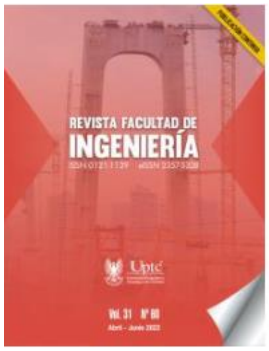
Revista Facultad de Ingeniería
Journal Homepage: https://revistas.uptc.edu.co/index.php/ingenieria



# Approach to the Best Practices of Software Development Based on DevOps and SCRUM Used in Very Small Entities

Manuel-Alejandro Pastrana-Pardo[1]
Hugo-Armando Ordóñez-Erazo[2]
Carlos-Alberto Cobos-Lozada[3]

**Received:** May 11, 2022 **Accepted:** September 22, 2022 **Published:** September 28, 2022

**Citation:** 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 (61), e14828, 2022. https://doi.org/10.19053/01211129.v31.n61.2022.14828

## Abstract

Very small software development entities have a maximum of 25 employees and have limited cash flow and time to implement improvements in their processes that

[1] Institución Universitaria Antonio José Camacho (Cali-Valle del Cauca, Colombia). mapastrana@admon.uniajc.edu.co. ORCID: https://orcid.org/0000-0002-6506-0659
[2] Ph. D. Universidad del Cauca (Popayán-Valle, Colombia). hugoordonez@unicauca.edu.co. ORCID: https://orcid.org/0000-0002-3465-5617
[3] Ph. D. Universidad del Cauca (Popayán-Valle, Colombia). ORCID: https://orcid.org/0000-0002-6263-1911

allow them to be more competitive. This is one of the reasons why these companies turn to the implementation of agile frameworks such as SCRUM to manage the software development process. But when they start their adoption, they find that the documents only suggest the changes that can be made, but not how to make them, making the process of discovering which techniques, events and artifacts are the ones to implement in a costly trial and error approach and in some cases unfeasible. The same happens with other frameworks that can be complementary to SCRUM, such as DevOps, which proposes a rapprochement between the development and operations area, where the largest number of tasks are automated and quality controls are increased to obtain better products. This article exposes three best practices based on DevOps, its use models and when within SCRUM they can be used to facilitate its adoption in these companies. Present a model for the use of versioning, integration, and deployment continuous and the recommended moments for its implementation within SCRUM. The best practices most reported in the literature for software development based on SCRUM and DevOps were identified. Three of the best practices were selected and a usage model was built for each of them. These practices were evaluated using a case study and the results obtained were evaluated. The practices were evaluated in three (3) very small entities, obtaining changes in the support cases reported weekly and in the number of successful deployments. The division of the development process into phases shows that the one that represents the greatest possibility of splicing between the set of practices suggested by DevOps in SCRUM is the development and quality phase. Likewise, the set of suggested practices points to the implementation of controls for quality assurance, providing key information for the learning and improvement of the development team.

**Keywords:** DevOps; Software Quality Assurance; Software Engineering; SCRUM; SQA.

## Acercamiento a las buenas prácticas para el desarrollo de software basado en DevOps y SCRUM utilizadas en empresas muy pequeñas

## Resumen

Las empresas muy pequeñas de desarrollo de software poseen un máximo de 25 empleados y tienen un limitado flujo de caja y tiempo para implementar mejoras en sus procesos que les permita ser más competitivos. Esta es una de las razones por las que estas empresas recurren a la implementación de marcos de trabajo ágil como SCRUM para gestionar el proceso de desarrollo de software. Pero cuando inician su adopción, encuentran que los documentos solo sugieren los cambios que se pueden realizar, pero no como hacerlos, tornando el proceso de descubrir cuales técnicas, eventos y artefactos son los que deben implementar en un enfoque de prueba y error costoso y en algunos casos inviable. Lo mismo sucede con otros marcos que pueden ser complementarios a SCRUM como DevOps, que propone un acercamiento entre el área de desarrollo y operaciones, donde se automaticen la mayor cantidad de tareas y se incrementen los controles de calidad para obtener mejores productos. Este artículo expone tres buenas prácticas basadas en DevOps, sus modelos de uso y en qué momentos dentro de SCRUM pueden ser utilizadas para facilitar su adopción en estas empresas. Se tiene como como objetivo exponer un modelo para el uso de versionamiento, integración y despliegue continuos y los momentos recomendados para su implementación dentro de SCRUM. Se identificaron las buenas prácticas más reportadas en la literatura para desarrollo de software basado en SCRUM y DevOps. Se seleccionaron tres de las mejores prácticas y se construyó un modelo de uso para cada una de ellas. Estas prácticas se pusieron a prueba mediante un caso de estudio y se evaluaron los resultados obtenidos. Las prácticas fueron evaluadas en 3 empresas, obteniendo cambios en los casos de soporte reportados semanalmente y en el número de despliegues exitosos. La división del proceso de desarrollo en fases evidencia que la fase que representa mayor posibilidad de empalme entre el conjunto de prácticas sugeridas por DevOps en SCRUM es la de desarrollo y calidad. El conjunto de prácticas sugeridas apunta a la implementación de controles para el aseguramiento de la calidad entregando información clave para el aprendizaje y mejora del equipo de desarrollo.

**Palabras clave:** Aseguramiento de la Calidad de Software; DevOps; Ingeniería de software; SCRUM; SQA.

## Abordagem de boas práticas para desenvolvimento de software baseado em DevOps e SCRUM utilizado em microempresas

### Resumo

As empresas de desenvolvimento de software muito pequenas têm no máximo 25 funcionários e possuem fluxo de caixa e tempo limitados para implementar melhorias em seus processos que lhes permitam ser mais competitivas. Essa é uma das razões pelas quais essas empresas recorrem à implementação de frameworks ágeis como o SCRUM para gerenciar o processo de desenvolvimento de software. Mas quando iniciam sua adoção, descobrem que os documentos apenas sugerem as mudanças que podem ser feitas, mas não como fazê-las, tornando o processo de descoberta de quais técnicas, eventos e artefatos são os únicos a serem implementados em uma tentativa e erro dispendiosa abordagem e, em alguns casos, inviável. O mesmo acontece com outros frameworks que podem ser complementares ao SCRUM, como o DevOps, que propõe uma aproximação entre a área de desenvolvimento e operações, onde o maior número de tarefas é automatizado e os controles de qualidade são aumentados para obter melhores produtos. Este artigo expõe três boas práticas baseadas em DevOps, seus modelos de uso e quando dentro do SCRUM podem ser utilizados para facilitar sua adoção nessas empresas. O objetivo é expor um modelo para uso de versionamento, integração e deployment contínuos e os momentos recomendados para sua implementação dentro do SCRUM. Foram identificadas as boas práticas mais relatadas na literatura para desenvolvimento de software baseado em SCRUM e DevOps. Três das melhores práticas foram selecionadas e um modelo de uso foi construído para cada uma delas. Estas práticas foram postas à prova através de um estudo de caso e os resultados obtidos foram avaliados. As práticas foram avaliadas em 3 empresas, obtendo mudanças nos casos de suporte relatados semanalmente e no número de implantações bem-sucedidas. A divisão do processo de desenvolvimento em fases mostra que a fase que representa a maior possibilidade de junção entre o conjunto de práticas sugeridas pelo DevOps no SCRUM é a de desenvolvimento e qualidade. O conjunto de práticas sugeridas aponta para a implantação de controles para garantia da qualidade, fornecendo informações fundamentais para aprendizado e aprimoramento da equipe de desenvolvimento.

**Palavras-chave:** DevOps; Engenharia de software; Garantia de Qualidade de Software; SCRUM; SQA.

## I. INTRODUCTION

The software development industry requires early deployments on production, with high quality and little reprocessing when it comes to maintenance and support to guarantee the profitability of the projects, as indicated in [1]. This leads to the accelerated pace of this type of company requiring effective quality controls, with early feedback on the evolution of the product and allowing project participants to learn what they are doing well and what they can improve as expressed in [2].

A key factor for the implementation of best practices that make it possible to implement the quality controls required by companies is their size. The most common classification is given as: very small entities, made up of a maximum of 25 employees, small entities made up of more than 25 employees and less than 50, medium-sized entities that have between 50 and 250 employees, and large entities that have more than 250 employees [3].

According to [4], very small entities (VSE) make up a large part of the industry and are the ones who suffer the most because their development processes are often empirical, lacking of practices such as code versioning, continuous integration (CI) and continuous deployment (CD) which enable better quality results, guaranteeing optimization of profitability and being more competitive compared to larger companies. According to [3], some of the problems that most affect these companies are the overload of functions of various roles for the same person, the limited cash flow for reinvestment in improving internal processes, the limited number of types of projects they can access, the few quality controls and little or insufficient documentation.

Therefore, this article presents a review of some practices recommended by DevOps that are common in VSEs and when the information they deliver can be used within SCRUM events to guarantee continuous improvement. The article is organized as follows: section 2 describes the motivation scenario, section 3 describes the methodology, section 4 the results, and section 5 the conclusions and future work.

VSE are companies capable of taking on few projects at the same time, because several of their employees are attending multiple functions of different roles (work overload), which can be considered as a factor for high turnover of personnel that these companies usually have [4]. Most of the projects they conduct are developed with non-systematized practices based on the (empirical) experience of the development group rather than on a formal software engineering process. By using empirical processes, VSE find it difficult to implement best practices that lead to the continuous improvement of the company's processes, especially those related to managing the evolution of development and its quality in accordance with [5].

Due to the above, these kind of entities according to [5], require organizing and centralizing the management of the source code, allowing traceability of the history of changes and who has made them. In addition, they need to identify if the changes made to the project, when integrated, produce errors when creating the release that will be put on the test or production servers. Although there are many more practices such as static code analysis, unit tests, functional test automation, according to [5] the most frequent for adoption in early stages in the VSE are versioning, continuous integration and continuous deployment, as mentioned earlier.

## II. Methods

The process followed in this research is made up of the following phases: I) identification of best basic DevOps practices for software development in VSE II) identification of the relationship between SCRUM and DevOps, III) proposal for a versioning model, IV) proposal for a continuous integration (CI) model and V) proposal for a continuous deployment (CD) model. Phases detailed below.

### *A. Identification of Basic Best Practices for Software Development in the VSE*

An investigation was conducted on DevOps and its best practices oriented to preventive quality as an axis of continuous feedback for companies. The objective of this point was to find out the state of these issues and provide a starting point for the identification of the base practices that the VSE should implement.

Initially, a definition of DevOps and its state of the art was search for, highlighting the work of [6] who defines the term as a collaboration between the software development area and the operations area that supports all systems and company services at the hardware level. Its objective is to automate the largest number of tasks related to the management of applications built or under construction through a set of best practices and rules of interaction. From this work, a notorious interest of the academic community in delving into the problems that prevent its implementation can be identified. As important, in [7] the problems for the adoption of the practices suggested by DevOps and their integration with agile frameworks are exposed, leaving a clear opportunity for the research community at this point.

In [8] a revision was found related to adopting DevOps to achieve a continuous delivery process, as in [9], where this is conducted to the implementation of scripts called pipelines that allow integrating the versioning practices, CI, and CD automatically, facilitating its adoption by companies. Additionally, in [10] it is confirmed that the most common practices in companies and that aim to be the first steps in the adoption of DevOps are the three mentioned above.

Finally, the works presented in [5] and [8] point out that an integration is possible between the practices of versioning, continuous integration, continuous deployment and agile frameworks such as SCRUM for the management of changes and the evolution of the source code, the initial verification of the quality of the deployable unit and the generation of information that supports decision-making in the management of software development projects and support for already built systems.

## *B. Identification of the Relationship Between SCRUM and DevOps*

According to [9], the software development cycle is composed of *analysis and planning*, *design*, *development and quality*, and *deployment*. These phases overlap with the agile SCRUM framework and its recommended practices described in [10]. Next, in the Table 1, the SCRUM practices are listed for each phase of the development cycle.

**Table 1.** SCRUM best practices by phase of the development cycle.

| **Development cycle phase** | **SCRUM recommended practice** |
|---|---|
| Analysis and planning | Sprint Planning Meeting |
| Design | |
| Development and quality | Daily SCRUM Meeting / Sprint 0 |
| Deployment | Sprint review / Sprint Retrospective |

On the other hand, according to [15] DevOps proposes to reduce rework and improve organizational culture through a quality environment that implements a set of automatically synchronized practices that are always available as a feedback mechanism, which represents a complement for SCRUM. Practices such as versioning, CI and CD are the most used by companies in the early stages of DevOps adoption as indicated by [8]. Next, a relationship between SCRUM and DevOps within the software development lifecycle is presented in Table 2.

**Table 2.** Relationship between SCRUM and DevOps within the development cycle.

| **Development cycle phase** | **SCRUM Recommended Practice** | **DevOps Recommended Practice** |
|---|---|---|
| Analysis and planning | Sprint Planning Meeting | • None. |
| Design | | • Archetype design. |
| Development and quality | Sprint 0 | • Implementation of the archetype for the development baseline.<br>• Configuration and implementation of versioning model.<br>• Archetypal development baseline versioning.<br>• CI model implementation.<br>• CD model implementation. |
| | Daily SCRUM meeting | |
| Deployment | Sprint review | • CD to ensure sprint review.<br>• Review of information generated by versioning practices, CI, and CD for retrospective analysis of the sprint. |
| | Sprint retrospective | |

Within *the analysis and planning phase*, the functional needs of the project are compiled through user stories, where the functionalities of the applications are described in the acceptance criteria section. When all user stories are built, they are grouped in an artifact called product backlog. Completed the above, it is required to estimate and prioritize it as indicated in [10], to finally determine how long the construction of the solution takes by calculating the number of sprints (measurement of time that goes from 1 to 4 weeks). With this information clearly defined, it is possible to carry out the decomposition of the activities that resolve "what will be done" within the project sprint by sprint and create the SCRUM board as mentioned in [10].

Once the above is done, it is possible to start *the design phase* by building the artifact called Software Architecture Document (SAD), where the non-functional needs of the system are reflected by describing quality attributes of the architecture. These requirements are resolved through design patterns, which are existing solutions to recurring problems, are proven effective and are found in the development frameworks of the various programming languages as indicated [11]. These decisions mark the architectural style of the solution and are reflected in UML diagrams that respond to the different views of the SAD. Once the above is done, it is possible to create a project archetype that responds to the characteristics detailed in the SAD, allowing to establish and version the development baseline so that the whole team can have it at the time of starting development.

During *the development and quality phase*, it is required that at the beginning of each project, according [10], a sprint 0 or preparation for development is carried out. This allows verifying that the entire team knows what will be done and how it will be done, carry out concept tests, build the archetype, create the repository for versioning and implement the model to be used in it, configure the tools that support CI and CD practices, version the development baseline and verify that all team members have access to the tools and configurations required for the formal start of the project. Once sprint 0 has been completed, development begins.

At the beginning of each development sprint, it should start with the sprint planning meeting as indicated [10], where it is guaranteed that the entire team knows what user story is going to be carried out and what activities are required at the development level to complete them. This way they can break them down in the SCRUM board where the daily follow-up will be conducted. Additionally, the team verifies that the architectural decisions that guide the structuring of the project are known and that they are reflected in the baseline. It is also verified that all the members are linked to the repository, so they can manage their changes there, besides, they have downloaded the development baseline built-in sprint 0, being ready to start with the construction of the user story that each one selects. It is important that the teams take as a best practice within the organizational culture that at the beginning of each working day they must perform a pull (update of changes) on the integration branch in which they are working within the versioner. Similarly, at the end of each working day, the changes made must be uploaded to each developer's own branch to ensure that they are not lost. The details of the versioning model will be described later for a better understanding.

During the evolution of the sprint, according to [10], an event called the daily SCRUM meeting is held. The objective of this meeting is to identify, through the presentation of each one of the members, the advances of the previous day, what they are planning to do during that day and the impediments they must advance. This promotes proper project management through a continuous flow of information that allows knowing the real status of the project and react to delays in no more than 24 hours. Here the teams can make use of versioning as a practice to show the work done the day before. In addition, each time a user story is completed, the versioned changes of all collaborators must be integrated to ensure that the release can be generated and deployed in the test environment that is required for the respective functional review, which it can be manual or automated. This meeting is held every day of the sprint with the aim of identifying whether the goals will be achieved as estimated or if it is not possible and what will be done about it.

Regarding the *deployment phase*, at the end of each integration of changes it is necessary to carry out the deployment, in this way it is possible to have the latest stable version placed in a test environment and ready to present the sprint review as indicated in [10]. During this event, the team presents to the project stakeholders what was done during the sprint according to the commitments. After this, the team performs a last event called sprint retrospective that, according to [10], allows a review of the positive aspects and opportunities for improvement that are the product of the learning that the sprint execution has left. Here it is key to review the versioning history to identify if its use has been correct, in addition to knowing how many stories were built without delay according to the planning, how many required improvements and how many detected corrections after delivery. Likewise, it is reviewed how many times the continuous integration was not successful and for what reason, besides, it is observed if there were some cases where the continuous deployment was not successful. All this information together with the use of retrospective techniques allows the team to have the opportunity to continuously learn and improve, sprint by sprint.

### *C. Proposal for a Versioning Model*

Based on the work of [12], it is possible to identify that the starting point of a process oriented to best practices for software development begins with versioning. Versioners are tools that function as repositories that centralize changes and guarantee availability always for all the members of the development teams (collective ownership of the code). Additionally, all the changes stored by the tool are saved in a history for traceability and if an error is generated in the integration of changes, return to the previous stable version without major delays (failure recovery).

To make use of this practice, it is recommended to first know the proper structure of a versioning model, followed by the steps to use and the practices that must be implemented in its use. Versioners are made up of a local registry on the user's machine and a repository registry that stores and integrates all changes within the tool. The purpose of this tool is to synchronize local changes with the version hosted in the tool. All changes are housed in a structural separation that the tool creates called a branch, and usually the initial branch that every versioner creates is called master.

If the development team only work on the master branch, quiet conflicts would be generated when integrating the changes and they would not have a latest stable version because the source code would be constantly being manipulated. For this reason, it is recommended to detach another branch derived from the master, which fulfills the functions of integrating the changes of all the participants. This allows changes to be centralized in the integrations branch and once these are considered stable, the latest version is synchronized with master, guaranteeing that what it is holding in master will always be the most recent and stable of the project. To guarantee greater control of changes, it is recommended to detach from the integration branch, one branch for each project collaborator. The objective of this is for each developer to work on their specific activities until they have guaranteed to have finished, uploading only to the integrations branch when this is done. Once the developer's changes are integrated, the entire team should be notified seeking everyone's synchronization.

To integrate the changes from the developer branch to the integrations branch, it is recommended to do it through a *pull request* and not directly. This generates a request that is submitted for review (manual code inspection) so that a member of the team determines if the changes do not negatively affect the project and comply with the development policies implemented in the team (code standard). If the changes are authorized, the new is included (merged) in the desired branch through the versioner and the tool automatically notifies members by email of its success. Otherwise, it indicates that it has been rejected and the reason, with the intention of the developer who requested to upload the change can take the necessary corrective actions and try again. Fig. 1 summarizes the step by step of what has been mentioned. Additionally, it is recommended that all members always download the changes from the integrations branch at the start of the working day to work on the latest development version and at the end of the day they should always upload the changes they have made to their branch.

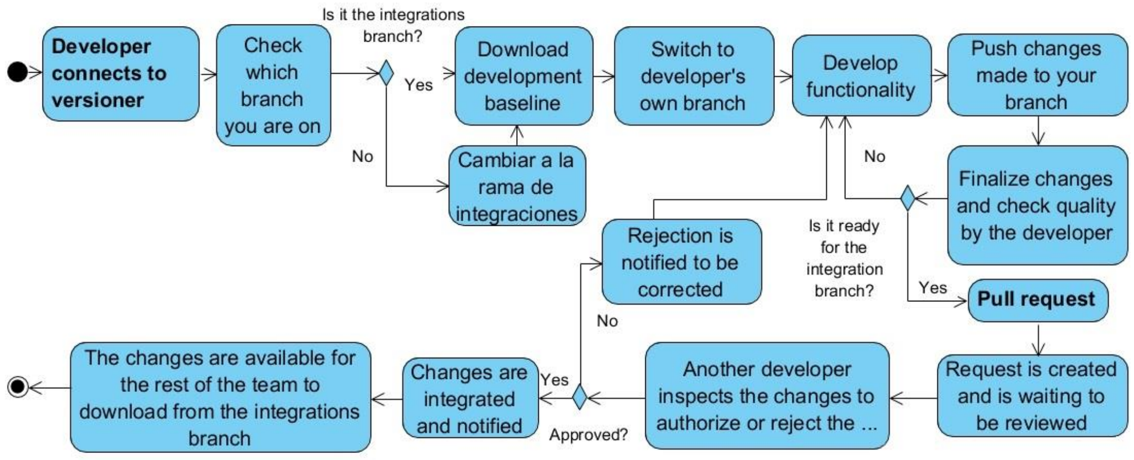

Developer connects to versioner
Check which branch you are on
Is it the integrations branch?
Yes
No
Download development baseline
Cambiar a la rama de integraciones
Switch to developer's own branch
Develop functionality
Push changes made to your branch
Finalize changes and check quality by the developer
Is it ready for the integration branch?
No
Yes
Pull request
Request is created and is waiting to be reviewed
Another developer inspects the changes to authorize or reject the ...
Approved?
No
Yes
Rejection is notified to be corrected
Changes are integrated and notified
The changes are available for the rest of the team to download from the integrations branch

**Fig. 1.** Versioning model.

### *D. Proposal for a Continuous Integration (CI) Model*

The practice of continuous integration (CI) goes hand in hand with versioning according to [13]. Although versioning helps to centralize changes, maintain order, and trace the evolution of the system being developed, it does not allow verifying the impact of the changes generated on the deployable unit. Due to the above and in accordance with [14], it is advisable to adopt the practice of continuous integration that can validate this as part of the organizational culture.

To implement the IC, it is necessary for the tool that implements this practice to be able to generate the deployable unit through console instructions, depending on the programming language and the operating system. If the process has been successful, the team must be notified that the CI has finished correctly and in case of failure, it will notify who was the last to make changes, so that they can make the pertinent adjustments. The above is summarized in Fig. 2.

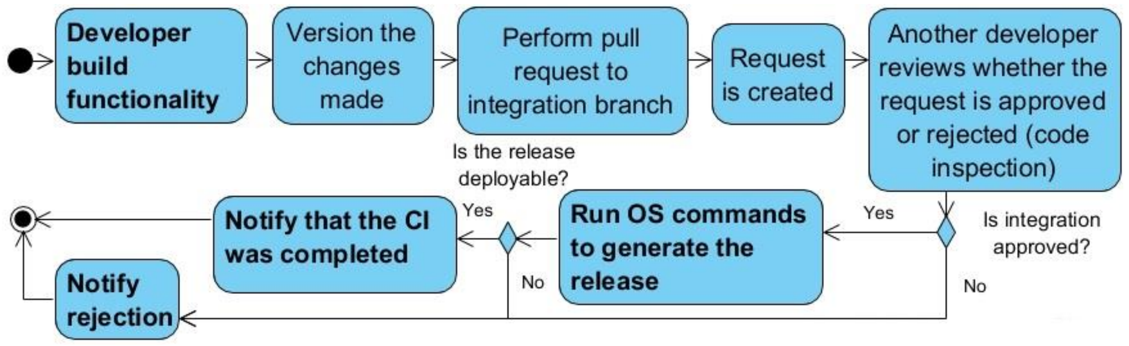

Developer build functionality
Version the changes made
Perform pull request to integration branch
Request is created
Another developer reviews whether the request is approved or rejected (code inspection)
Is integration approved?
Yes
No
Run OS commands to generate the release
Is the release deployable?
Yes
No
Notify that the CI was completed
Notify rejection

**Fig. 2.** CI Model.

### *E. Proposal for a Continuous Deployment (CD) Model*

When the source code is versioned and the CI generates the deployable unit file, which must be put on an application server for the software to be operational, the CD is possible according to [16]. The CD practice takes advantage of the fact that the CI has generated the deployable unit, so as not to repeat this process, and transfers that file to the server (physical or in the cloud) where the application server that will deploy it is hosted. Once the file is taken to the server, it is placed in the required location, depending on the application server and the operating system-specific commands required to perform the deployment are executed, which in most cases make the server of applications, but not the physical server, require restarting their service. The above is summarized in Fig. 3.

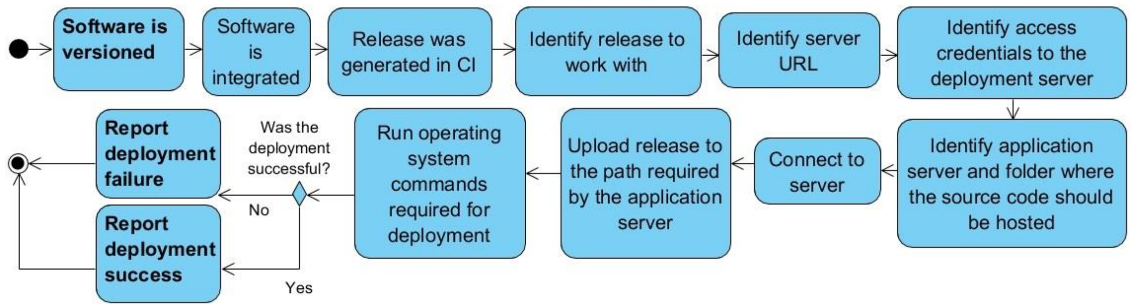

Software is versioned
Software is integrated
Release was generated in CI
Identify release to work with
Identify server URL
Identify access credentials to the deployment server
Identify application server and folder where the source code should be hosted
Connect to server
Upload release to the path required by the application server
Run operating system commands required for deployment
Was the deployment successful?
No
Yes
Report deployment failure
Report deployment success

**Fig. 3.** CD Model.

## III. Results

To review the impact caused by the implementation of the suggested practices, three companies were selected, which for confidentiality reasons will be called EMP1, EMP2 and EMP3.

The EMP1 company has less than 25 employees and develops data analytics solutions. Additionally, it was detected that they use code versioning, where they have a master branch that hosts the latest completely stable version of the development and a branch that integrates the work of all the developers that allows centralizing the constant changes of the team during each sprint.

Having identified the above, the versioning model is exposed for the refinement of the practice, recommending that a branch should be created from the integration branch for each developer that can be named with the first letter of the name followed by the first surname. In the case of homonyms, the name to be assigned in each branch is negotiated with those involved. This allows over time to maintain control over the evolution of the projects, identifying changes made by each developer and recovering from syntax or logical errors that could affect the dropdown and that are detected by the IC when a pull request is made to the branch of integrations. Likewise, the EMP1 has implemented the IC practice to prevent syntactic errors from generating the deployable unit correctly.

EMP1 was aware of, but had not implemented, the practice of continuous deployment. Once the CD model has been exposed, the process of adopting this practice begins. Since the IC and the CD in some cases can go in the same tool (as it happens with the EMP1), the file that allows the IC is modified, to add the step of the CD configuring the path of the test server, the permissions of access and operating system commands for deployment. With the above, it was achieved that every time a stable integration is carried out, it is automatically deployed on the test server and the quality team is notified for review.

The EMP2 was characterized as a VSE with problems typical of this kind of companies. Its biggest limitation is that the development and technology area is made up of 9 people. Before the process of adopting best practices, all their projects were built in PHP and Bootstrap using a code generator called PHPRUNNER. They did not use versioning and were unaware of the model of working with versions. Therefore, the model was exposed and implemented. One of the direct impacts that was achieved whit this it was the organization of work, the non-loss of information, the elimination of overload of personnel who developed the same, the traceability of changes and the availability of codes always for those who need them. Additionally, they identified this step as an opportunity for the implementation of the other two practices and even showed total interested that once they have adopted the previous practice as part of their organizational culture, they can explore the implementation of more practices aimed at preventive quality that optimize your development process based on SCRUM.

The implemented versioning tool allows continuous integration, so it was natural in their process to build the required configuration file and adopt the practice. The impact was significant for EMP2 for several reasons. The first is that they have detected that on some occasions two people work on the same functionality at the same time for different support cases. When doing this before the implementation of the practices, the changes were overwritten, but with the implementation of the versioning it is prevented from happening because whoever tries to upload the latest change is forced to download the previous settings, unify everything in a local and then upload them to the versioner. The second reason is that the CI tool allows to automatically detect if the changes are preventing the deployable unit from being generated. In this case, the notification and attention to the situation is resolved by the last person who uploaded the change. Additionally, EMP2, through code reviews that allow the integration to be approved, avoided reprocessing, and thus decongested the support channel they provide to their products. Also, the static analysis of the code helped to detect that they do not use any dependency manager for their programming language, they generate a lot of unnecessary code, and the current architecture did not fully meet the real needs of the company that has been sacrificed for development speed, making support more demanding for the entire team.

The EMP3 company reflects an empirical and non-systematized software development process. Therefore, they do not work with the best practices suggested in this article. Its main function is to develop custom software in JAVA using Spring, JPA and Swagger for the construction of a Rest API that exposes the functionalities of the system and that are consumed by the presentation layer, built in Dart with Flutter for web and mobile. The three (3) practices were exposed by creating repositories for both the backend built in java and the frontend. Each repository has the structure described in the proposed versioning model. This allowed greater traceability on the evolution of its main application. Additionally, it allowed them to organize support cases according to who developed it. The practice of CI and CD by the backend was easy for the team to implement and adopt, giving them the ability to detect changes that prevent the deployable unit from being built and quickly put it on the test server. On the part of the continuous deployment for the web, the script was created without difficulty, and it was deployed in an agile way, but for the mobile case the tool does not cover the possibility of integrating with the Apple Store or Play Store deployment platform, so recommended expanding the investigation to another tool that does allow it. Table 3 summarizes what happened with the three companies.

**Table 1.** Impact of implemented practices

| Entity | Detected Practices | Implemented Practices | Impact |
|---|---|---|---|
| EMP1 | Versioning | Improved Versioning | The versioning model is improved for greater control. |
| | CI | CD | Manual deployment is eliminated and automated. |
| EMP2 | None | Versioning | |

| Entity | Detected Practices | Implemented Practices | Impact |
|---|---|---|---|
| | | CI | Quality controls are implemented that decongest the company's supports. |
| | | CD | Faster deployment speed with less rework. |
| EMP3 | None | Versioning | Improved company support. |
| | | CI | |
| | | CD | Elimination of manual deployments for web applications. |

Once the implementation of the suggested best practices was conducted, the first measurement that was made was the number of failures that appear after development and that are reported by end users when the software is already operating (support cases). This is measured before and after implementation to compare the impact achieved.

The EMP1 before practices reported 9 to 12 weekly requests, while EMP2 had an average of 16 to 22. On the other hand, EMP3 presented 5 to 8 weekly reports. After the implementation of best practices, EMP1 reports a decrease of 41.7% of cases, reaching a report of 3 to 5 weekly requests, while EMP2 indicates a decrease of 45.5%, achieving a range of 8 to 10 reports. per week The EMP3 shows a decrease of 37.5% of the reports reflecting only 2 to 3 weekly requests. The above results are summarized in Fig. 4.

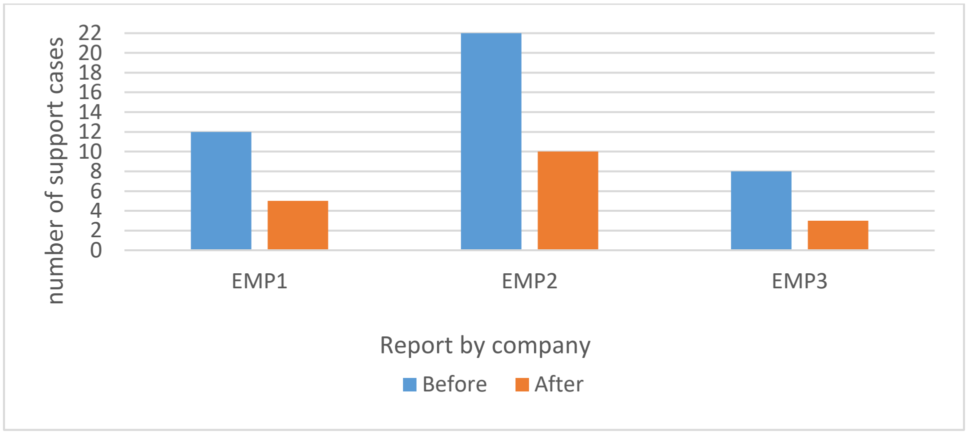


**Fig. 4.** Support cases before and after the implementation of good practices.

The practice of versioning, CI and CD were significant for each company, integrating the different changes in a controlled way and being always prepared to face a production release. EMP2 and EMP3 are the ones that suffered the most from manual deployments, because in some situations and due to response time pressure on support, they directly manipulated the code deployed on the server to solve a problem. Not only losing control of the change, but also occasionally generating inoperability at times due to poor handling. The measurement of effective deployments before implementing the practices shows that EMP1 had a total of 90% success, due to a deployment checklist that is executed manually. After implementation, they get 100% and the manual review of the checklist disappears, leaving the developer in charge free to fulfill other functions. The EMP2 had a 50% correct deployment before the practices, because its process is empirical and without any control. After the implementation, 100% correct deployments are obtained. The EMP3 had a total of 80% correct deployments before the practices because it also had a checklist. After implementation they reach 100%. Fig. 5 summarizes the results presented above.

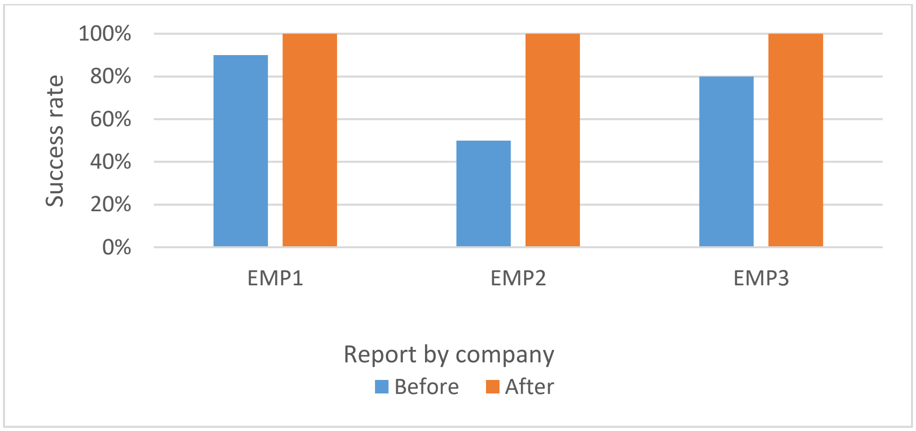


**Fig. 5.** Successful deployments before and after the implementation of good practices.

## IV. Conclusions and Future Work

The division by phases of the development process reflects that development and quality presents a high possibility of including best practices based on DevOps for SCRUM, managing to approach an initial selection of best practices, detail them and propose an implementation way, which is tested in a case study.

From the case study it was determined that the practice of versioning allows a historical follow-up evidencing progress or delays in the project, when a change was made and who makes it, guaranteeing that the code is always available for those who need it. Additionally, unifying the work of all the collaborators through an integration branch, with the pull request command, implies doing a manual inspection of the code whenever it is requested to merge the changes, which adds a quality filter at that point in the process. Together with the versioning, the IC implements another quality filter, verifying if a modification has been uploaded that by inadvertence prevents the deployable unit from being generated or not. Together, these practices generate an environment of preventive quality that implements significant controls for the development process. Also, the implementation of the CD allowed the development teams to guarantee a 100% effective deployment from the early stages of a project, as evidenced by the results.

On the other hand, there is evidence of a direct impact on the quality of what is developed, because the number of support cases that companies must address decreases when quality controls are increased during development.

The research group hopes to explore as future work other approaches to the actual versioning model, such as replacing the name of each developer branch for an attribute or characteristic (feature), as well as delving into the implementation of unit tests and their automatic inspection through CI, analysis static code, automated functional tests, and the integration of all practices to the current model.

## AUTHORS' CONTRIBUTION

**Manuel-Alejandro Pastrana-Pardo:** Conceptualization, Methodology Resources, Writing- Original draft.

**Hugo-Armando Ordoñez-Erazo:** Conceptualization, Methodology, Resources, Writing- Original draft, Supervision, Project administration, Funding acquisition.

**Carlos-Alberto Cobos-Lozado:** Supervision, Writing- Review and Editing.

## ACKNOWLEDGMENTS

The authors express their gratitude to the Institución Universitaria Antonio José Camacho for its participation in the research project entitled *"Implementation of standardized best practices for the development of software based on SCRUM and DevOps in very small companies"*. In the same way, to the Research Vice-Rectory of the Universidad del Cauca for the support provided to the group of researchers in the execution of this project.

## REFERENCES

[1] S. Martinez-Fernandez, A. Vollmer, A.Jedlitschka, X. Franch, L. Lopez, P. Ram, P. Rodriguez, S. Aaramaa, A. Bagnato, M. Choras, J. Partanen, “Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study,” *IEEE Access*, vol. 7, pp. 68219–68239, 2019. https://doi.org/10.1109/ACCESS.2019.2917403

[2] P. Rodríguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, M. Oivo, “DevOps in practice: A multiple case study of five companies,” *Information and Software Technology*, vol. 114, pp. 217–230, 2019. https://doi.org/10.1016/j.infsof.2019.06.010

[3] ISO/IEC JTC 1/SC 7/WG 24, *ISO/IEC DTR 29110-5-6-3:2018*, 2019. https://isotc.iso.org/livelink/livelink/open/jtc1sc7wg24

[4] M. Munoz, J. Mejia, A. Lagunas, “Implementation of the ISO/IEC 29110 standard in agile environments: A systematic literature review,” in *Iberian Conference on Information Systems and Technologies*, 2018, pp. 1–6. https://doi.org/10.23919/CISTI.2018.8399332

[5] A. Hemon, B. Lyonnet, F. Rowe, B. Fitzgerald, “From Agile to DevOps: Smart Skills and Collaborations,” *Information Systems Frontiers*, vol. 22, no. 4, pp. 927–945, 2020. https://doi.org/10.1007/s10796-019-09905-1

[6] S. Badshah, A. A. Khan, B. Khan, “Towards Process Improvement in DevOps,” in *Proceedings of the Evaluation and Assessment in Software Engineering*, Apr. 2020, pp. 427–433. https://doi.org/10.1145/3383219.3383280

[7] M. Z. Toh, S. Sahibuddin, M. N. Mahrin, “Adoption Issues in DevOps from the Perspective of Continuous Delivery Pipeline,” in *Proceedings of the 2019 8th International Conference on Software and Computer Applications*, 2019, pp. 173–177. https://doi.org/10.1145/3316615.3316619

[8] A. Hemon, B. Lyonnet, F. Rowe, B. Fitzgerald, “Conceptualizing the transition from agile to DevOps: A maturity model for a smarter is function,” in *IFIP Advances in Information and Communication Technology*, 2019, pp. 209–223. https://doi.org/10.1007/978-3-030-04315-5_15

[9] R. S. Pressman, B. R. Maxim, *Software Engineering: A Practitioner’s Approach*, Eighth Edition, McGraw-Hil. Boston, USA: McGraw-Hill, 2015.

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

[11] M. O. Onarcan, Y. Fu, M. O. Onarcan, Y. Fu, “A Case Study on Design Patterns and Software Defects in Open Source Software,” *Journal of Software Engineering and Applications*, vol. 11, no. 5, pp. 249–273, May 2018. https://doi.org/10.4236/JSEA.2018.115016

[12] M. Pastrana, H. Ordoñez, A. Rojas, A. Ordoñez, “Ensuring Compliance with Sprint Requirements in SCRUM: Preventive Quality Assurance in SCRUM,” in *Advances in Intelligent Systems and Computing*, 2019, pp. 33–45. https://doi.org/10.1007/978-981-13-6861-5_3

[13] N. Railic, M. Savic, *Architecting Continuous Integration and Continuous Deployment for Microservice Architecture*, 2021. https://doi.org/10.1109/INFOTEH51037.2021.9400696

[14] 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*, 2017, pp. 60–71. https://doi.org/10.1109/ASE.2017.8115619