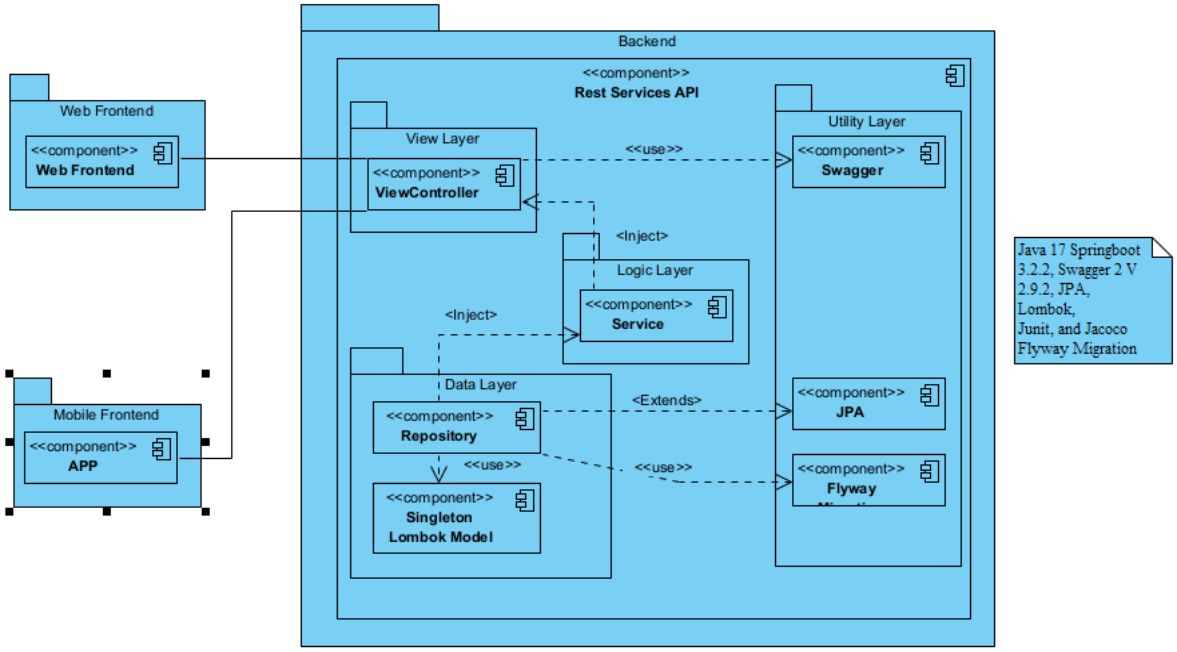

Fig. 2. *Solution's architecture.*

3.1 Source Code Versioning

The first step taken by all the students was to take the initial version of the project built into the code versioner, which contains complete functionality in which each class is implemented as an example of

how they should be implemented in other project functionalities. This is called an archetype. In this case, Bitbucket—a version control system based on Git—was used.

Using the tool properly requires a structure based on [15]. The model proposed in [20] suggested that a branch called master needs to contain the most stable version of the code in production. Also, a branch called "develop," derived from the master, must include the code prepared for the next version, continuously evolving until the quality has been checked. This branch is locked to changes directly, so making a Pull Request (PR) to include them is mandatory. Finally, from the last branch mentioned for integrations, the branches of each developer are derived, allowing each to develop the various functionalities in an organized way and have traceability of the changes. This allowed the teams to place an order and not overwrite changes to the work of the other members. A necessary recommendation for teams is to use clear and descriptive commit messages when pushing changes, following commit message conventions such as Conventional Commits. This facilitates the mandatory manual code inspection in the PR to approve the merge. The sample source code is available at https://bitbucket.org/Mpastrana/demo/src/master/ and can be cloned with a git bash using the command *git clone https://Mpastrana@bitbucket.org/Mpastrana/demo.git* (Fig. 3).

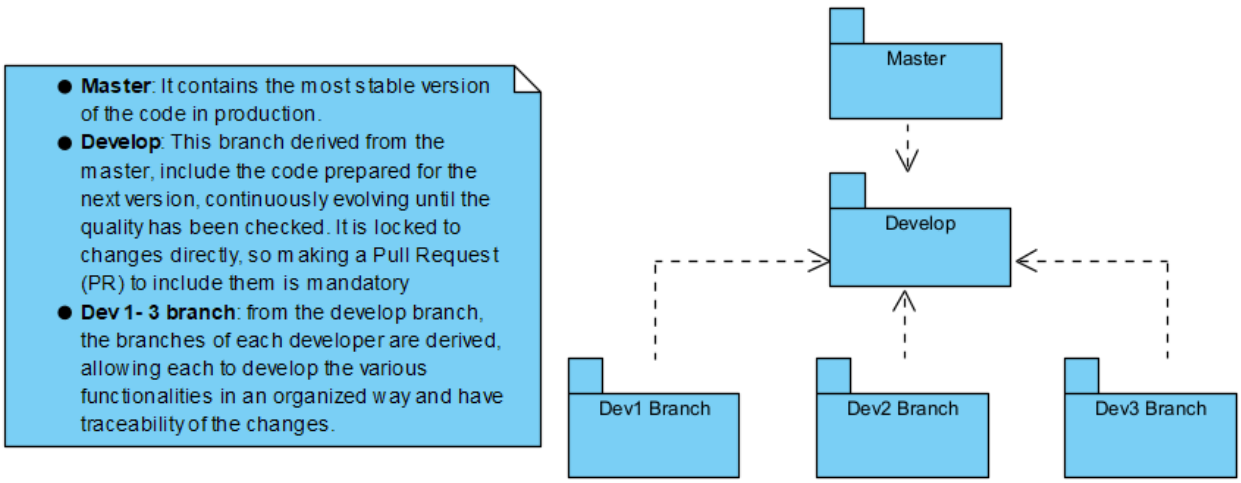

Fig. 3. *Branches' structure.*

All participating students successfully adopted version control practices, with *100% compliance* in using descriptive commit messages and following the established branching strategy. This uniformity in practices enhanced the overall project management and traceability of changes, leading to a more organized development process. By establishing a clear branching strategy and requiring pull requests for code integration, developers could work concurrently on different features without interfering with each other's work, thus facilitating smoother collaboration. This led to a *reduction in merge conflicts* among team members.

3.2 Pull Requests and Code Inspection

As mentioned in the method, it is necessary to implement this practice to take three steps: First, a team member who needs to include the changes in the integration branch (develop) must request the Bitbucket (*Pull Request - PR).* This assigns another team member to do the *manual code inspection*, which involves reviewing logic, architecture, and compliance with coding standards and providing detailed comments and suggestions for improvement if required. Finally, the reviewer of the code must *approve PR and merge* using pull requests for code inspection allowed for cleaner code submissions, making the review process faster and more efficient.

This allowed development teams to understand that the integration of the changes within the versioner should not be uploaded without a prior review to ensure that the modifications will not negatively affect the evolution of the project. By doing code inspections requested by the PR, students are more aware of the evolution of the project, they control syntactic errors or problems in coding that cause them to go out of the programming standard agreed upon by them, guaranteeing not only greater cleanliness and maintainability of the code, but also increasing communication and collaboration of the entire team. All the students applied this effectively.

3.3 Automation with Pipeline

Versioners like Bitbucket allow these practices to be integrated into the same tool and facilitate the learning process for students. Once the previous step is completed and approved, a script can be created to execute the **continuous integration**, which implies that all the project *unit tests* have been executed and have passed satisfactorily. If this does not happen, the process is interrupted, and the team is notified that it is impossible to do so. The objective is to have control over the evolution of the project at the syntactic level of the coding, detecting possible changes that generate errors to be corrected immediately. Then, when all the unit tests are approved, the *deployable unit's code construction (build)* is automated. Each commit in the integration branch of Bitbucket triggers this.

To do this, an initial script is explained in detail and created to guide all the students in this step. This must be placed at the root folder level and is called bitbucket-pipelines.yml. The script used is presented below:

```
1. image: amazoncorretto:17 # Docker image containing the required JDK
2. clone:
3.   depth: full # Does the full cloning of the branch download the latest version
4. definitions: # If caches are required for any tool, it is indicated here
5.   caches:
6.   steps:      # Steps to be executed
7.    - step: &build
8.        name: Build and Test
9.        caches:
10.         - gradle
```

```
11.      script:
12.      - chmod +x gradlew          # Grant gradle execute permissions
13.      - ./gradlew build           # Use gradle to build the project
14.    artifacts:
15.      - build/libs/**
16. pipelines:                       # Indicates when the step is executed
17.   branches:
18.    develop:
19.      - step: *build
20. pull-requests:
21.   '**':
22.     - step: *build
```

One of the advantages of using the Bitbucket tool is that it provides a template for the implementation of this step. So, it is useful for explaining how it can be done for this project or other projects with other technologies. Thanks to the use of the template, the explanation of the steps required by the script, and the detailed explanation of the elements that compose them, complexity is reduced and students were able to implement it satisfactorily.

3.4 Static Code Analysis

For this step, SonarCloud was used. This tool evaluates a detailed inspection of the code without executing it, providing an accurate evaluation of the quality of the code through different rules that allow evaluating vulnerability, security-based aspects, possible bugs, code duplicity, and bad coding practices in the project (code smell), as shown in Fig.4.

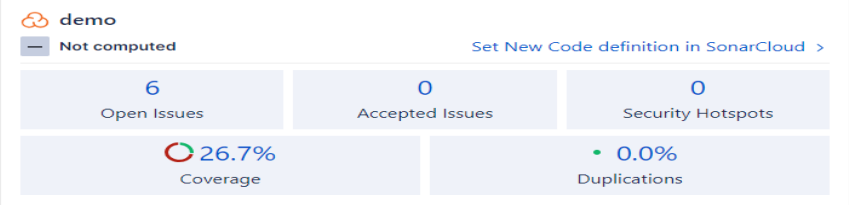

Fig. 4. *SonarCloud control panel view.*

To use SonarCloud, an account must be created at https://sonarcloud.io/login, using the Bitbucket account credentials to link the two tools using a private key. Once the account is created, a next-next option form is followed to guide the configuration of the analysis project, starting by requesting to select the repository from the versioner.

Finally, SonarCloud requests to indicate which analysis method will be used to integrate the static code analyzer with the tool that is doing the continuous integration (CI). This is done to suggest how to tell the CI script that it should add the step of calling this tool, as shown below in Fig. 5.

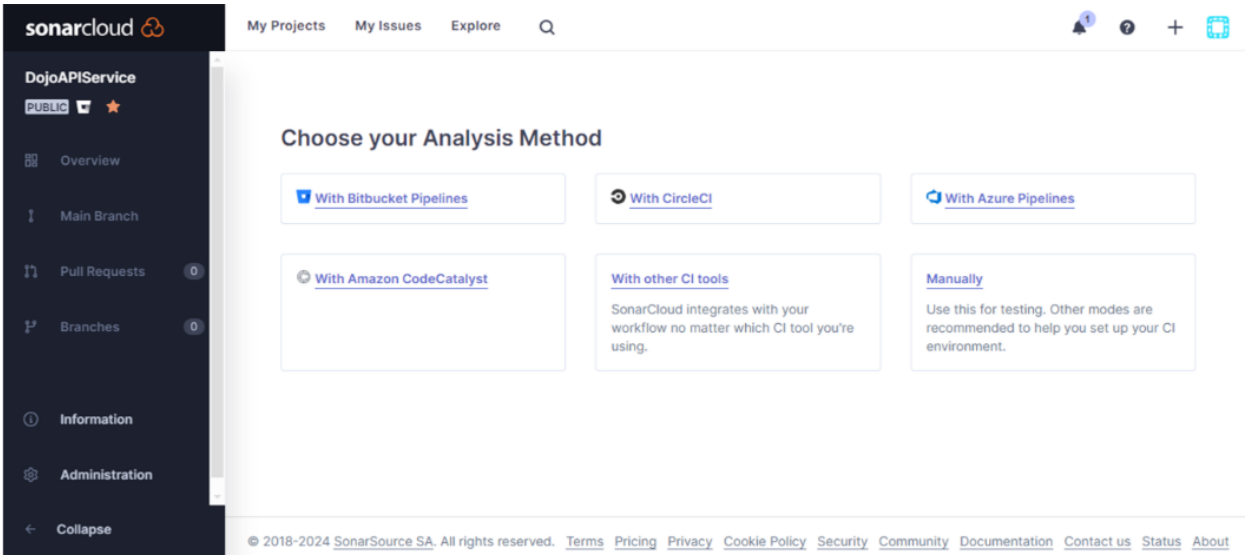

Fig. 5. *Selection of the tool for automation of integration through pipelines.*

To create the key, they must go to the repository in the Bitbucket tool. As this is the first time, the tool will prompt them to activate the pipeline to trigger the practices from the approved merger of changes in the integrations branch. Once the above is done, the environment variable is created, as shown in Fig. 6.

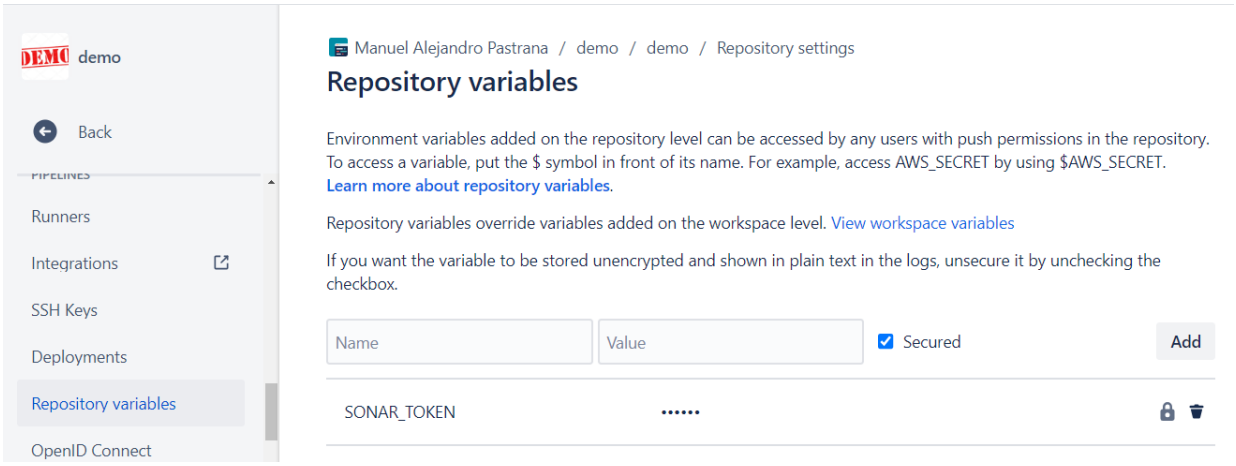

Fig. 6. *Variable environment creation for the tool-to-tool linking key.*

By integrating source code versioning, pull requests, and static code analysis into the pipeline, the team could identify potential issues early in the development process, reducing the likelihood of defects in the production environment, which is in line with what was stated by [25].

One of the positive aspects of using SonarCloud in the exercise is that, like BitBucket, the tool provides a template to modify the initial script, which makes it easy to include specific changes to adopt the tool. Therefore, to complete the exercise, students must modify the script as indicated in the next step and thus be able to perform the inspection automatically.

With the first linking of the project in the SonarCloud, the tool provides an initial inspection that tells teams their code quality with respect to aspects such as security, code duplication, cyclomatic complexity, and adjustments required in terms of the international coding standard, such as the proper use of constants instead of duplicating standard messages several times, do not leave defined variables, methods or libs unused, do not leave unnecessary comments, correctly control errors, among other elements that the tool reviews. Additionally, the tool indicates if there is the presence of unit tests and how much coverage they have on the code.

According to the exercise presented here, the dashboard provided by SonarCloud made it easier for students to detect the necessary improvements in their code, as well as to understand each suggestion made by the tool and how to solve it. This helps not only with continuous code improvement but also allows them to learn about the way the team is coding so as not to repeat the same thing in the future and generates a gradual improvement in the way they code their projects.

3.5 Automation Script

Finally, the bitbucket-pipelines.yml script is modified to include a SonarCloud review, where continuous code quality monitoring through version control, integration, and automated analysis allowed the team to address technical debt, improving the codebase's maintainability proactively. These practices allowed for early technical debt identification during the software engineering course, enabling the team to address potential issues before they escalated. This proactive approach not only improved code quality but also reduced the long-term costs associated with fixing accumulated technical debt, ultimately leading to a more maintainable and efficient codebase, which is consistent with the findings reported by [25]. The result is shown below.

1. image: amazoncorretto:17 # Docker image containing the required JDK

2. clone:

3. depth: full # Does the full cloning of the branch download the latest version

4. definitions: # If caches are required for any tool, it is indicated here

5. caches:

6. sonar: ~/.sonar/cache # Caching SonarCloud artifacts will speed up your build

7. steps: # Steps to be executed

8. - step: &build-text-sonarcloud

9. name: Build, test, and analyze on SonarCloud

10. caches:

```
11.         – gradle
12.         - sonar
13.       script:
14.       - chmod +x gradlew          # Grant gradle execute permissions
15.       - ./gradlew build           # Use gradle to build the project
16.     artifacts:
17.       - build/libs/**
16. pipelines:                        # Indicates when the step is executed
17.   branches:
18.     develop:
19.       - step: *build-text-sonarcloud
20. pull-requests:
21.   '**':
22.     - step: *build-text-sonarcloud
```

4. CONCLUSIONS AND FUTURE WORK

The article presents a step-by-step guide that allows unifying five practices proposed by DevOps: versioning with pull request to apply manual code inspection before integrating the changes, Unit Testing (UT), Continuous Integration (CI), Static Code Analysis, and an automation script, which integrates the practices.

Results from the ten student projects demonstrated tangible benefits from adopting this guide. Specifically, it promotes a preventive quality axis when carrying out software development projects. The combination of these practices allows software development teams to maintain control over the evolution of the project, identifying a history of changes and allowing the maintenance of a high-quality and stable version. In the same way, it allows the maintenance of standards and response quickly to changes in the software development environment, identifying whether the merged changes have brought syntactic errors that prevent the construction of the deployable unit or have modified the work of others without previously reviewing and adjusting it (through unit testing and continuous integration). Additionally, static code analysis plays a fundamental role in the codification quality, aiming to detect bad software development practices as soon as possible, teaching team members the best way to solve them, and promoting continuous improvement. Additionally, the feedback from the student teams indicated that these practices enabled them to identify and resolve quality issues early in the development cycle, before formal testing phases, as demonstrated by them in the results. This early detection was facilitated by the continuous feedback loop established through the automated pipeline, enabling students to iteratively improve their code quality.

Given the scarcity of detailed implementation guides with specific tools, this paper introduces a structured, easy-to-implement framework for development teams to adopt DevOps-recommended practices. Sequentially presenting the process facilitates a step-by-step understanding and gradual

adoption of each practice, helping teams to implement a software development process guided by DevOps, specifically in improving the development, testing, and deployment phases. This approach encourages teams to leverage the continuous flow of information generated during implementation, promoting proactive decision-making and responsiveness. As demonstrated in the results, maintaining rigorous control over project evolution enables teams to address unforeseen issues early rather than discovering them during the testing phases. This not only enhances project outcomes but also fosters a culture of automation and quality-focused practices, ultimately contributing to improved software development processes.

Furthermore, the implementation of continuous integration practices resulted in a more streamlined building process, allowing for faster feedback on code changes. Student feedback indicated that this rapid feedback cycle enabled them to address integration issues more efficiently, reducing debugging and resolving conflicts more easily due to the information provided by the implemented tools.

While the scope of this study is limited to an educational setting, results suggest that the proposed implementation guide offers a viable approach for introducing DevOps practices in a structured and incremental manner. The guide's step-by-step nature and tool-specific instructions facilitated the adoption of these practices by students with varying levels of prior experience, as mentioned in the results section.

Future work should focus on evaluating the guide's effectiveness in industrial settings and exploring its applicability to different types of software projects. Additionally, it is necessary to quantify the impact of each practice on specific quality metrics and to develop more comprehensive assessment methods for evaluating DevOps competencies. Additionally, it is recommended to create similar guides for other technologies and compare results to identify possible generic models that facilitate the practices' adoption and implementation. Likewise, in a real-world scenario or when applying the guide to large-scale systems with multiple microservices, cloud deployments, or multi-region architectures, it would be necessary to test its scalability.

AUTHORS' CONTRIBUTION

Manuel Pastrana: Conceptualization, Methodology, Writing – original draft.

Hugo-Armando Ordoñez-Erazo: Conceptualization, Methodology, Supervision, Writing – reviewing and editing.

Carlos-Alberto Cobos-Lozada: Supervision, Writing – reviewing and editing.

Mirna Muñoz: Supervision, Writing – reviewing and editing.

REFERENCES

[1] S. Hastie, S. Wojewoda, *Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch*, 2015. http://www.infoq.com/articles/standish-chaos-2015

[2] M. A. Pastrana, H. A. Ordoñez-Erazo, C. A. Cobos-Lozada, "Adaptando DevOps a la norma ISO 29110 a través de metodologías agiles en VSE desarrolladoras de software colombianas," *Investigación e Innovación en Ingenierías*, vol. 12, no. 1, pp. 189-203, jun. 2024. https://doi.org/10.17081/INVINNO.12.1.6916

[3] R. Jabbari, N. bin Ali, K. Petersen, B. Tanveer, "Towards a benefits dependency network for DevOps based on a systematic literature review," *Journal of Software: Evolution and Process*, vol. 30, no. 11, e1957, Nov. 2018. https://doi.org/10.1002/smr.1957

[4] A. Grotta, E. P. V Prado, "DevOpsBL: DevOps-based learning on information systems higher education," Federal Institute of Sao Paulo (IFSP), Brazil: Association for Information Systems, 2021.

[5] G. Rong, S. Gu, H. Zhang, D. Shao, "DevOpsEnvy: An Education Support System for DevOps," in *IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, 2017, pp. 37-46. https://doi.org/10.1109/CSEET.2017.17

[6] J.-M. Bruel, M. Jiménez, "Devops'18 education panel: Teaching feedback and challenges," in *Lecture Notes in Computer Science*, 2019. https://doi.org/10.1007/978-3-030-06019-0_17

[7] R. Hobeck, I. Weber, L. Bass, H. Yasar, "Teaching DevOps: A tale of two universities," in *Proceedings of the 2021 ACM SIGPLAN International Symposium on SPLASH-E*, 2021, pp. 26-31. https://doi.org/10.1145/3484272.3484962

[8] K. Kuusinen, S. Albertsen, "Industry-academy collaboration in teaching devops and continuous delivery to software engineering students: Towards improved industrial relevance in higher education," in *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 2019, pp. 23-27. https://doi.org/10.1109/ICSE-SEET.2019.00011

[9] R. Jabbari, N. Bin Ali, K. Petersen, B. Tanveer, "What is DevOps? A Systematic Mapping Study on Definitions and Practices," in *Proceedings of the Scientific Workshop*, 2016. https://doi.org/10.1145/2962695.2962707

[10] M. A. Pastrana Pardo, H. A. Ordóñez Erazo, C. A. Cobos Lozada, "Approach to the Best Practices of Software Development Based on DevOps and SCRUM Used in Very Small Entities," *Revista Facultad de Ingeniería*, vol. 31, no. 61, e14828, Sep. 2022. https://doi.org/10.19053/01211129.V31.N61.2022.14828

[11] A. D. Robinson, "Very small entities (VSE); The final systems engineering (SE) frontier," in *12th Annual IEEE International Systems Conference, SysCon*, 2018, pp. 1-4. https://doi.org/10.1109/SYSCON.2018.8369570

[12] M. Senapathi, J. Buchan, H. Osman, "DevOps capabilities, practices, and challenges: Insights from a case study," in *ACM International Conference Proceeding Series*, 2018. https://doi.org/10.1145/3210459.3210465

[13] K. Maroukian, S. R. Gulliver, "The Link between Transformational and Servant Leadership in DevOps-Oriented Organizations," in *PervasiveHealth: Pervasive Computing Technologies for Healthcare*, 2020, pp. 21-29. https://doi.org/10.1145/3393822.3432340

[14] A. Belalcazar, "Incorporation of Good Practices in the Development and Deployment of Applications through Alignment of ITIL and Devops," in *International Conference on Information Systems and Computer Science (INCISCOS)*, Nov. 2017, pp. 224-230. https://doi.og/10.1109/INCISCOS.2017.31

[15] K. Schwaberm, J. Sutherland, *La Guía Definitiva de Scrum: Las Reglas del Juego*, 2020. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf

[16] S. Vadapalli, *DevOps: Continuous Delivery, Integration, and Deployment with DevOps Dive into the core DevOps strategies*, 2018. https://books.google.com.mx/books?id=N5RRDwAAQBAJ

[17] T. Pandiyavathi, B. Sivakumar, "DevOps Challenges and Practices in Software Engineering," *Lecture Notes in Networks and Systems*, vol. 665 LNNS, pp. 49-57, 2023. https://doi.org/10.1007/978-981-99-1726-6_5

[18] R. S. Pressman, B. R. Maxim, *Software Engineering: A Practitioner's Approach*, Boston, USA: McGraw-Hill, 2015.

[19] W. de Kort, "What Is DevOps?" in *DevOps on the Microsoft Stack*, Berkeley, CA: Apress, 2016, pp. 3-8. https://doi.org/10.1007/978-1-4842-1446-6_1

[20] M.-A. Pastrana-Pardo, H.-A. Ordóñez-Erazo, C.-A. Cobos-Lozada, "Process Model Represented in BPMN for Guiding the Implementation of Software Development Practices in Very Small Companies Harmonizing DEVOPS and SCRUM," *Revista Facultad de Ingeniería*, vol. 31, no. 62, e15207, Dec. 2022. https://doi.org/10.19053/01211129.v31.n62.2022.15207

[21] V. Ivanov, K. Smolander, "Implementation of a DevOps Pipeline for Serverless Applications," *Lecture Notes in Computer Science*, vol. 11271 LNCS, pp. 48-64, 2018. https://doi.org/10.1007/978-3-030-03673-7_4

[22] A. Agarwal, S. Gupta, T. Choudhury, "Continuous and Integrated Software Development using DevOps," in *International Conference on Advances in Computing and Communication Engineering*, 2018, pp. 290-293. https://doi.org/10.1109/icacce.2018.8458052

[23] P. Mumbarkar, S. Prasad, "Adopting DevOps: Capabilities, practices, and challenges faced by organizations," in *AIP Conference Proceedings*, 2022. https://doi.org/10.1063/5.0110594/2828181

[24] R. Chatley, I. Procaccini, "Threading DevOps Practices through a University Software Engineering Programme," in *IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, 2020, pp. 90-94. https://doi.org/10.1109/CSEET49119.2020.9206211

[25] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, B. Vasilescu, "The impact of continuous integration on other software development practices: A large-scale empirical study," in *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, Oct. 2017, pp. 60-71. https://doi.org/10.1109/ASE.2017.8115619