ISSN 2303-4521
Periodicals of Engineering and Natural Sciences *Original Research*
Vol. 9, No. 4, November 2021, pp.854-863

Documenting and implementing DevOps good practices with test automation and continuous deployment tools through software refinement

Manuel Alejandro Pastrana Pardo[1], Hugo Armando Ordoñez Erazo[2], Carlos Alberto Cobos Lozada[2]
[1] Faculty of Engineering, Antonio Jose Camacho University Institution
[2] Faculty of Electronic Engineering and Telecommunications, Cauca University

ABSTRACT

The accelerated pace of life of companies in Colombia and the world, entails the need to obtain software developments with the highest quality, in the shortest possible time and with minimal reprocessing after it is put into production. Therefore, the use of good software development practices and their automation through tools is no longer a luxury for development teams today, but part of their way of working. Unfortunately, in Colombia many of these helps and forms of work are not widely used. This paper presents the documentation and implementation of preventive quality tools and good practices for software development that allow code versioning, continuous integration, automation of functional tests, static code analysis and continuous deployment.

Objective: Present the good practices implemented in the Smart Campus Ecosystem case study for software development.

Methodology or method: Good practices for software development based on XP and DevOps are reviewed. A set of tools is selected for implementation that has a direct impact on the quality of software development. These tools are used in the UNIAJC Smart campus ecosystem case study. The results of the implementation are documented in this article.

Results: The preventive quality model is exposed, put on test and the results are documented.

Conclusions: The preventive quality model helps to increase the results of quality assurance through the set of tools that provide development teams with key information for refinement and refactoring of source code within development runtime and no later than this stage.

Keywords: DevOps, SQA, Software Quality Assurance, Smart Campus, Software Engineering.

Corresponding Author:

Manuel Alejandro Pastrana Pardo
Faculty of Engineering
Antonio Jose Camacho University Institution
Address. Av. 6 Nte. #29N-25, Cali, Valle del Cauca, Colombia
E-mail: mapastrana@admon.uniajc.edu.co

1. Introduction

In the software development industry, from 1994 until the date, the standish group has generated a report on the projects that turn out to be cases of success, failure and those who present scope changes within their execution. The objective of this measurement is to identify the main causes that prevent being successful within the planned scope, time and cost. The report presented in [1] indicates that the greatest number of problems are presented in 3 specific stages of the development process. The first stage is the review of the requirements, the second stage is the software design, and the third stage is the implied quality in the development process. In this last one, it exists two approaches, in one side, the conventional, which requires the software product to be made before, and subsequently to this, tests are performed to establish if the software performs or not with the requirement for its approval. This might take several iterations to be achieved. The second quality assurance approach, is called preventive and it goes more hand in hand with an agile philosophy, where it is indicated that it must exist a set of filters that allows during the development time, to measure and advice the team, in how their work is affecting the quality of the final product. In this way, during the same development time, all the needed measures are taken, so it might be adjusted before the delivery. For the [2] in its original work, generates 12 good practices, recommended by its well-known framework for the development of Extreme programming software or XP that is focused in increasing the development quality during the construction stage and not subsequent to it. These recommendations, are not so easy of implementing, they required a certain technical degree and the team disposition for the implementation. Other approaches that help mitigating the aforementioned problem, suggesting good practices through the implementation of automation tools, as the DevOps case.

They exist authors that have dedicated their work looking for alternatives for the implementations of these development models, due to the difficulty that involves for the work teams the change in the organizational culture such as [3] who mentions that the difficulty resides mainly, on the agile frameworks as Scrum and XP, which provide guidelines on what to do to organize a development process under good practices and a preventive approach, but they do not indicate how they should be carry out, what may cause a confusion in the development teamsdue that they do not have a punctual guidelines to follow up, living it to a free interpretation. Due to this reason and looking for a good way of align the good practices with tools who permit their implementation in a simpler way, in [4] it exposes a way that makes it possible, implementing different methods for the development of software as the collective characteristics of the code, the continuous integration, the continuous deployment, the code standard, the unit test and the sustainable rate.

As a case of study within the macro project smart campus ecosystem, of the University Institution Antonio Jose Camacho (UNIAJC) a quality assurance environment was implemented with practices that are supported in tools that help improving the results on projects of software development to be build. This model allows the collective ownership of the code, which means that the source code is always available to any member of the team in any moment, to be implemented through a version control tool.

This one is joined to two other tools, by one side, the static code analyzer, which allows to verify the performance with the programming standards, besides informing constantly about the use or the no use of good practices of codification according to the object-oriented programming language that are being used. In other hand you found the continuous integrator, which allows to verify if the changes made to the versioned code are affecting in a negative way the release. That is to say that every time that a change is done in the version control tool; this tool detects the changes and tries to generates a packaging for deployment. If it’s not able to do it, it will inform the team, that the change of one of its members is generating difficulties for the possibility of deployment. Even though the design suggested by [4] is effective and susceptible to improvements, due that in its initial scope it leaves out other practices like the deployment continuous and the functional test automation, due to this reason, the possibility of improving the model was seeing, through the inclusion of these two practices which increases the quality controls to which this project is submitted during the building stage.

2. Theoretical framework

DevOps could be defined as the set of principles and practices which allows to face in a different way the release of software products, increasing their quality through the active collaboration between the developing areas and company operations according [5].This relation is created in order to improve the integration and communication of these two areas, in order to obtain the benefits of modern approaches in software development. An example of this is a set of principles that are presented below for a better appreciation of the concept and that are proposed by [6]:

- **Action focused on the client:** The client requirement might slightly change during the construction of the service or product. If there is not a good communication, problems may arise at the moment of introducing a new function or new characteristic, that requires the client in a subsequent stage.
- **Create with the goal in mind:** Roll processes where each individual has a function, must be putted aside, instead a thought where all the context is visualized and understood and that composes the creation of the product or service, of the work that is being done.

- **responsibility end to end:** The development and operation areas must be totally responsible of their activities and ensure that they are done the best way, with the best results during all the life cycle of the product.
- **Autonomous and multifunctional teams:** It is highly recommended to have team members with a profile that has deep knowledge in a specific topic, but with some understanding in other topics.
- **Continuous improvement:** The DevOps culture promotes the continues improvement. The way of acquiring this thinking is through experience, therefor an open thinking to learn as much as it is possible from failures must be adopted.
- **Automate as much as you can:** it is not just about automate the continuous release of processes, consolidation and deployment, but to observe that other functions or characteristics may be automate in order to obtain a significant advantage.

These principles are implemented through different stages belonging to the life cycle of DevOps, according [6] it could be represented with the next 6 stages, even though it exists other variations which include more or less stages:

- **Planning:** In this stage, the initial setup is done to stablish the base lines of the project and the different tasks to be performed.
- **Development or construction:** Stage during the one the product is built. This is how, designs of infrastructure and tests are made.
- **Continuous integration:** Automate the mechanism of review, tests and alerts planed in the first stage of the cycle. The joint of the tests in each section will guarantee the correct performance, otherwise it will make evident the possible mistakes and the exact place of its origin.
- **Continuous deployment:** The deployments are made through tools or scripts with the goal of reducing the human intervention during the process and moreover reducing eventual mistakes.
- **Monitoring:** The parameters to be monitored are stablished to control the infrastructure and applications. For example, number of security bugs detected, number of security bugs resolved, number of bad programming practices, (or code smell) detected and resolved, current technical debt, number of successful continuous integrations, number of continuous integrations failed, rate of unitary test coverage, etc.
- **Operation:** Resources are defined through the growth of the application, through tools that facilitate the dynamic modification of the infrastructure in quality attributes: scalability, persistence, availability, transformation and security.
- **Continuous feedback:** In this stage the information presented is analyzed by everyone in order to allow the continuous improvement of the product and make decisions aimed to stablish, when the necessary adjustments should be applied.

DevOps doesn't just depend on communication strategies between development and operation areas. It also depends on what type of tools are used and how the members manage themselves, when using them, as well as on the organizational culture and the processes that are impacted by their implementation. This represents a challenge due to the ever-evolving variety of tools available. The objectives of the tools are to facilitate the work of the different areas, allow continuous release and maintain software reliability according to [7].

3. Methodology

The research processes used to investigate, document, perform the implementation and the tests is described hereafter:

3.1. State of the art

In the initial stage a research about devOps, good practices and preventive quality was made. The target of this item is to know in which development software stage are these topics and provide a knowledge basis, which exposes the great amplitude of possibilities that may be approached as future projects reference.
The first one is the work of [3], where a predictive research about the behavior of employees is presented in case of having a Devops implementation, which goes hand to hand with the resent work of [8] that exposes the advantages and problems of the DevOps inclusion in the organizational culture. The above, is based on the evaluation of the incertitude and the personality of employees. The target is to identify the advances, and problems that may arise, having the opportunity of creating strategies to mitigate and improve the

implementation process of DevOps. The identification process is done using surveys and structural model equations.
On the other side, this research made by [9] at IBM about the transformation of five projects business intelligence to secdevops, which is a DevOps application, where the security of information is prioritized. The research has as target, to identify which security aspects are generated during the transformation impact from a manual implementation process to an automated process. Additionally, it is valuable mentioning the great contribution provided by the original work of [2], since it provides a fundamental basis to model an agile and efficient work, but which, in turn, involves a great effort when using said fundamentals, due to this, important technical skills are needed. Of the fundamentals or good practices that it provides, the following stand out: continuous integration (CI), continuous deployment (CD), collective ownership of the code, code standard and unit tests. Some important works around DevOps have delved especially into CI and CD practices such as [10], who delves into various configurations of tools that allow these operations to be release and the difficulties encountered in their adoption. Likewise, these practices were important points in the work release by [4], where a model is presented, and which is based on the agile Scrum framework and some of the good XP practices to model an environment of preventive quality of software, implemented with free software tools. These practices and tools dovetail with DevOps recommendations as indicated [11] and [12] who suggest that preventive quality models should lead to the automation of the continuous submission of the process.

3.2. Tools used to implement good development practices:

Based on the good practices raised by [4] which include versioning, CI and static code analysis currently implemented in the Smart Campus ecosystem case study, other tools were investigated by way of example to expose different possible configurations, this allowed to identify if the current ones required optimization or not. Additionally, by understanding the model, it is possible to identify the need of a complement in aspects not covered, such as continuous deployment to both, the test environment and the production environment. Likewise, the implementation of functional test automation becomes a necessity which helps increasing the reviews release as the project progresses for each sprint executed.

Table 1. categorization of tool for the implementation of good development practices.

Category	Tools
Virtualization	VMware, KVM, Xen, VirtualBox
Containers	VMware, KVM, Xen, VirtualBox
Tests and building	Solano CI, Jenkins, Maven, Ant, Gradle
Application deployments	Capistrano, Red Hat Ansible
Version’s control	SVN, Git, Github, Gitlab, Bitbucket
Application servers	Weblogic, Glassfish, Tomcat, Jetty, Spring boot, Apache, Nginx
Monitoring	Kibana, Cacti, Sensu, Ganglia, Icinga
Logging	PaperTrail, Loggly, Splunk, SumoLogic
Process Supervisor	Systemd, GNU Emacs
Security	Snort, Acunetics
Data Basis	Postgres, Oracle, Apache Solr, MySQL, Redis, Firebase, MongoDB
Code Statistical Analysis	SonarQube, Pendantic

Currently there are various automation tools to implement good development practices. This means that using the model exposed by [4], different implementations can be configured and the choice of tools will depend on the objective and investment that the company wishes to make. The research indicates some representative tool options on the market, which may be alternatives to those implemented in the Smart Campus ecosystem. The results are shown in Table 1.

3.3. Tool selection

After reviewing the tools listed in Table 1, it was determined that those implemented in the Smart campus ecosystem do not require changes, since they correctly fulfill their function. In the same way, it seeks to complement the model of [4] with the minimum of possible changes to avoid a migration that can be expensive in terms of the time involved. Therefore, to complement the versioning model, CI and static code analysis, the CD practices are included, provided by the same CI tool. Regarding the automation of functional tests from the CD, the Katalon tool was selected. In this case, the use of free software is used, aligning DevOps principles and tools, with good practices suggested in [2]. That is, the model and the practices described in [4] would be expanded with the two practices mentioned. Table 2 presents a comparison that allows to implement good practices through tools, detailing the selected tools below and clarifying that versioning, static code analysis, continuous integration and unit tests are already part of the culture of Smart campus ecosystem case study work and are implemented under the model of [4]:

- **Git:** According to [13], it is a free software designed by Linus Torvalds that is used to control the evolutionary changes of a software product, assigning a version to the code to generate order and traceability on its modifications.
- **SonarQube:** According to [14], it is a tool that is in charge of performing code reviews automatically in order to find errors, vulnerabilities and the so-called code smells or, due to their translation into Spanish, odorous code. SonarQube is responsible for finding inconsistencies in the code in the different branches of the project.
- **Jenkins:** As [15]indicates, it is a server that allows to automate all kinds of actions, such as releases and tests, among others. the role in this project is to automate testing and deployment.
- **Katalon Studio:** As stated in [16], it is a free automate software for web, API and mobile testing. It offers a wide range of possible configurations that allows easy integration with Jenkins and Git. In addition, it is focused on users with limited knowledge, unlike tools like Selenium and Cypress which require more technical skills to be use them properly.
- **PostgreSQL:** As mentioned in [17], it is a relational database system that uses the SQL language whose main characteristic is the safe storage and scaling of tasks. The role it fulfills in this configuration is to provide the service to Sonar in order to store information about executed tests, among others.

Table 2. XP Good Practices Vs. Automate DevOps tools

XP Good Practices	DevOps	Tools
Collective ownership of the code	Versions Control	Git, Github, Gitlab, Bitbucket
Code standard	Static code Analyzer	Sonar, Kiuwan
Continuous Integration	CI	SolanoCI, Jenkins, TravisCI
continuous deployment	CD	Jenkins, Heroku
Unitary test	Development Framework	JUnit, PHPUnit

Therefore, the new model that is represented in Figure. 1 would initiate its interaction with development teams through the GitLab community edition 12.7.5 version control tool. Here the development teams will centralize the evolutionary changes of the development, guaranteeing traceability, recovery from failures and availability at all times of the code for any interested party. The changes made to the repository will be detected by the Jenkins tool, which must trigger a set of actions that guarantee the automate of preventive quality reports for the entire team. The first action that arises is the continuous integration (CI) performed by Jenkins 2.22.1 that allows generating a deployment unit with the latest changes made, identifying whether or not a change has been generated syntactically that prevents it and notifying the result by email. If there is a failure, it notifies the user in charge that it was not possible to perform the action, so that an action can be taken immediately.

This prevents that at the moment of making the integration of changes, facing a release, may happened a contingency that delays it. In case of achieving the generation of the deployment unit, the tool triggers three actions in parallel. on one side, it asks sonar community edition 8.2 to make a static code review to verify its quality with respect/respecting to international standards of good practice. here security revisions of coding mistakes, or that can trigger errors over time are highlighted (referred in the tool as bugs) or of bad programming practices called odorous code. The accumulation of things to be solved involves a code refactoring time called technical debt, which is measured by the tool to indicate to the team how much time should be involved in applying the improvements. Additionally, the tool measures the results of the unit tests and the rate of their coverage. The higher the coverage rate, the better the quality assurance result in development time. On the other hand, Jenkins connects with the respective docker container in which the application server where the project is to be deployed is located. This is done depending on whether it belongs to the development test server (SmartDev which is a Dell PowerEdge 2900) or whether it belongs to the final production deployment environment (SmartCampus which is an HP ProLiant DL210 Gen9).

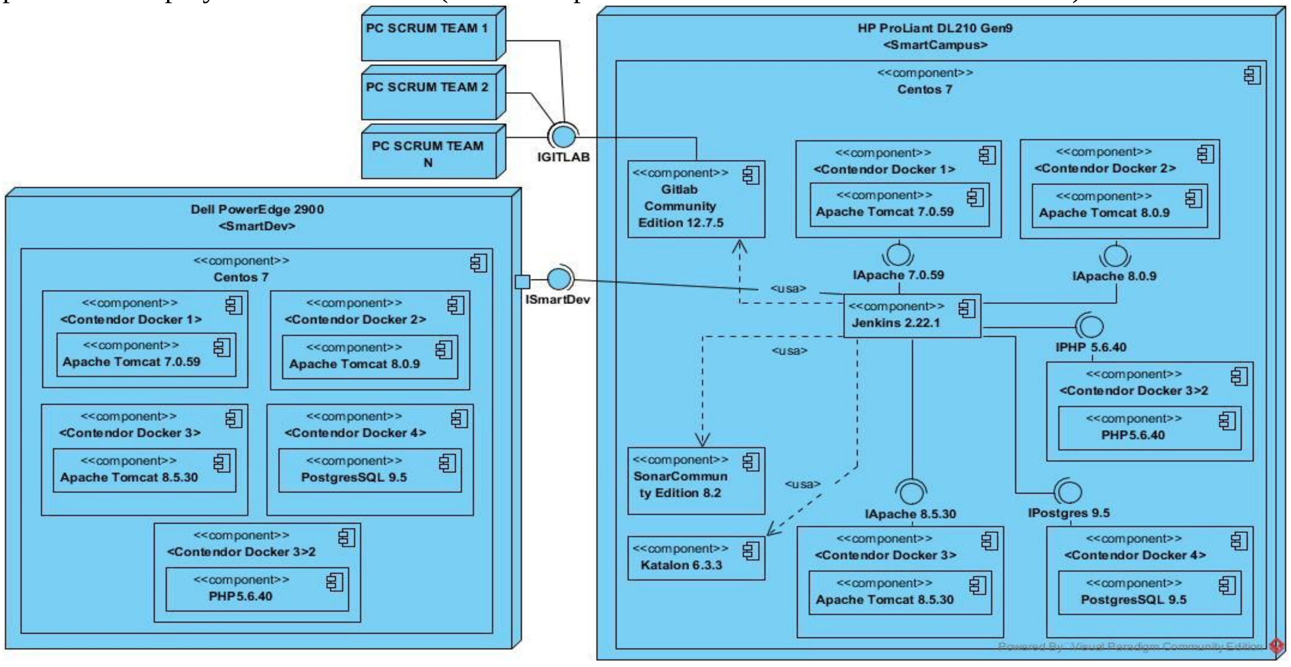
PC SCRUM TEAM 1
PC SCRUM TEAM 2
PC SCRUM TEAM N
IGITLAB
HP ProLiant DL210 Gen9
<SmartCampus>
<<component>>
Centos 7
<<component>> Gitlab Community Edition 12.7.5
<<component>> <Contendor Docker 1>
<<component>> Apache Tomcat 7.0.59
<<component>> <Contendor Docker 2>
<<component>> Apache Tomcat 8.0.9
IApache 7.0.59
IApache 8.0.9
<usa>
<<component>> Jenkins 2.22.1
IPHP 5.6.40
<usa>
<<component>> <Contendor Docker 3>2
<<component>> PHP5.6.40
<<component>> SonarCommunity Edition 8.2
<usa>
IApache 8.5.30
IPostgres 9.5
<<component>> Katalon 6.3.3
<<component>> <Contendor Docker 3>
<<component>> Apache Tomcat 8.5.30
<<component>> <Contendor Docker 4>
<<component>> PostgresSQL 9.5
Dell PowerEdge 2900
<SmartDev>
<<component>>
Centos 7
ISmartDev
<<component>> <Contendor Docker 1>
<<component>> Apache Tomcat 7.0.59
<<component>> <Contendor Docker 2>
<<component>> Apache Tomcat 8.0.9
<<component>> <Contendor Docker 3>
<<component>> Apache Tomcat 8.5.30
<<component>> <Contendor Docker 4>
<<component>> PostgresSQL 9.5
<<component>> <Contendor Docker 3>2
<<component>> PHP5.6.40
Powered By: Visual Paradigm Community Edition

Figure 1. Model implemented in Smart Campus. Source the authors.

This allows the applicatives to be put into operation quickly. once the deployment is done, the tests cycle, configurated previously is triggered in the katalon tool, so that automatically the functional correctness of the application may be reviewed.
Additionally, it is necessary to give a brief explanation about continuous integration (CI) and continuous deployment (CD), since they directly influence the model proposed by [4] and consequently, in this article.

3.3.1. Continuous Integration

Also known as CI for its acronym in English, it is a good development practice that consists in identifying if the changes to the source code that have been entered into version control tool affects the deployment unit of the project. In the event that these changes have a negative effect, the tool is able to recognize it and report that it is not possible to perform the requested task and indicates who has made the last change. This allows syntax, configuration or settings mistakes to be resolved immediately and not in an advanced stage of the project, where it is no longer easy to remember what changes have been done. This practice may be complemented with the continuous deployment in a natural way, due that the ready-to-deploy version can be put on an application server to perform functional tests. In this way, a significant reduction in time is achieved when finding and solving mistakes in early development stages, avoiding that mistake become increasingly tedious to solve in later stages according to [4]. It is important to note that the impact of the implementation of this practice may have greater significance in more complex projects and with a greater number of people in the team. the advantages generated by the inclusion of this practice within the development process are the following according to [4]:

- Having a central repository of source code versions allows to keep records of code changes, identify stable versions of the project, identify who has made changes, and recover from failures. Therefore, when a problem is detected, it is known exactly who made the last change and MAY be corrected quickly.
- Automate the process of changes allows not to have negative surprises when making releases. which guarantees from the beginning of the project being prepared for the deployment.
- Performing unitary tests automatically, allows the development team to guarantee, during development time, that it has successfully achieved the solution of the needs specified in the project requirements in the analysis artefact.
- Reduces the feedback time of mistakes with the client.
- Strengthens the confidence of programmers when uploading THE code.
- Continuous availability of compilations.

On the other hand, it is also important to mention the disadvantages of this practice, which are not many, but it is very important to pay attention to them and find a way to reduce their negative impact:

- Overload due to system maintenance.
- Primary need of a server for compilations.
- Change of mentality of the entire work team, so that its implementation and use is effective.

3.3.2. Continuous deployment

Also known as CD for its acronym in English, it is a good practice that aims to implement a deliverable in the desired environment (of test or production), reducing the cycles of the generation of new versions to a couple of weeks, days or hours and in this way releasing value quickly, reliably and repeatedly at a lower cost. In other words, it is to automate the entire workflow to simplify the fast release of the software as indicated [18].

3.4. Tool Settings

The previously selected tools were configured in a virtual machine with CentOS 7 operating system, a replica of the Smart Campus ecosystem test environment. This practice was carried out in order to validate the operation in a real development environment, with few variations in the configuration, allowing to measure the performance of the additional Katalon tool. In the initial configurations of the virtual machine, the tools of the model of [4] which are GitLab, Jenkins and Sonar were implemented. Additionally, continuous deployment with GitLab and automated functional tests with Katalon were implemented.
Within the Jenkins application, which allows the necessary integrations with the other tools of the quality environment, through configurations through the plugins, the installation of what is required was carried out to display the reports provided by each of the tools. In the case of katalon it is called the Katalon Studio Plugin.
As the first step in the installation of the katalon tool on the server, it is necessary to create an account at https://www.katalon.com in order to use the tool. Additionally, as the virtual machine's operating system does not have a graphical interface, it is required to install Google Chrome, since the Katalon tests are executed in a browser. To do this, the XVFB plug-in must be used, which will be in charge of executing these tests in the virtual memory of the system, without the need for a graphical environment. The following commands allow the installation of the aforementioned:

- wget -O google-chrome-stable_current_x86_64.rpm https://dl.google.com/linux/direct/google-chrome-stable_current_x86_64.rpm
- sudo yum install xorg-x11-server-Xvfb.x86_64

The tests were built within a graphical environment such as Windows, where it was possible to generate the CMD code that contains the necessary commands. Among these, the following can be highlighted: location of the project file that will be executed, the browser where the tests will be executed, the API Key of each user and the Test Suite that will be executed. In addition, the tests created were added to a GitLab repository, being able to facilitate their portability. In this way it was possible to use files with the extension. prj, Test Cases, Test Suites and the reports within the execution of the project previously configured in Jenkins on the server.

3.4.1. Implementation challenges

When re-deploying, there will always be challenges in adapting new tools and practices. The result depends on many factors, ranging from the personal skills of the employees to the structure of the organization. Below are the challenges encountered when implementing DevOps that have been reported in previous studies such as [19]:

- Significant problems to implement an automated process when variations imply changes in the database. So, it is recommended that projects use a database versioner such as flywayDB in java, for example.
- Significant problems to implement an automated process when variations imply changes in the database. So, it is recommended that projects use a database versioner such as flywayDB in java, for example.
- Difficulty for software developers and system administrators in learning new technologies, tools and methods. Therefore, a training process is necessary to adopt new practices and tools, managing to provide the necessary information on the benefits and practical uses of what is proposed and allowing them to be integrated into the organizational culture.
- Difficulty in fully automating the implementation process, due to changes in infrastructure management. Therefore, the identification of the development and operations process must be correctly mapped and must contain adequate integration and communication mechanisms.

4. Results

The UMATA (Mobile) applications, coded in Ionic and Class Room Reservations (Web), coded in php with Laravel, were taken as test projects. On the part of the Web application, a good connection was obtained with the entire implemented system. It was possible to build the tests (Test Cases) correctly in Katalon. In addition, the integration of Jenkins with Sonar was achieved to perform the static code analysis of the application. From what was previously mentioned about the tests, it can be said that this characteristic generated an improvement in the model, as evidenced in Figure 3, where it is observed that the automated tests are superimposed on the manual ones in relation to the time it takes to perform them. It is important to note that results may vary, due to directly influencing external factors, for example a person's ability to perform tests manually or the processing power of the equipment that runs the automated tests. The difference will be noticeable in larger modules and with more tasks to be carried out, this could be the filling of a form or the registration of several users consecutively.

Table 3. Manual tests (M) Vs Automate test (A) in Katalon.

Test case	M (seconds)	A (seconds)
help	19	11.330
Inicial Menu Navegation	14	11.964
Navigation Available Rooms	30	18.373
Navigation book rooms	26	17.974
SeeMyReservations	12	15.625
TOTAL	112	75.266

On the part of the Katalon console version, that is, the Katalon Runtime Engine, it did not interpret the Test Cases correctly. This is because additional configurations were required to allow the latest plug-ins to be installed and to function properly. Such a task requires a more advanced knowledge of the use of Selenium and Katalon. The required adjustments were made and the tests were run again obtaining a favorable result.
In the case of the UMATA application, at the time of building the automated tests in Katalon, the first drawbacks/disadvantages arose, due that to emulate the application, plug-ins had to be installed that would allow its execution in the web browser and thus have the possibility to record the evidence. Later, in order to run the application correctly, it was necessary to automatically open a series of Web sites that required users to authenticate. When these sites were opened, the login was enabled to enter and consult the information or use the functionalities that it contained. This was a challenge, since it was necessary to program the opening

of the additional tabs and enter the web addresses, but the confirmation could not be made, because Katalon does not record the clicks and pulsations on these web pages. In this case, better skills were needed in the programming of the Katalon tool and in the Ionic language to automate processes of this type or more complex. It MAY be highlighted from this exercise that despite the fact that the environment exists, it works differently for each programming language with the one the solutions have been build, due to the fact that fewer or more additional configurations are required for each case, especially the cases of Web applications are easier to test in the tool than mobile application cases. As for Katalon, the previous tests had a negative result compared to the results obtained with the web application encoded in pure PHP. This is mainly due to the fact that this application did not need a large assembly for its operation and that its programming structure did not contain plugins or styles, allowing Katalon to correctly interpret the recorded steps.

5. Conclusions

From the results obtained, it is concluded that an automated system may increase the quality and significantly reduce release times. However, the aim is not to put aside other types of tests, such as unitary tests, for example, but rather to complement them, because even though it is an automated system, it is not 100% reliable. There will be cases where the analyzes will generate false positives or no errors will be found and the idea is to avoid cases where the propagation of these, may become a snowball, making the maintainability and scalability of the application tedious. On the other hand, it is important to mention that the number of tests carried out are few and therefore the results may vary in other types of configurations and with other types of projects. It is recommended to use this type of system together with conventional tests to raise the quality of the product. It may seem tedious at first, but when you develop the skills to run fluently the analyzes, great results will be achieved. It is important to note that in some cases the impact will be greater in more complex projects and with more people involved in the areas. In addition, the environment configurations may vary greatly depending on the application to be tested, that is, a web application will not have the same characteristics as a mobile one, each one requires a different level of knowledge about the language and the necessary configurations for its correct emulation. Another important factor highlighted during the tests is that the level of adaptation of Katalon will also depend to a great extent on the ability to program the automation of steps, in some cases it will be more complex than in others. For future works, a more in-depth investigation of the non-traditional languages supported by the environment will be able to be approached and to identify which ones could not be examined correctly, in addition to the correct programming using Selenium and Katalon to automate more complex processes, developing a configuration guide. of various types of projects.

References

[1] Johnson James, "CHAOS Report: Decision Latency Theory: It Is All About the Interval - James Johnson - Google Libros," 2018. [Online]. Available: https://books.google.com.co/books?hl=es&lr=&id=WV1QDwAAQBAJ&oi=fnd&pg=PA1&dq=CHAOS+Report+2018&ots=9_CUVJxL_j&sig=-o6C1KfyFn2rDEKyUy-NgIQ72Mw#v=onepage&q=CHAOS Report 2018&f=false. [Accessed: 03-Jun-2021].

[2] K. Beck, *Extreme Programming Explained: Embrace Change*, no. c. 1999.

[3] J. Shropshire, P. Menard, and B. Sweeney, "Uncertainty, Personality, and Attitudes toward DevOps," *AMCIS 2017 Proc.*, Aug. 2017.

[4] M. Pastrana, H. Ordoñez, A. Rojas, and A. Ordoñez, "Ensuring Compliance with Sprint Requirements in SCRUM: Preventive Quality Assurance in SCRUM," in *Advances in Intelligent Systems and Computing*, 2019, vol. 924, pp. 33–45.

[5] B. B. Nicolau de França, H. Jeronimo, and G. H. Travassos, "Characterizing DevOps by hearing multiple voices," in *ACM International Conference Proceeding Series*, 2016, pp. 53–62.

[6] DevOps Agile Skills Association, "6 Principles of DevOps – DevOps Agile Skills Association (DASA)," 2019. [Online]. Available: https://www.devopsagileskills.org/dasa-devops-principles/. [Accessed: 02-Jun-2021].

[7] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of DevOps concepts and challenges," *ACM Computing Surveys*, vol. 52, no. 6. Association for Computing Machinery, 01-Nov-2019.

[8] 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.

[9] V. Mohan, L. Ben Othmane, and A. Kres, "BP: Security concerns and best practices for automation of software deployment processes: An industrial case study," in *Proceedings - 2018 IEEE Cybersecurity Development Conference, SecDev 2018*, 2018, pp. 21–28.

[10] S. Vadapalli, *DevOps: Continuous Delivery, Integration, and Deployment with DevOps Dive into the core DevOps strategies*. 2018.

[11] F. M. A. Erich, C. Amrit, and M. Daneva, "A qualitative study of DevOps usage in practice," in *Journal of Software: Evolution and Process*, 2017, vol. 29, no. 6.

[12] G. Rong, Z. Jin, H. Zhang, Y. Zhang, W. Ye, and D. Shao, "DevDocOps: Towards Automated Documentation for DevOps," in *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, 2019, pp. 243–252.

[13] Chacon Scott and Straub Ben, "Git - Book," *Apress*, 2020. [Online]. Available: https://git-scm.com/book/es/v2. [Accessed: 02-Jun-2021].

[14] SonarQube Official Website, "SonarQube Documentation | SonarQube Docs," 2020. [Online]. Available: https://docs.sonarqube.org/latest/. [Accessed: 02-Jun-2021].

[15] Jenkins Official Website, "Jenkins User Documentation," 2020. [Online]. Available: https://www.jenkins.io/doc/. [Accessed: 02-Jun-2021].

[16] Katalon Official Website, "Welcome to Katalon Docs | Katalon Docs," 2019. [Online]. Available: https://docs.katalon.com/katalon-studio/docs/index.html#products. [Accessed: 02-Jun-2021].

[17] PostgreSQL Official Website, "PostgreSQL: About," 2020. [Online]. Available: https://www.postgresql.org/about/. [Accessed: 02-Jun-2021].

[18] P. Rodríguez *et al.*, "Continuous deployment of software intensive products and services: A systematic mapping study," *J. Syst. Softw.*, vol. 123, pp. 263–291, Jan. 2017.

[19] L. E. Lwakatare *et al.*, "DevOps in practice: A multiple case study of five companies," *Inf. Softw. Technol.*, vol. 114, no. June, pp. 217–230, 2019.