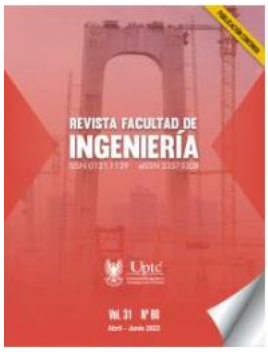
Revista Facultad de Ingeniería
Journal Homepage: https://revistas.uptc.edu.co/index.php/ingenieria



Approach to the Best Practices in 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 in 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 entities in software development have a maximum of 25 employees. Their cash flow and time available for implementing improvements in their

[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

processes to enable them to be more competitive are limited, leading them to turn to agile frameworks such as SCRUM to manage the software development process. However, when they try to adopt these, they find that the documents only suggest changes that can be made and not how to make them. As a result, the trial and error process of discovering which techniques, events and artifacts ought to be implemented is costly and, in some cases, unfeasible. The same applies to other frameworks that can complement SCRUM, such as DevOps, a framework that proposes a rapprochement between the development and operations areas, in which as many tasks as possible are automated, and quality controls are increased to obtain better quality products. This article presents three best practices based on DevOps, its models of use and when these can be used within SCRUM to facilitate its adoption in the smallest companies. A model is presented for the use of versioning, integration, and continuous deployment and the particular moments recommended for implementing these within SCRUM. The best practices most widely reported in the literature for software development based on SCRUM and DevOps were identified. Three were then selected, and a usage model was built for each of them. Then, they were evaluated using a case study, and the results were assessed. 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 reveals that the development and quality phase provides more possibilities for splicing among the set of practices suggested by DevOps in SCRUM. Likewise, the set of suggested practices points to the implementation of controls for quality assurance, providing key information for development team learning and improvement.

Keywords: DevOps; SCRUM; Software Engineering; Software Quality Assurance; 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

Software development companies require early deployments of tools in production, with high quality and minimum reprocessing when it comes to maintenance and support to ensure project profitability, as indicated in [1]. The accelerated pace of this type of company consequently requires effective quality controls, with early feedback on the evolution of the product, allowing project participants to learn what they are doing well and what they might improve on, as expressed in [2].

A key factor regarding 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, comprising a maximum of 25 employees, small entities 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 (VSEs) make up a large part of the industry and suffer the most since their development processes are often empirical, lacking practices such as code versioning, continuous integration (CI) and continuous deployment (CD), all of which enable better quality results, ensuring optimization of profitability while becoming 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.

This article reviews several practices recommended by DevOps that are common in VSEs and for which the information they deliver can be used within SCRUM events to ensure continuous improvement. The article is organized as follows: Section 2 describes the motivation scenario, Section 3 outlines the methods, Section 4 presents the results, and Section 5 lays out the conclusions and future work.

II. Motivation Scenario

VSEs are able to take on a few projects simultaneously since a number of their employees carry out multiple functions relating to different roles (work overload). This may be considered a factor in the high turnover of personnel with which these companies usually have to contend [4]. Most of their projects are carried out via non-systematized practices based on the (empirical) experience of the development group rather than on a formal software engineering process. Depending on empirical processes, VSEs 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, according to [5].

As a result, these kinds of entities, according to [5], would benefit from 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 whether or not 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, and functional test automation, according to [5], the most frequently adopted in the early stages in VSEs are versioning, continuous integration, and continuous deployment.

III. Methods

The process followed in this research comprises the following phases: I) identification of DevOps basic best practices in software development in VSEs; 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. These phases are detailed below.

A. Identification of Basic Best Practices in Software Development in VSEs

An investigation was carried out on DevOps and its best practices oriented to preventive quality as an axis of continuous feedback for companies. The objective of this stage was to determine the state of these issues and provide a starting point for identifying the basic practices that VSEs should implement.

Initially, a definition of DevOps and its state of the art was sought, highlighting the work of [6], who define 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. It aims 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 strong interest can be identified in the academic community regarding the problems that prevent its implementation. Just as important, in [7], the problems for adopting the practices suggested by DevOps and their integration with agile frameworks are laid out, demonstrating a clear opportunity for the research community on this point.

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

Finally, the works presented in [5] and [8] indicate that 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 the 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 comprises *analysis and planning*, *design*, *development and quality*, and *deployment*. These phases overlap with the agile SCRUM framework and its recommended practices described in [10]. Table 1 below lists the SCRUM practices for each phase of the development cycle.

Table 1. SCRUM best practices by development cycle phase.

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

Moreover, 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, thus representing a complement for SCRUM. Versioning, CI and CD are the practices most widely used by companies in the early stages of DevOps adoption, as indicated by [8]. A relationship between SCRUM and DevOps within the software development lifecycle is presented in Table 2 below.

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 archetype for the development baseline. • Configuration and implementation of the versioning model. • Archetypal development baseline versioning. • CI model implementation. • CD model implementation.
	Daily SCRUM meeting	
Deployment	Sprint review	• CD to ensure sprint review. • 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. Once the above is complete, it is required to estimate and prioritize it as indicated in [10] to finally determine how long it takes to build the solution 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 decompose 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 the quality attributes of the architecture. These requirements are resolved through design patterns -existing solutions to recurring problems, proven effective and 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. It is then 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 on starting development.

During *the development and quality phase*, it is required that at the beginning of each project, according to [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, carrying out concept tests, building the archetype, creating the repository for versioning and implementing the model to be used in it, configuring the tools that support CI and CD practices, versioning the development baseline and verifying 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.

Each development sprint should start with the sprint planning meeting as indicated [10], where it is ensured that the entire team knows what user story will be carried out and what activities are required at the development level to complete them. This way, they can break them down on 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 that they can manage their changes there. They have furthermore downloaded the development baseline built-in sprint 0 and are ready to start the construction of the user story that each one selects. The teams must take as a best practice within the organizational culture performing a pull (update of changes) on the integration branch in which they are working within the versioner at the beginning of each working day. 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. This meeting aims 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 overcome. This promotes proper project management through a continuous flow of information that allows knowing the real status of the project and reacting to delays in no more than 24 hours. Here the teams can use 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 may be manual or automated. This meeting is held every day of the sprint to identify whether or not the goals will be achieved as estimated 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 final 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 gained from the sprint execution. Here it is key to review the versioning history to identify if its use was 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 often the continuous integration was unsuccessful and for what reason. Moreover, it is observed whether or not there were some cases in which continuous deployment was unsuccessful. All this information, together with the use of retrospective techniques, allows the team to learn and improve sprint by sprint continuously.

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 in software development is versioning. Versioners are tools that function as repositories that centralize changes and ensure availability for all members of the development team (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 integrating the changes, it returns to the previous stable version without major delays (failure recovery).

To implement this practice, it is recommended first to understand the correct structure of a versioning model, followed by the steps for its use and the practices that must be implemented therein. Versioners comprise 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 given the term *master*.

If the development team were to work only on the master branch, quiet conflicts would be generated in integrating the changes, and there would not be 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 to fulfill the function 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 the master, ensuring that what it is holding in the master will always be the most recent and stable version of the project. For greater control of changes, it is recommended to detach one branch for each project collaborator from the integration branch. The objective of this is for each developer to work on their specific activities until they are quite sure they have finished, only uploading to the integrations branch when this is done. Once the developer changes are integrated, the entire team should be notified, seeking synchronization of everyone.

It is recommended to integrate changes from the developer branch into the integrations branch through a *pull request* and not directly. This generates a request submitted for review (manual code inspection) so that a team member 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 of its success through email. Otherwise, it indicates that it has been rejected, and the reason is supplied with the aim that 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.

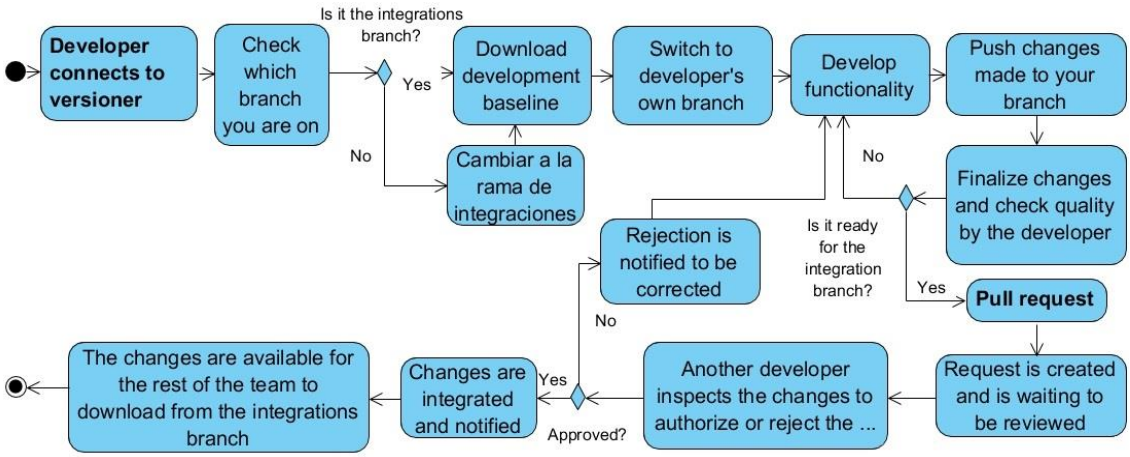


Fig. 1. Versioning model.

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 following [14], it is advisable to adopt the practice of continuous integration that can validate this as part of the organizational culture.

To implement the CI, the implementation tool must be able to generate the deployable unit through instructions from the console, depending on the programming language and operating system. If the process was 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.

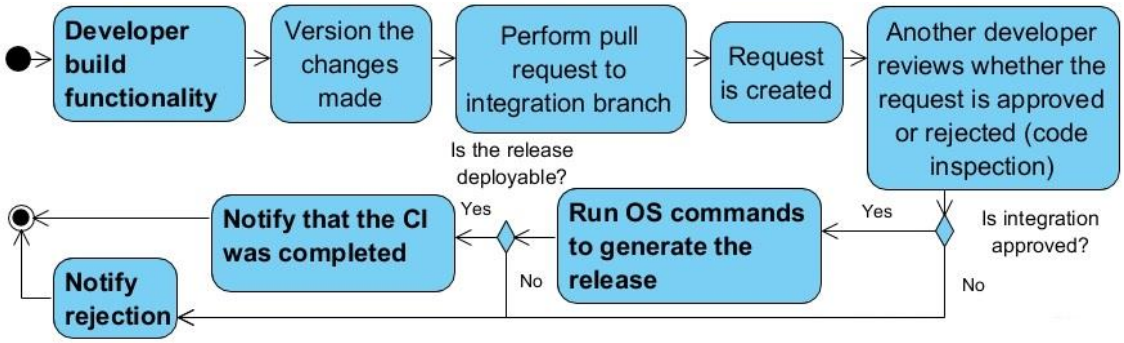


Fig. 2. CI Model.

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 has been taken to the server, it is placed in the required location, depending on the application server. The operating system-specific commands required to perform the deployment are then executed, which in most cases cause the applications server, but not the physical server, to require to restart its service. The above is summarized in Fig. 3.

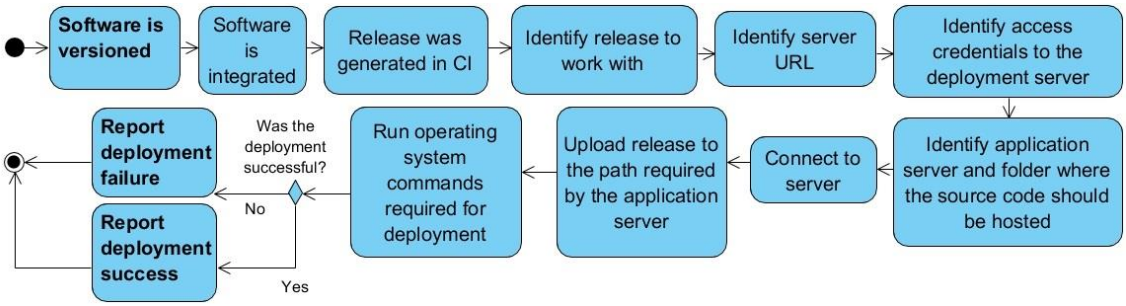


Fig. 3. CD Model.

IV. RESULTS

Three companies were selected to review the impact of implementing the suggested practices. For confidentiality reasons, they will be called COY1, COY2, and COY3

Company COY1 has less than 25 employees and develops data analytics solutions. Moreover, it uses code versioning, where there is 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 exhibited 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 for maintaining control over the evolution of the projects over time, identifying changes made by each developer and recovering from syntax or logical errors that could affect the dropdown and that are detected by the CI when a pull request is made to the integrations branch. COY1 likewise implemented CI to prevent syntactic errors from generating the deployable unit correctly.

COY1 was aware of CD but had not implemented it. Once the CD model has been exhibited, the process of adopting this practice begins. Since the CI and the CD, in some cases, can be implemented in the same tool (as happens with the COY1), the file that allows the CI is modified to add the step of CD configuring the test server path, 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.

COY2 was characterized as a VSE with problems typical of this kind of company. Its biggest limitation is that the development and technology area is made up of 9 people. Before adopting these best practices, all its projects were built in PHP and Bootstrap using a code generator called PHPRUNNER. It did not use versioning

and was unaware of the model of working with versions. Therefore, the model was exhibited and implemented. One of the direct impacts achieved with this 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, it identified this step as an opportunity for the implementation of the other two practices, and even showed total interest that once they have adopted the previous practice as part of its organizational culture, it can explore the implementation of more practices aimed at the preventive quality that optimizes its SCRUM-based development process.

The versioning tool implemented allows continuous integration, so it was natural in its process to build the required configuration file and adopt the practice. The impact was significant for COY2 for several reasons. The first was that it detected that, on some occasions, two people worked 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. However, with the implementation of versioning, it is prevented because whoever tries to upload the latest change is forced to download the previous settings, unify everything locally 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 are resolved by the last person who uploaded the change. Additionally, COY2, through code reviews that allow the integration to be approved, avoided reprocessing, and thus decongested the support channel it provides to its products. The static analysis of the code also helped to detect that it does not use any dependency manager for its programming language, it generates a lot of unnecessary code, and the current architecture did not fully meet the real needs of the company that have been sacrificed for speed of development, making support more demanding for the entire team.

Company COY3 reflects an empirical and non-systematized software development process. Therefore, it did 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 to construct a Rest API that exhibits the system's functionalities and that

are consumed by the presentation layer, built in Dart with Flutter for web and mobile. The three (3) practices were exhibited 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 in the evolution of its main application. Additionally, it allowed the company 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, allowing them to detect changes that prevent the deployable unit from being built and quickly put it on the test server. On the part of continuous deployment for the web, the script was created without difficulty and deployed in an agile way. However, for the mobile case, the tool does not cover the possibility of integrating with the Apple Store or Play Store deployment platform, so it is recommended to expand 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
COY1	Versioning	Improved versioning	The versioning model is improved for greater control.
	CI	CD	Manual deployment is eliminated and automated.
COY2	None	Versioning	Quality controls are implemented that decongest company supports.
		CI	
		CD	Faster deployment speed with less rework.
COY3	None	Versioning	Improved company support.
		CI	
		CD	Elimination of manual deployments for web applications.

Once the implementation of the suggested best practices was carried out, the first measurement 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 was measured before and after implementation to compare the impact achieved.

COY1 reported 9 to 12 weekly requests before practices, COY2 had an average of 16 to 22, and COY3 had 5 to 8 weekly reports. Following the implementation of

best practices, COY1 reported a decrease of 41.7% of cases, reaching a report of 3 to 5 weekly requests, while COY2 indicated a decrease of 45.5%, achieving a range of 8 to 10 reports per week. COY3 showed a decrease of 37.5% of the reports reflected in only 2 to 3 weekly requests. The above results are summarized in Fig. 4.

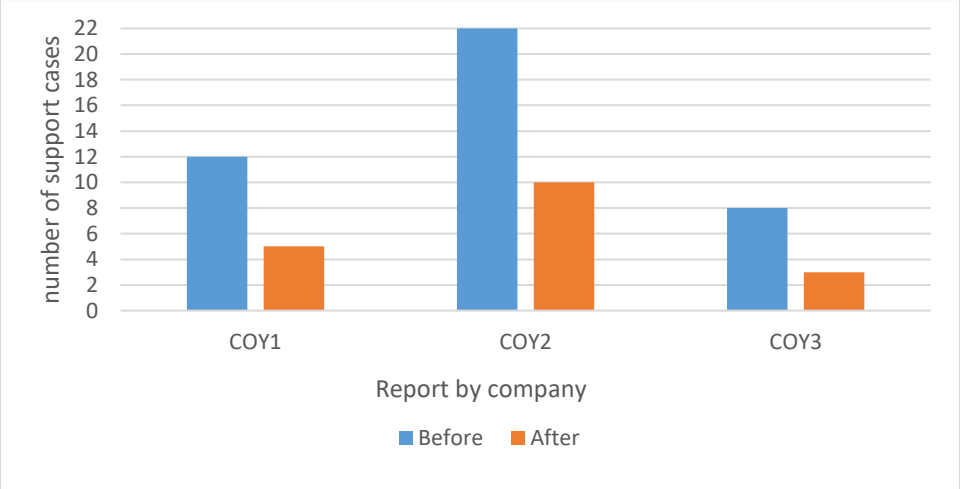


Fig. 4. Support cases before and after the implementation of best practices.

The practices of versioning, CI and CD were significant for each company, integrating the different changes in a controlled way and always being prepared to face a production release. COY2 and COY3 suffered most from manual deployments because, in some situations and due to response time pressure on support, they directly handled 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. Measurement of effective deployments before implementing the practices shows that COY1 had a 90% success rate due to a deployment checklist that is executed manually. After implementation, they obtained 100%, and the manual review of the checklist disappeared, leaving the developer in charge free to fulfill other functions. COY2 had a 50% rate of correct deployment before the practices because its process is empirical and without any control. After implementation, 100% correct deployments were obtained. COY3 had 80% correct deployments before the practices because it also had a checklist.

After implementation, they reached 100%. Fig. 5 summarizes the results presented above.

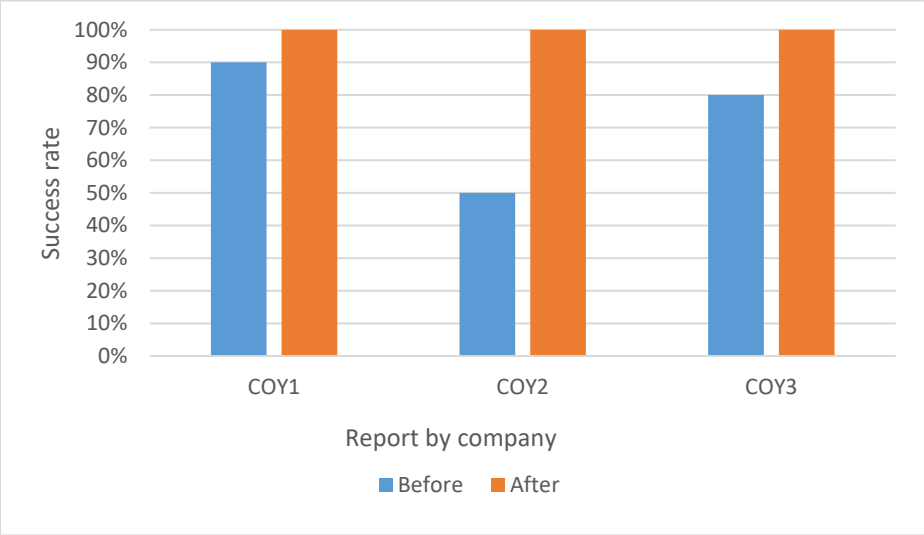


Fig. 5. Successful deployments before and after the implementation of best practices.

V. Conclusions and Future Work

The division by phases of the development process reflects that development and quality present 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 a way of implementation that was then 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, ensuring 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 manually inspecting the code whenever it is requested to merge the changes, which adds a quality filter at that point in the process. Together with versioning, CI 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 a preventive quality that implements significant controls for the development process. Also, the implementation of CD allowed the

development teams to ensure a 100% effective deployment from the early stages of a project, as seen in the results.

Moreover, there is evidence of a direct impact on the quality of what is developed because the number of support cases that companies were required to address decreased when quality controls were increased during development.

The research group hopes to explore as future work other approaches to the current 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, acquisition of funding.

Carlos-Alberto Cobos-Lozado: Supervision, writing-review and editing.

ACKNOWLEDGMENTS

The authors express their gratitude to the Institución Universitaria Antonio José Camacho for participating in the research project entitled *"Implementation of standardized best practices for the development of software based on SCRUM and DevOps in very small companies."* Likewise, to the vice-rector for research office of 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*, ISO, 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*, 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, L. Guía, D. de Scrum, L. Reglas, L*a 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, 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," in *20th International Symposium INFOTEH-JAHORINA*, 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