Increasing Quality in Software Development with DevOps: A Step-by-Step Guide

**Abstract.** Small companies have a crucial place in the software development chain. This fact challenges them to continuously search for technology that helps them make their development processes efficient. One way they are following is the implementation of agile practices or, more recently, the implementation of DevOps. However, most do not have the expected results due to the lack of information about how to do it in a continuous search for technology that adequately helps them. This paper provides a detailed guide for implementing an automated software development cycle using DevOps practices by integrating source code versioning, code review, automated build, and static code analysis. Implementing these practices aims to enhance software quality, reduce rework, and boost market competitiveness.

**Keywords:** DevOps, Continuous Integration, CI, Software Engineering, good practices, Software Development, Step-by-Step Guide.

1. Introduction

For the software development industry, both efficiency and quality in the products and services they offer are essential to stand out from the local and global competition [1]. However, the different companies' sizes and characteristics generate differentiating factors that facilitate or prevent the implementation of continuous improvement processes [2]. Larger companies can access more profitable projects and invest in specialized consultancies to implement best practices for software development that guarantee best results [3]. However, smaller companies cannot do this in the same way because they must take more care of their resources to ensure the operability and sustainability of the company, as indicated [4].

Therefore, this type of company is heading towards the market trend regarding standards, frameworks, and practices that have shown effectiveness in other companies [5]. Agile frameworks such as Scrum and DevOps, through their suggested practices, have proven their effectiveness in facing the various challenges of this competitive industry, as mentioned [6]. However, these frameworks only indicate a set of suggestions that, in the end, must be interpreted by the companies that want to implement them, leaving everything to a process of trial and error, which in most cases is very costly for medium and small companies [7]. Looking for a way to alleviate the exploratory process and as a practical and viable solution, this article proposes a step-by-step guide to implement an automated cycle that integrates the practices of versioning, construction automation, unit testing, continuous integration, and static code analysis, as a mechanism that seeks to better the quality of the software and the development process itself as a starting point in a path to achieve a DevOps environment. The rest of the document is organized as follow, section 2 shows the motivation scenario, section 3 detail the method used, section 4 displays the results and discussion, and section 5 exposes the conclusions.

1. Motivation Scenario

Software development companies, regardless of their size, face multiple challenges associated with their daily activity, such as managing technical debt, speed in product delivery, and the need to maintain high-quality standards that meet the needs of the market, in addition to ensuring customer loyalty [8]. To achieve this, the use of best practices for software development based on DevOps is critical because it allows the implementation of different preventive quality filters [9], such as control points and continuous feedback that serve the development team directly during the construction time, empowering those who build the product to quality [10]. This scenario, according to [11] allows collaborators to know at all times, during the increment of the deliverable, the actual status of the evolution 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, which is a time box that goes from 1 to 4 weeks where commitments are agreed, and feedback is obtained from the interested parties at the end of it, as mentioned [12].

Although frameworks such as DevOps propose a set of best practices that can be applied manually in the initial stages as suggested by [13], 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 [14] and [15].

1. Method

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 vital steps mentioned in [16]:

In the **first step, source code versioning** works as the starting point where the evolutionary changes of the software under construction are managed. It is the starting point that will allow the detection of the incorporated changes and triggering the cycle. For this, it is necessary that the changes cannot be entered directly into a branch of integrations, so it is required to execute **the second step**, where a developer, through a pull request, will request to include these changes. Here, another member of the team must perform a **manual inspection of the code**; if it is approved, they merge with the integration branch, and **the third step** is triggered, which is automation **with a pipeline** that starts first to build the code depending on the instructions associated with the programming language [14]. 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** to be carried out would be the **static analysis of code** using an external tool such as Sonar Cloud to measure elements such as security, the possibility of errors (bugs), vulnerabilities, code duplicity, and odorous codes that are 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 the delivery of the sprint results. The above is summarized in Fig. 1. Each step is detailed below.

Diagrama

Descripción generada automáticamente

**Fig. 1** Step by step for the implementation of DevOps practices. Own source

* 1. Source Code Versioning

According to [16] Using a version control system, such as 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 makes them, when, and what has been modified. They also facilitate collaboration between developers and ensure a clear and complete project history.

**Steps required in using the versioner:**

1. **Initialize the Repository**: You 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:
   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. 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.
   3. Finally, developers work on their branches derived from the integrations branch, where they manage their developments.
3. **Commits and Branches**: Developers create branches from the integrations 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 fitty (50) characters.
   1. Pull Requests and Code Review

The work of [17] 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 request in the version control tool, providing a detailed description of the changes made and the purpose of the changes.
2. **Manual Code Inspection**: Another team member must review the PR to ensure it meets quality standards and does not introduce errors or vulnerabilities per 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 the changes will be merged into the integrations branch. This merge triggers the next step in the automated pipeline.
   1. Automation with Pipeline

An automated pipeline manages the creation (build) 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 [18]. 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 build is successful, a deployable unit (artifact) is created that 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.
   1. Static Code Analysis - AEC

Static Code Analysis (AEC), according to [16] It 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. All this aims to measure the code's quality and reduce 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 the delivery is.

**Steps:**

1. **AEC integration**: Configure the pipeline to run AEC using a tool. This integration allows for automatic and continuous evaluation of the code by measuring several aspects and always displaying them in a dashboard available to the development team. This encourages continuous improvement as part of the work culture during development.
2. **Execution of the Analysis**: The pipeline executes the static analysis, evaluating security, bugs, vulnerabilities, code duplication, and code smells. By providing this information to the team, all modifications required to increase the deliverable's quality will be executed before the sprint's completion.
3. **Review of Results**: The team reviews the reports generated by SonarCloud and takes corrective actions if necessary. Detailed reports allow you to identify and correct problems before implementing improvements become more costly.
   1. Automation Script

All the previous steps must be taken to a script that allows 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 like is suggested by [19].

1. Results and Discussion

Following the recommendations in [20], eight software development projects were implemented and conducted by students from Antonio José Camacho University Institution as part of a seminar course. Projects start from a baseline of source code that contains an archetype-level REST services API. The project includes a model view controller (MVC) structure as the predominant structural pattern. It is built under the Java 17 programming language and the Springboot 3.2.2, Swagger 2 V 2.9.2, JPA, Lombok, Junit, and Jacoco frameworks. Postgres 11 was used as a database, but thanks to JPA, it can be easily migrated to any other version or even another database manager. Gradle 8.5 is used as the dependency manager.

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 registry, which allows JPAs to inherit generic methods in operations with the database, making coding faster. In addition, these latter classes are also organized in their 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 SpringBoot frame. 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.

Finally, the presentation layer class allows API and consumer interaction. It exposes the API's available methods and communicates with the logical layer to send a request or receive a response. Fig. 2 summarizes the structure indicated by the IDE (Integrated Development Environment).

Texto

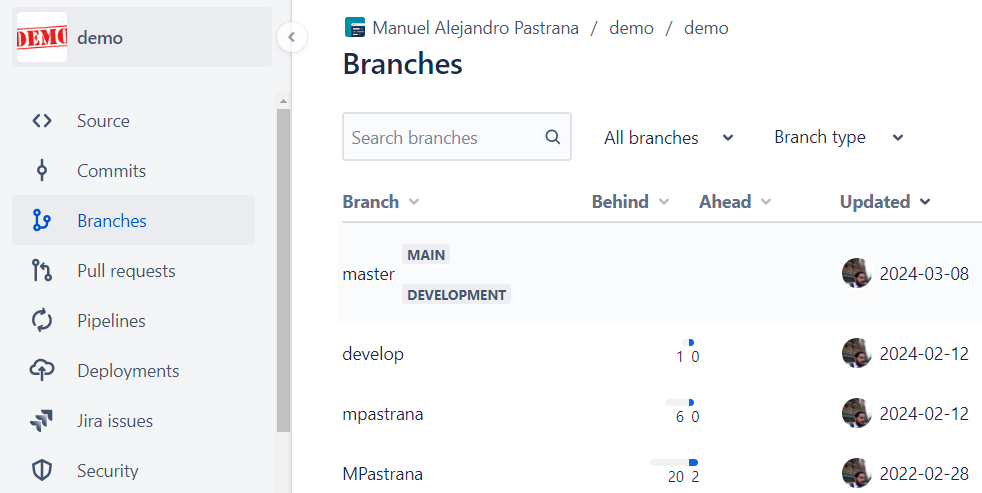
Descripción generada automáticamente

**Fig. 2** Example project structure. Own source

* 1. Source Code Versioning

With the previous project already created, the first implementation step is to take it to the code versioner. As mentioned earlier, version control is fundamental in modern software development. In this case, the BitBucket verifier, a distributed version control system based on Git, was used. It allows developers to work simultaneously on distinct parts of the project without interfering with each other.

To workflow with the tool, a structure based on [16] is required, indicating a main branch called main/master that contains the code in production, the most stable version. A so-called "develop" branch is derived from this branch, which includes the code prepared for the next version, continuously evolving until the quality has been checked. This branch is locked to pull changes directly, so making a Pull Request (PR) to include them is mandatory. Finally, from that branch comes one so that each developer can manage their changes, work from the baseline, and have a history of their evolution controlled in the tool. This allows the system to have an order and not overwrite changes to the work of the other team 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 in the git clone <https://Mpastrana@bitbucket.org/Mpastrana/demo.git>. Fig. 3 represents the structure mentioned above.



**Fig. 3** Branches created in the versioner, for example, guide. Own source.

* 1. Pull Requests and Code Inspection

Inspecting code through PR improves code quality and encourages knowledge sharing among team members. To use this practice, it is suggested to do several things: First, perform an automatic analysis through tools such as linters and code style analysis that provide feedback so that the code arrives at the request as cleanly as possible and according to the language standard. Second, perform a manual inspection of code where you seek to review logic, architecture, and compliance with coding standards. Another team member does this, other than the one who made the request. Finally, provide detailed comments and suggestions for improvement if required. It is essential to understand that PR will only merge the changes in the integrations branch if accepted. Fig. 4 shows an example of what an approved PR request would look like in the Bitbucket tool.



**Fig. 4** Example PR. Own source

* 1. Automation with Pipeline

This step is conducted with the objective that once the merge of a change in the integrations branch (develop) is approved, using PR, a script is triggered that allows the execution of the continuous integration to be carried out. Therefore, each commit triggers a process of automatic construction and testing using instructions specific to the programming language. Versioners such as Bitbucket, GitHub, and GitLab allow these practices to be integrated into the same tool. The script to be created must be placed at the root folder level and is called bitbucket-pipelines.yml. Here, we will put the following instructions on using the dependency manager to execute the project's unit tests and, after its approval, try to create the deployable unit. The script is detailed below.

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

*2. clone:*

*3. depth: full # Does the full cloning of the branch to 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*

* 1. Static Code Analysis

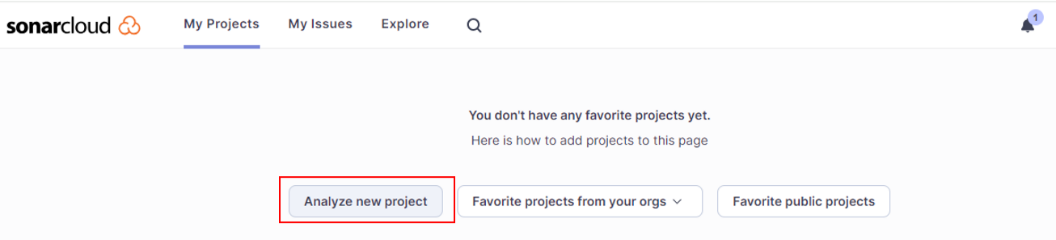
To use a Static Code Analysis (AEC) tool, such as SonarCloud, which was used in this project, it is only necessary to have the project in the code versioner. This tool evaluates the code without executing it, providing a quick and accurate evaluation of the quality of the code through different rules that allow evaluating vulnerability and security-based aspects, possible bugs, code duplicity, and bad coding practices in the project (odorous code). The above can be seen in Fig.5.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

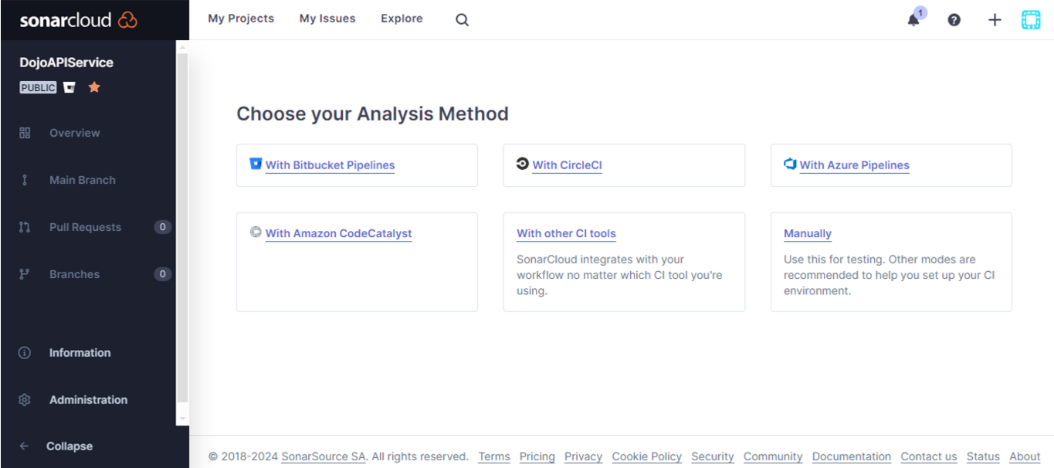
**Fig. 5** Sonar control panel. Own source

The tool requires creating an account at https://sonarcloud.io/login. Once the account is created, a next-next option form is followed for the configuration. This starts by requesting to create the project, as in Fig. 6. You must have access to the repository to see the available projects. There, the project is selected, and the selection is indicated as new.



**Fig. 6** Project creation in SonarCloud. Own source.

Finally, this tool indicates that you must select which tool it will be integrated with so that it runs automatically when continuous integration is done, as indicated below in Fig. 7:



**Fig. 7** Selection of integration tool for automation. Own source.

* 1. Automation Script

Following and complementing the previous script is necessary to chain all the practices, including the AEC mentioned above. Since the last point described in the Static Code Analysis step indicates that the tool in which this point is integrated must be selected, Bitbucket was selected, which is the versioner used. This will require integration with the tool using a key that links them and, on the other hand, the modification of the IC script.

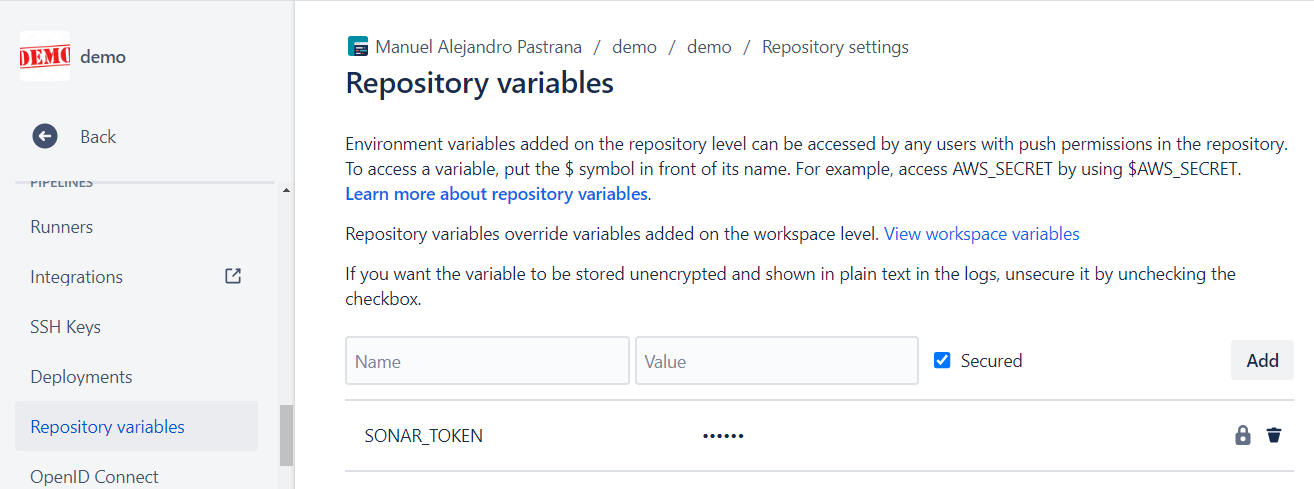
To create the key, you must go to the repository in the Bitbucket tool. If this is the first time you have done this, the tool will prompt you to activate the pipeline to allow you to trigger the practices from the approved merge of changes in the integrations branch. See Fig. 8.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Fig. 8** Pipeline activation in bitbucket. Own source.

The environment variable is created once the above is done, as shown in Fig. 9.



**Fig. 9** Variable environment creation for the tool-to-tool linking key. Own source.

Finally, the bitbucket-pipelines.yml script is modified to include a sonar review. The result is as follows:

1*. image: amazoncoretto:17*

*2. clone:*

*3. depth: full*

*4. definitions:*

*5. caches:*

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

*7. steps:*

*8. - step: &build-test-sonarcloud*

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

*10. caches:*

*11. - gradle*

*12. - sonar*

*13. script:*

*14. - chmod +x gradlew*

*15. - ./gradlew build sonar*

*16. artifacts:*

*17. - build/libs/\*\**

*18. pipelines:*

*19. branches:*

*20. develop:*

*21. - step: \*build-test-sonarcloud*

*22. pull-requests:*

*22. '\*\*':*

*23. - step: \*build-test-sonarcloud*

* 1. Benefits of Implementation

The combination of these practices allows organizations first to maintain control over the evolution of the project, identifying a history of changes made and allowing them to maintain a stable version always, in addition to controlling when and with what quality the merging of the work of the team members is accepted. On the other hand, it allows you to maintain high-quality standards and respond 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). Likewise, the AEC will conduct a thorough review that will reduce reprocessing in the future.

1. Conclusions

Integrating DevOps practices into the software development process, combined with an automated cycle of code construction, testing, and static analysis, is an effective strategy for improving software quality. By adopting this approach, development teams can reduce technical debt, increase efficiency, and strengthen their competitive position in the market.

Implementing practice chaining through an execution automation script, such as the one in the step-by-step guide, provides multiple benefits for software development processes. The results observed in the application within the refresher seminar course include a decrease in rework by detecting and correcting errors in the early stages of development, thus reducing the cost and time spent on fixing this in the future. Using these best practices also encourages continuous improvement through the constant feedback generated by quality analysis delivered by the tools, ensuring that the software evolves and improves over time. Finally, it promotes the competitiveness of work teams by maintaining a high standard of quality in the software developed, which will positively affect the companies that adopt it. High-quality products attract more customers and allow businesses to stand out.

References

[1] Y. Khmelevsky, X. Li, and S. Madnick, “Software development using agile and scrum in distributed teams,” in *11th Annual IEEE International Systems Conference, SysCon 2017 - Proceedings*, 2017. doi: 10.1109/SYSCON.2017.7934766.

[2] M. Muñoz and M. Negrete, “Reinforcing DevOps Generic Process with a Guidance Based on the Basic Profile of ISO/IEC 29110,” in *Advances in Intelligent Systems and Computing*, Springer., Springer, Ed., Springer Cham, 2020, pp. 65–79. doi: 10.1007/978-3-030-33547-2\_6.

[3] C. Y. Laporte and R. V. O’Connor, “Implementing process improvement in very small enterprises with ISO/IEC 29110: A multiple case study analysis,” in *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016*, 2017, pp. 125–130. doi: 10.1109/QUATIC.2016.033.

[4] A. Hemon, B. Fitzgerald, B. Lyonnet, and F. Rowe, “Innovative Practices for Knowledge Sharing in Large-Scale DevOps,” *IEEE Softw*, vol. 37, no. 3, pp. 30–37, 2020, doi: 10.1109/MS.2019.2958900.

[5] C. Ebert and M. Paasivaara, “Scaling Agile,” *IEEE Software*. 2017. doi: 10.1109/MS.2017.4121226.

[6] M. A. Pastrana, H. A. Ordoñez, and C. Cobos, “ISO 29110 en Colombia: de la teoría a la práctica,” *Revista Guillermo de Ockham*, vol. 17, no. 2, pp. 71–80, Dec. 2019, doi: 10.21500/22563202.4299.

[7] H. Kniberg, “Scrum and XP from the Trenches,” *Reading*, 2007, doi: 978-1-4303-2264-1.

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

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

[10] K. Maroukian and 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. doi: 10.1145/3393822.3432340.

[11] A. Belalcazar, “Incorporation of Good Practices in the Development and Deployment of Applications through Alignment of ITIL and DevOps,” in *2017 International Conference on Information Systems and Computer Science (INCISCOS)*, IEEE, Nov. 2017, pp. 224–230. doi: 10.1109/INCISCOS.2017.31.

[12] K. Schwaber and J. Sutherland, “La Guía de Scrum. La Guía Definitiva de Scrum: Las Reglas del Juego.,” 2020, Accessed: Sep. 29, 2022. [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf

[13] J.-M. Bruel and M. Jiménez, “Devops’18 education panel: Teaching feedback and challenges,” vol. 11350 LNCS. Springer Verlag, University of Toulouse, IRIT, Toulouse, France, pp. 221–226, 2019. doi: 10.1007/978-3-030-06019-0\_17.

[14] S. Vadapalli, *DevOps: Continuous Delivery, Integration, and Deployment with DevOps Dive into the core DevOps strategies*. 2018. Accessed: Jun. 02, 2021. [Online]. Available: https://books.google.com.mx/books?id=N5RRDwAAQBAJ

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

[16] M.-A. Pastrana-Pardo, H.-A. Ordóñez-Erazo, and 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, pp. e15207–e15207, Dec. 2022, doi: 10.19053/01211129.V31.N62.2022.15207.

[17] V. Ivanov and K. Smolander, “Implementation of a DevOps Pipeline for Serverless Applications,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11271 LNCS, 2018, pp. 48–64. doi: 10.1007/978-3-030-03673-7\_4.

[18] A. Agarwal, S. Gupta, and T. Choudhury, “Continuous and Integrated Software Development using DevOps,” 2018. doi: 10.1109/icacce.2018.8458052.

[19] P. Mumbarkar and S. Prasad, “Adopting DevOps: Capabilities, practices, and challenges faced by organizations,” *AIP Conf Proc*, vol. 2519, no. 1, Oct. 2022, doi: 10.1063/5.0110594/2828181.

[20] R. Chatley and I. Procaccini, “Threading DevOps Practices through a University Software Engineering Programme,” D. M., H. E., K. S., B. B., and T. B., Eds., Dept of Computing, Imperial College London, United Kingdom: Institute of Electrical and Electronics Engineers Inc., 2020, pp. 90–94. doi: 10.1109/CSEET49119.2020.9206211.