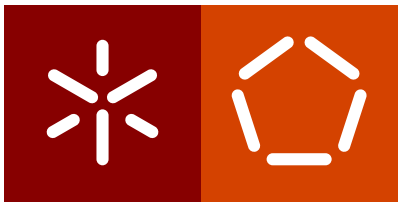
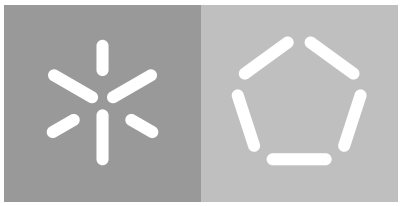


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Guilherme dos Santos Rodrigues

DevOps - The New Development Process

October 2016



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Guilherme dos Santos Rodrigues

DevOps - The New Development Process

Master dissertation
Master Degree in Computer Engineering

Dissertation supervised by
António Nestor Ribeiro

October 2016

ABSTRACT

As result of the disruption caused by the appearance of digital companies with an agile delivery model, there is a need to review the delivery model in organisations. The *DevOps* philosophy aims to provide an answer to this problem, bringing all of the individuals responsible for the delivery to work collaboratively, with the support of a set of tools to automate and streamline all of the processes. The objective of this dissertation is to re-evaluate and, consequently, improve the delivery model within Deloitte, proposing an automated process for code analysis, functional tests and deployment of software packages. To this end, a set of cutting-edge tools were analysed and a case study in the context of a real project was built, in order to put into practice the automation of the processes developed.

Keywords: DevOps / Continuous Delivery / Automation

RESUMO

Resultado da disrupção causada pelo surgimento de empresas digitais com um modelo de entrega ágil, existe a necessidade de rever o modelo de entrega nas organizações. A filosofia *DevOps* pretende dar resposta a esta problemática, aproximando os intervenientes responsáveis pela entrega, com o apoio de um conjunto de ferramentas que permitem automatizar e otimizar todos os processos. O objetivo deste trabalho passa por reavaliar e, consequentemente, melhorar o modelo de entrega de projetos dentro da Deloitte, propondo para isso uma automatização dos processos de análise automática de código, de testes funcionais e de criação automática de pacotes de *software*. Para esse efeito, foram analisadas um conjunto de ferramentas de última geração e elaborado um caso de estudo no contexto de um projeto real, de modo a colocar em prática a automatização dos processos desenvolvidos.

Palavras-chave: DevOps / Entrega Contínua / Automação

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 CONTEXT

As result of the disruption caused by the appearance of companies with an agile delivery model, there is a need to review the delivery model in most organisations.

One of the recent developments in this area, called DevOps [KBS13], aims to strengthen the relationship between the development and operations teams, bringing them to work closer together, aided by a wide variety of tools for requirements, development, testing, deployment and monitoring.

DevOps is not tailored to any particular type of organisation as it can be partially applied on any individual process of the delivery pipeline. However, each company must identify the areas in which the DevOps philosophy could make an impact for both the development of the software as well as to the business. Some of the common measurable goals are [Wal13]:

- Reduce time-to-market for new features;
- Increase overall availability of the product;
- Reduce the time it takes to deploy a software release;
- Increase the percentage of defects detected in testing before production release;
- Make more efficient use of hardware infrastructure;
- Provide performance and user feedback to the product manager in a timelier manner.

DevOps is, in a nutshell, a movement which aims to achieve better collaboration between the organisational entities of development and operations. This movement stresses collaboration, communication and integration between development and operations teams. Rather than seeing these entities not working well together, DevOps recognises the interdependence of software development and IT operations, bearing in mind that the final goal of organisations is to produce quality software and IT services more rapidly, with frequent iterations.

1.2 MOTIVATION

For the past several years, competition between organisations has been increasing. The need to assure to the client the quality and timings of the delivery, having in mind a reduced project budget, is of the utmost importance in order to become competitive in a global market.

In order to accomplish this, there is the need to change the current mind-set in the delivery process among organisations and this is where DevOps philosophy comes into place. DevOps is about changing the culture and improving collaboration between development and operations, as well as automating as many of the common jobs in the delivery pipeline. Most of this can be made possible with the creation and exploitation of a set of tools.

There are, according to the IT Revolution Press[1], three main principles in which DevOps is based on:

- Emphasize the performance of the entire system, as opposed to the performance of a specific department (e.g. Development or IT Operations);
- Shorten and amplify feedback loops so that necessary corrections can be continually made;
- Create a culture that fosters continual experimentation, taking risks and learning from failure, and understanding that repetition and practice is the prerequisite to mastery.

According to Puppet Labs annual report [PL15], high-performing IT organisations that have embraced DevOps deploy 30 times more frequently with 200 times shorter lead times[2] and they have 60 times fewer failures and recover 168 times faster. In every organisation, failures cannot be avoided, but it is clear that with the help of this new DevOps philosophy, aided by the right set of tools, that detecting and recovering from failure are easier to accomplish, and that can mean the difference between leading the market and getting caught up by the competition.

1.3 OBJECTIVES

Having set the initial background, the underlying objectives to this master's dissertation are the following:

- Describe the development lifecycle of a solution under the DevOps philosophy;
- Describe an automated code analysis strategy;

1 http://itrevolution.com
2 http://www.investopedia.com/terms/l/leadtime.asp

- Describe an automated end to end testing strategy from the requirement phase until the software testing;
- Describe a strategy to promote software packages through multiple environments and deployments to production environment;
- Analyse a set of tools, with the intent to support all of the previously enumerated strategies.

After describing the lifecycle, model, strategies and the suite of tools, the final goal is to put all of them in practise. In order to do so, it will be created a demo application to show all of the necessary steps to configure and use the selected tools, simulating a portion of the DevOps flow. The demo will be performed on the Misys core banking application Essence with the following inputs:

- The requirement will be to add a new custom field to a screen;
- The field will have business logic behind;
- The code will be analysed, prior to being pushed to the repository;
- An automated test will be triggered in order to validate the functionality of the requirement;
- A software package will be built in case the tests finish with success.

Besides the previously enumerated objectives, this dissertation is aligned with Deloitte's expectations for the near future, which is to bring DevOps culture and techniques to their projects. Therefore, the outcome of this dissertation will contribute with concrete results and a practical use case to prove that this philosophy can bring value to the company, as well as to the developers and testers.

1.4 Structure of the Dissertation

This dissertation is organised as follows:

- Chapter 1 - Introduction - sets the initial context, motivation and objectives of this dissertation;
- Chapter 2 - State of the Art - introduces the Waterfall and Agile methodologies, the DevOps philosophy, set of DevOps tools chosen to support the study held by this dissertation in the several phases of a project: requirements, development, testing, deployment and monitoring phases, and the CALMS framework which is used in the DevOps approach;

- Chapter 3 - Problems and its Challenges - presents the problems and challenges that can be encountered when using the DevOps approach;
- Chapter 4 - Development - highlight the main decisions that were taken throughout the dissertation, all of the scripts, jobs and artefacts that were created and the outcome of the flows that were automated;
- Chapter 5 - Conclusion and Future Work - presents the dissertation conclusions and possible future work.

2

STATE OF THE ART

2.1 WATERFALL APPROACH

For the past decades, it was almost a certainty to find the traditional Waterfall approach [AD12] being used inside the organisation's software delivery projects. This approach is mainly based on the following phases [FBK14]:

1. Requirement Gathering: In this initial phase, the development teams are required to promote meetings with the client, note down and detail the list of functionalities and software behaviours to be developed. These requirements can be split up into System and Business Logic Requirements that will eventually be translated into technical requirements.

2. Development: During this phase, the development team converts the Functional and Non-Functional Requirements into a complete information system, it is where the code is actually written.

3. Deployment: The aim of this phase is to produce software packages which will then be deployed to other environments. In Table 3, it is presented the major tasks that compose the deployment process[1]:

Table 1.: Major Deployment Phase Tasks and Owners

Major Tasks	Owners
Completing deployment preparations: The team updates the deployment, plan installs, configures, and tests hardware and software components.	Release Management, Development
Creating operations procedures: The team creates and documents procedures and defines checkpoints to help the operations team monitor and maintain the solution.	Release Management, Development
Deploying the solution: The team deploys the core technology and completes site deployments.	Release Management, Development

1 https://technet.microsoft.com/en-us/library/bb497043.aspx

Table 1.: Major Deployment Phase Tasks and Owners

Major Tasks	Owners
Stabilising the deployment: Project team and operations work toward a predefined state of completion for the solution.	Project Team
Transferring ownership to operations: The team formally hands over responsibility for the solution to the operations team.	Release Management
Closing the Deploying Phase: The team meets the Deployment Complete Milestone requirements and later completes post-project reviews with the customer and project team.	Project Team

4. Tests: This is the phase where it is assessed if the functionalities developed and deployed in the system matches the initial client requirements. There are multiple levels of testing, from System Integrated Tests[2] to User Acceptance Tests[3] and Performance Tests. Based on the tests results, there might be the need to go back to the development phase to fix bugs and repeat the process.

Figure 1 sums up the 4 phases that were detailed above.

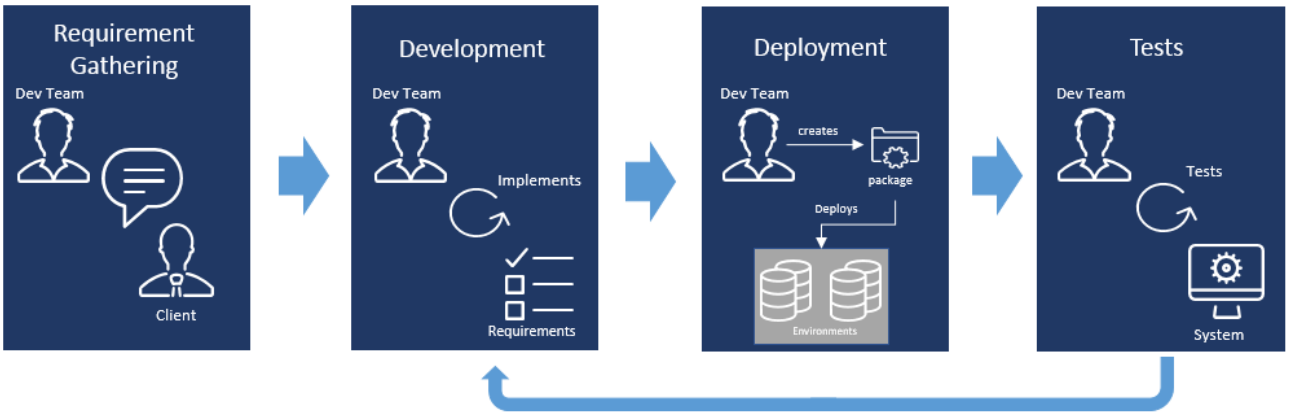


Figure 1.: Waterfall Approach[4]

2.1.1 *Potential Issues*

With the approach that was above described and is presently followed by most companies, there are four major potential drawbacks to it. They are as follows:

2 https://www.techopedia.com/definition/24590/system-integration-testing-sit
3 https://www.techopedia.com/definition/3887/user-acceptance-testing-uat
4 © 2015 Deloitte

1. Increased Delivery Time: There is an increased development time because it is not done in batches, which implies that the feedback cycles are also longer. In conclusion, it takes a long time to get results.

2. Complex Deployments: With higher number of requirements to be developed by the team in just a single deployment, the probability of having a higher number of defects is substantially higher as well.

3. Rigid Approach: By having a rigid approach, it actually means that the management team will have a very hard time when performing any scope task. It is extremely difficult to integrate any kind of changes, once the project starts and the scope of it is defined.

4. Increased Risk: The fact of having a scope with a higher number of requirements delivered at once increases the risks of a deployment, as well as increasing the need to perform more complex tests in the system once the deployment is completed.

2.2 AGILE APPROACH

Early in 2001, and due to the need of discussing alternative development methods from the traditional Waterfall approach that involved heavy preplanning, a group of software developers got together and out of that meeting, the Agile Manifesto[5] was created. Table 2 resumes the principles[6] in which the Agile Manifesto is guided by:

Table 2.: Agile Manifesto

Individuals and Interactions	over	Processes and Tools
Working Software	over	Comprehensive Documentation
Customer Collaboration	over	Contract Negotiation
Responding to Change	over	Following a Plan

Following the manifesto above detailed, the agile methodology [DM14], underlines direct and frequent communication between client and developer through its incremental processes. The proposed solutions are evolved through communication and collaboration of organizations and teams. Therefore, the key success of agile methodology relies on communication among team members and the ability to adapt to rapid changes.

Agile software development includes the notion of iterative cycles, where all of the phases are interconnected, each phase being a feedback mechanism for the others [LS11]. Basically,

5 http://agilemanifesto.org/
6 http://agilemanifesto.org/principles.html

it is acknowledge that no phase is ever completed, meaning that all phases can keep evolving. This contrasts with the waterfall schedule, which assumes that each phase reaches a logical conclusion before the next phase is eligible to start.

Figure 2 illustrates the phases that are comprised in the agile development process.

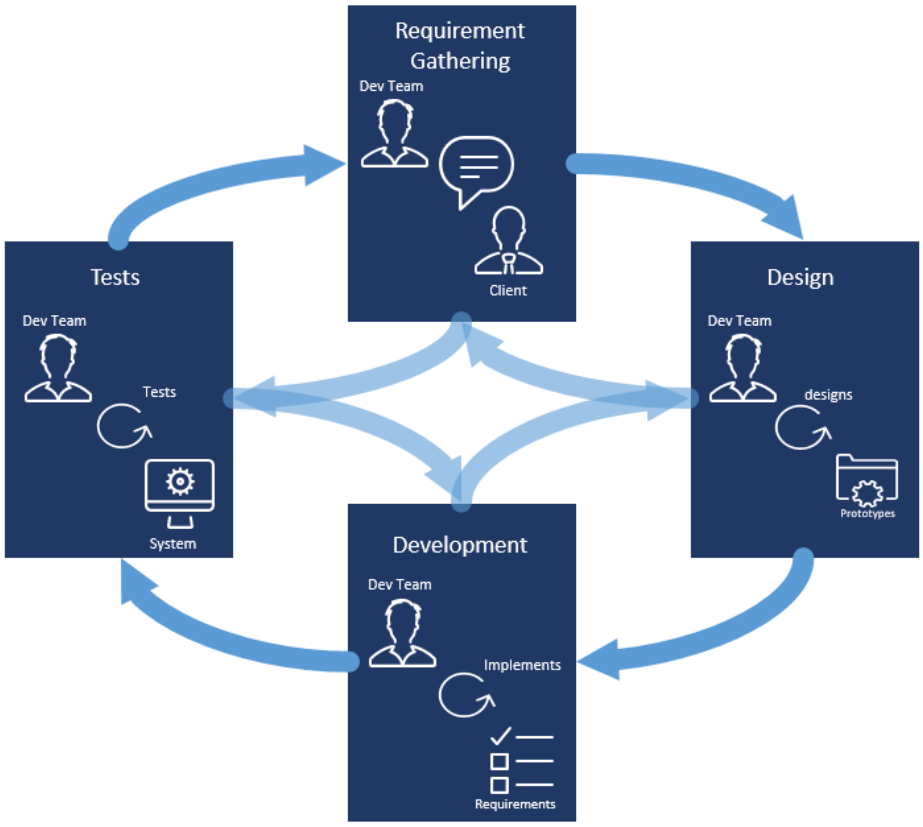


Figure 2.: Agile Approach[7]

2.3 DEVOPS APPROACH

The Agile software development has been around for over the last 20 years, and it was the first approach to move away from the traditional Waterfall method by shifting the software development towards a continuous development lifecycle. However, it did not include the operations side, meaning that all of the deployment activities were still Waterfall oriented.

In DevOps, it is intended to extend the continuous development goals that the Agile methodology proposes to continuous integration and releases. In order to perform these continuous releases, DevOps suggests automation of the change, configuration and release processes.

7 © 2015 Deloitte

Dev is used as a shorthand for developers in particular, but in practice it is even wider and it means that "all the people involved in developing the product", that include the product, Quality Assurance (QA), and other kinds of disciplines. Ops is a blanket term for systems engineers, system administrators, operations staff, release engineers, DBAs, network engineers, security professionals and various other sub-disciplines and job titles.[8]

DevOps is, in a nutshell, a movement which aims to achieve better collaboration between the organisational entities of development and operations. This movement stresses collaboration, communication and integration between development and operations teams. Rather than seeing these entities not working well together, DevOps recognises the interdependence of software development and IT operations, bearing in mind that the final goal of organisations is to produce quality software and IT services more rapidly, with frequent iterations.

2.3.1 *DevOps Principles*

As mentioned before, there needs to be a change in the mind-set of the two groups, with the goal of working closer together. The other component to the formula is applying the right framework and tools.

A framework and a set of new tools are needed in this fast pace world to aid both the development and operations teams. The recommended framework and an overview of the most used tools will be explored in more detail in subchapters bellow.

Gene Kim, one of the major contributors to this philosophy, states in the IT Revolution Press[9] that there are three principles, also referred as "ways", underpinning DevOps:

- The First Way emphasizes the performance of the entire system, as opposed to the performance of a specific silo of work or department — this as can be as large a division (e.g., Development or IT Operations) or as small as an individual contributor (e.g., a developer, system administrator).

Figure 3.: The First Way: Systems Thinking

8 http://theagileadmin.com/what-is-devops/
9 http://itrevolution.com/the-three-ways-principles-underpinning-devops/

- The Second Way is about creating the right to left feedback loops. The goal of almost any process improvement initiative is to shorten and amplify feedback loops so necessary corrections can be continually made.

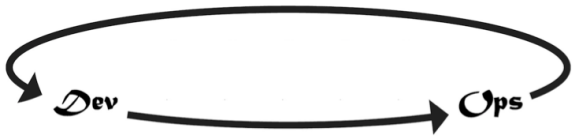


Figure 4.: The Second Way: Amplify Feedback Loops

- The Third Way is about creating a culture that fosters two things: continual experimentation, taking risks and learning from failure; and understanding that repetition and practice is the prerequisite to mastery.

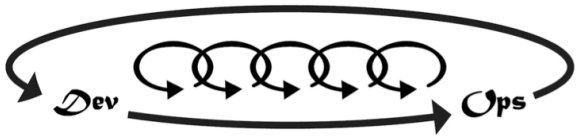


Figure 5.: The Third Way: Cultural of Continual Experimentation and Learning

2.3.2 *DevOps Techniques*

In order to adopt DevOps there are specific techniques [SC15] that need to be included and mastered. They are as follows:

- Continuous Improvement [HF10]: Technique used for identifying opportunities for streamlining work and reducing waste. It is an approach to work systematically to achieve small, incremental changes in processes in order to improve efficiency and quality. Figure 6 illustrates the lifecycle of this technique, composed of 4 phases:
  - Identify - Look out for opportunities in the process;
  - Plan - Schedule tasks to improve the process;
  - Execute - Implement changes;
  - Review - Monitor the effects of the changes implemented.

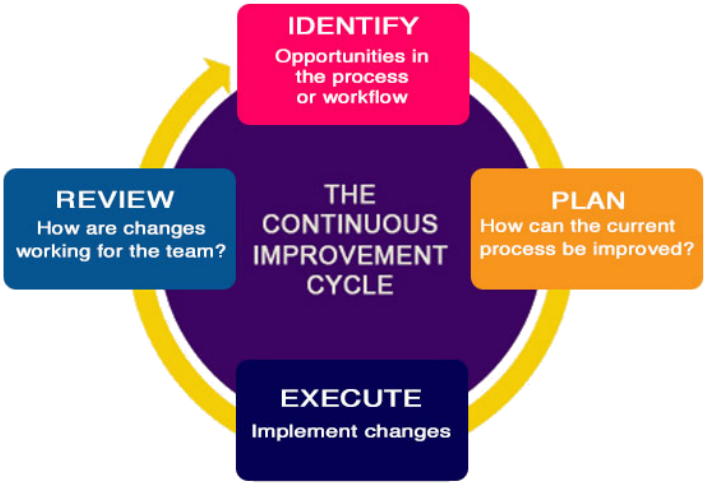


Figure 6.: Continuous Improvement Lifecycle[10]

- Release Planning [JLG+11]: Technique used to defining release roadmaps, project plans, and delivery schedules, allowing the team to have a common vision about what needs to be achieved.

- Continuous Integration [Vir15]: Technique that indicates that integration should happen at an early stage of the development process, share the changes between the multiple teams and analyse how the environment react to those changes. In addition, it is intended to automate this integration, such that as soon as the system detects changes, it should integrate them into the system to be then automatically built and tested.

- Continuous Delivery [Che15]: Technique used in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time, meaning that, at any point of time, the system has always the latest working software. Illustrated in Figure 7, some of the benefits of this technique are:
  - Accelerated time to market - Continuous Delivery allows companies to deliver new software releases to customers more quickly, helping them to stay a step ahead of the competition;
  - Improved product quality - With Continuous Delivery technique, developers can have their code immediately tested after performing commit. If the tests find a problem, developers are able to fix it before moving on to another task;
  - Reliable Releases - With the improved product quality, the risks associated with a release decreases significantly, and the release process turns out to be more reliable.

10 http://leankit.com/learn/kanban/continuous-improvement/

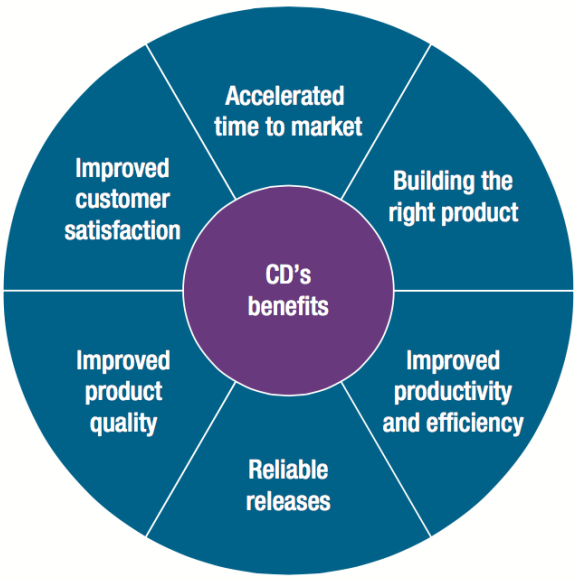


Figure 7.: Continuous Delivery Benefits [Che15]

- Continuous Testing [Vir15]: Technique that represents the process of executing automated tests, with the intent of obtaining fast and continuous feedback. As soon as the software is delivered (Continuous Delivery), the system should be able to execute automated tests to validate and provide feedback. In addition, this technique allows the testing process to begin earlier in development lifecycle, thus minimizing the risks of encountering bugs at later stages.

- Continuous Monitoring [Vir15]: Technique which is a direct consequence of some of the one above described. The capability to test early and on a system similar to production allow to monitor various quality parameters throughout and hence ability to react to any issue in timely manner.

2.3.2.1 *DevOps Adoption*

Adopting DevOps [SC15] is something that requires work on three dimensions: people, process and technology. However, prior to adopting DevOps, it is necessary to identify which are the business objectives and the current bottlenecks in the delivery pipeline. Often, there is unnecessary rework being done, because the defects are being uncovered in production, which forces the development to fix them. Also, it is common to have functionalities that are developed for some reasons, which are not required at the exact time they are produced.

People are the core of DevOps adoption and need to embrace a new culture, which may not be an easy process because it is not as linear as adopting a process or a tool. The DevOps culture is characterised by a high degree of collaboration across roles, focus on business and trust.

It is essential to start identifying bottlenecks in the current implemented processes early on, and break them in order to accelerate delivery. This can be done by working in smaller increments, being one of the agile methodology drivers, which translates in faster feedback loops. When adopting DevOps there are a set of techniques (2.3.2) that need to be taken into consideration.

Technology enables people to focus on high-value creative work while delegating routine tasks to automation. In an initial stage, it is required some additional work and effort but in the long run, these automated tasks can be reused, therefore increasing the efficiency.

2.3.3 *Benefits of DevOps*

The benefits associated with the practise of DevOps are directly linked with the ones associated with the used techniques (2.3.2). Some of the measurable benefits[11] can then be turned into key performance indicators (KPI[12]).

- Shorter Development Cycle: DevOps promotes a culture of increased collaboration and communication between the development and operations teams. This translates into shorter timeframes to move from engineering code into executable production code.

- Increased Release Velocity: The shorter development cycle creates increased frequency for release of code into production. What conventionally used to take months from requirements to release is now reduced to daily or hourly release build cycle. This fosters continuous development and deployment, subsequently increasing the value to the business.

- Improved Defect Detection: DevOps builds on top of the agile methodology and, in a sense, it can be considered as extending agile programming. It prescribes to several agile principles such as collaboration, iterative development, and modular programming, breaking larger codebases into smaller manageable features. This makes it easier to detect code defects.

- Reduced Deployment Failures and Rollbacks: The benefits gained from faster development and deployment can be nullified by failed deployment. But software when developed using a DevOps mind-set takes into account an operational point of view as well. This, when combined with improved defect detection, significantly decreases the number of pre and post deployment issues and therefore fewer rollbacks.

11 http://www.logicworks.net/blog/2014/10/measurable-important-benefits-devops/
12 http://management.about.com/cs/generalmanagement/a/keyperfindic.htm

- Reduced Time-to-Recovery: Even if the likelihood of failures is minimized, failures as a rule are inevitable. When failures do occur, the time to recover to operational efficiency is relatively reduced in a DevOps environment compared to a traditional IT environment. This is primarily due to the efficiencies gained by development team members understanding how operations teams work and vice-versa. Combined with a robust versioning process, rollbacks are made easier.

Looking at the business perspective, those 5 items summarises the benefits. However, there are also some benefits for both the development and operations teams associated with this approach. The fact that there is the intent of automating as much as possible the processes, it assures that human error is minimized as well as allowing the teams to focus on the things that most matter, which is to bring value to the solution, and consequently the client.

2.3.4 *CALMS Framework*

In the world of DevOps, the framework that is commonly advised to adopt, namely by one of DevOps pioneer, Jez Humble, is the CALMS Framework. It is acknowledged that there are five underlying principles that fully represents the framework:

C ulture adaption.

A utomate what you can.

L ean, simple and reliable processes.

M easuring everything.

S haring what matters.

2.3.4.1 *Culture*

Culture is not something that can be implemented, it is the natural environment that encompasses the entire development team, including the developers and operations. In this eco-system of people and tools, the culture is a key factor to remove barriers and facilitate collaboration to accomplish both teams and company objectives.

2.3.4.2 *Automation*

Once the new mind-set is established, it is all about releasing new applications or features at speed while maintaining quality. In order to do this, automation solutions are extremely important. Finding a way to lead all new initiatives with the help automation processes is an assurance that they will be able to easily and constantly scale.

2.3.4.3 *Lean*

As automation is built, it is essential to maintain every process and tool lean, simple and reliable. Lean approaches are all about providing increased value to customers or employees, while minimising waste [HMO10] [RTT12]. To accomplish this, lean thinking changes the focus of management from optimising separate technologies and assets, while eliminating waste along entire value streams, allowing companies to respond to changing customer desires with higher quality, lower costs, and with very fast throughput times.

2.3.4.4 *Measuring*

In a lean organisation, automating everything can sometimes imply that some of the ability to remember and document the work is lost, because if an individual team do not have visibility of the big picture eventually something will go wrong. Measurement is the glue that keeps it all together and, if implemented correctly, it will be possible to analyse and correlate all processes, tools and data together.

2.3.4.5 *Sharing*

Collaboration between developers and operations is core to a DevOps culture. Being able to share what works, what is causing problems and specifics of overall performance is key. Without sharing, it useless to care about having the right culture, automation adoption, embracing lean and having measurements that work. Sharing is not just reporting facts, it is regular exchanging ideas across teams and helping each other out.

2.4 THE DEVOPS TOOLS

The DevOps Lifecycle is composed of five phases, as seen in Figure 8, which have specific tools associated with them and will be detailed individually in the following subchapters. They are as follows: Requirements, Development, Testing, Deployment and Monitoring.

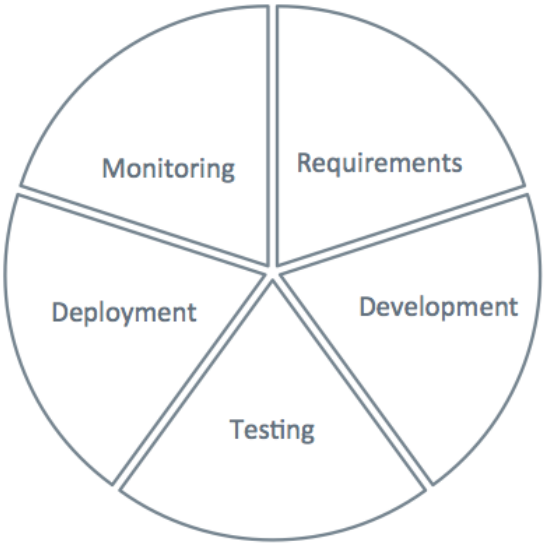


Figure 8.: DevOps Lifecycle

2.4.1 *Requirements*

Implementing a DevOps approach requires fast iterations, quick feedback cycles, small number of requirements and incremental improvements in real time, so it should be followed a Scrum[13] methodology aided by project tracking tools and also test designing tools.

The benefits that are inherited by using these kind of tools are the following:

- Transparency: Business and IT have full visibility over requirements.
- Easier Prioritization: Easier to detect dependencies and prioritize based on business needs.
- Less Complexity: Requirements are broken down into smaller components, easier to manage and deliver.
- Traceability: Clear link between requirements and functionalities delivered.
- User Friendly: User Stories specifications easier to understand by the business and IT.

Table 3 contains a set of application that are inserted in the Requirements phase, which are used for project tracking, test design and documentation.

13 https://www.scrumalliance.org/why-scrum

Table 3.: Requirements Applications

Project Tracking	Test Design	Documentation
Jira	Zephyr	Confluence
Rally Software	HP Quality Center	Sharepoint
Asana	Enterprise Tester	

2.4.2 *Development*

During this phase, it is demanding that the code is delivered more often, faster and with better quality. In order to accomplish this, there is the need of source management tools, version control system tools and code review tools.

The benefits that are inherited by using these kind of tools are the following:

- Minimize Human Error: Through automatic code review over the development, human error is minimized.
- Easier Environment Management: Through automation is easier to control how many environments are present and how they are configured.
- Infrastructure as Code: Environment setup automated allows an easy configuration.
- Continuous Delivery: Quick merge and integration of new developments on the main branch.

Table 4 contains a set of application that are inserted in the Development phase, which are used for source code management, version control, code review and repository management.

Table 4.: Development Applications

Source Code Management	Version Control System	Code Review	Repository Manager
GitLab	Git	Phabricator	Artifactory
GitHub	Subversion	Review Board	NuGet
Bitbucket	Team Foundation Server	Codacy	

2.4.3 *Testing*

During the testing phase, the aim is to have an automated process, reducing human error and optimising the delivery. Every time a set of requirements is ready to evolve, the system executes code analysis and a set of tests to guarantee the consistency and performance of the overall system. If they pass then the process of continuous deployment takes over and evolves it to the next stage.

The benefits that are inherited by using these kind of tools are the following:

- Automatic code review: Automated processes occur every day highlighting what needs to be improved in the code, and blocking commits of breaking code.
- Automatic tests: Automatic execution of test provide the tools to reach an optimal scenario where all lines of code are tested as well as all valid testing scenarios.
- Minimized human errors: Automation provides a scenario where errors can be highlighted and minimized.

Table 5 contains a set of application that are inserted in the Testing phase, which are used for functional testing, code analysis, unit testing and performance testing.

Table 5.: Testing Applications

Functional Testing	Code Analysis	Unit Testing Automation	Performance Testing
Selenium	Sonarqube	HP Unified Functional Testing	HP LoadRunner
HP Unified Functional Testing	JArchitect	Catch	WebLOAD
	Codacy	TestNG	The Grinder

2.4.4 *Deployment*

The aim of this phase it to have a continuous improvement process. A process where environments are configured without human intervention and code is deployed automatically. For that purpose, it is required tools for configuration management and continuous integration.

The benefits that are inherited by using these kind of tools are the following:

- Continuous delivery: Quick merge and integration of new developments on the main branch.

- Infrastructure as code: Environment setup automated allows an easy configuration.
- Easier environment management: Through automation is easier to control how many environments are present and how they are configured.

Table 6 contains a set of application that are inserted in the Deployment phase, which are used for configuration management and continuous integration.

Table 6.: Deployment Applications

Configuration Management	Continuous Integration
Puppet	Jenkins
Chef	Bamboo
Ansible	
SaltStack	

2.4.5 *Monitoring*

The aim of this phase is to continuously monitor infrastructure and solution in order to create an easy and clean process of status reporting. Is order to accomplish such thing, tools for continuous monitoring and logging are required.

The benefits that are inherited by using these kind of tools are the following:

- Easy and fast reporting: Quick tools in order to check status over environments and solutions.
- Early warning for preventing issues: Early warnings if thresholds are crossed or reached. Allowing prevention actions to avoid issues.

Table 7 contains a set of application that are inserted in the Monitoring phase, which are used for continuous monitoring and logging.

Table 7.: Monitoring Applications

Continuous Monitoring	Logging
Dynatrace	Logstash
New Relic	
Splunk	

3

THE PROBLEM AND ITS CHALLENGES

Creating new processes and adopting new behaviours in an effort to meet the business needs to delivering quicker and better software, can sometimes be the source of conflicts among the IT department. On the other hand, these challenges can be addressed and conflicts reduced through embracing DevOps practices. The following are the problems that DevOps philosophy attempts to solve:

- It is a business and cultural problem, since it stresses the collaboration between development and operations, while aiming to deliver software and services quicker;
- Delivering software and services quicker raises up some technology challenges, mainly since there are a new set of tools and processes which needs to be adopted and mastered.

3.1 COLLABORATION BETWEEN DEVELOPMENT AND OPERATIONS

One of the current biggest problem in organisations is that the operations teams, or the teams responsible for making infrastructure available to development, do not fully understand the needs of the business and may cause delays in the development cycle. In this continuous delivery fashion, business requirements are being introduced at a very rapid pace and, therefore, there is a constant demand for immediate physical and virtual servers. This is a critical part of the delivery process, because if there are any delays, it will have direct impact on task planning.

To minimize this problem, culturally something has to happen to guarantee that everyone knows that both operations and development are part of the same team. There has to be more communication between both teams and try to involve operations team in the earlier phases of the development cycle so they can have visibility of what is coming and anticipate any potential issue. In addition, aiming to deliver new infrastructure by automating it and creating an on demand deployment capability.

Another common problem among the two teams is the lack of cooperation and collaboration between them. DevOps philosophy is very clear in this point, both teams should

work collaboratively and share everything, ideas, goals, workloads and responsibility, with the final goal of delivering high quality software. The biggest challenge is to try and bring the goals of both teams to coexist. While the development team wants to change features, by adding new ones or modifying already existent, operations wants to keep their services running without any disruption. Their goal is system stability, resiliency and availability. Due to multiple and continuous changes, it may be the case that the delivered software is not well written or might contain bugs which will, ultimately, introduce problems on the operations side, and therefore, there is always some resistance from the operations team to any change. Last but not least, the working method of a standard development team is often agile while, on the other hand, the operations team follow more of a static process, a recipe with a set of procedures. Table 8 sums up what are the differences of both teams.

Table 8.: Developer vs Operations

Development Team	Operations Team
Change	Stability
Add/modify features	Create/enhance services
Deploy inconsistent software	Motivation to resist change
Agile process	Static process

This means that there is big gap between the development and the operations team which can, in the worst case scenario, lead to conflict and inefficiencies. In order to promote better communication between both teams, it is highly advisable that they should be working side by side every single day, also increasing the awareness of what each team is doing and the impacts that the developments might have in the infrastructure and vice versa.

3.2 Adopting a new set of tools and processes

One of the DevOps techniques, *Continuous Delivery*, proposes to rectify the gap that existed in the Agile methodology between development and operations teams, whereby, code written and committed to a version control system, will be analysed, tested, built, deployed and installed on the production environment, ready to be consumed by the end-user. Every action apart from the code writing and versioning can be automated, allowing all stakeholders to concentrate their efforts on designing and coding high quality software, rather than worrying with the manual processes of testing, building and deploying. Based on the DevOps lifecycle phases, Figure 9 illustrates the *Continuous Delivery* process flow. Although this flow represents the lifecycle of each requirement, which follows a waterfall approach, the fact is that two requirements can be at two different stages of the process, meaning that

there are scenarios where there are new requirements being released into production when, at the same time, others are being developed or tested.

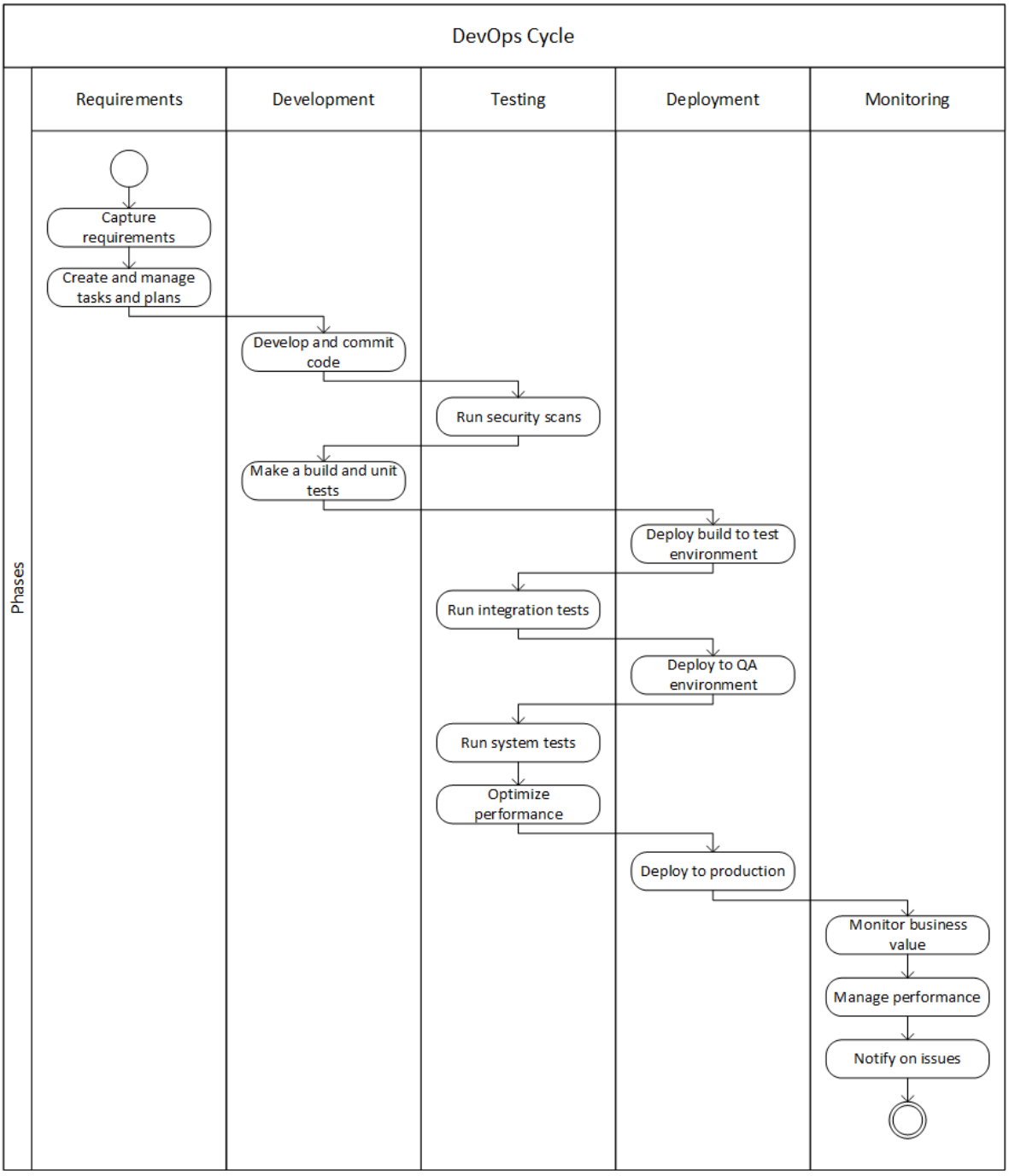


Figure 9.: Continuous Delivery Process Flow[1]

The biggest challenge lies in the fact that there is the need to master various new tools and processes. However, DevOps encourages everyone to contribute across the chain, meaning

1 Adapted from https://www.ibm.com

that a developer can configure deployments or a deployment engineer can add new test cases to the QA repository.
As expected, every organisation must look at this change as an investment. There is an initial phase of analysing the available tools in the market, deciding the DevOps suite, integrating all of the applications and designing and implementing the automated process flows. After this phase, the organisation will begin to take advantage of the benefits that DevOps practises bring, detailed in section 2.3.3.

3.2.1 *Automated Flows*

In this section, it will be illustrated several automated process flows in an abstract way, without going into many detail or selecting any specific tool. The processes that will be illustrated with a sequence diagram are *Code Analysis*, *Automated Tests*, *Code Review*, *Deployment* and *Monitoring*.

3.2.1.1 *Code Analysis*

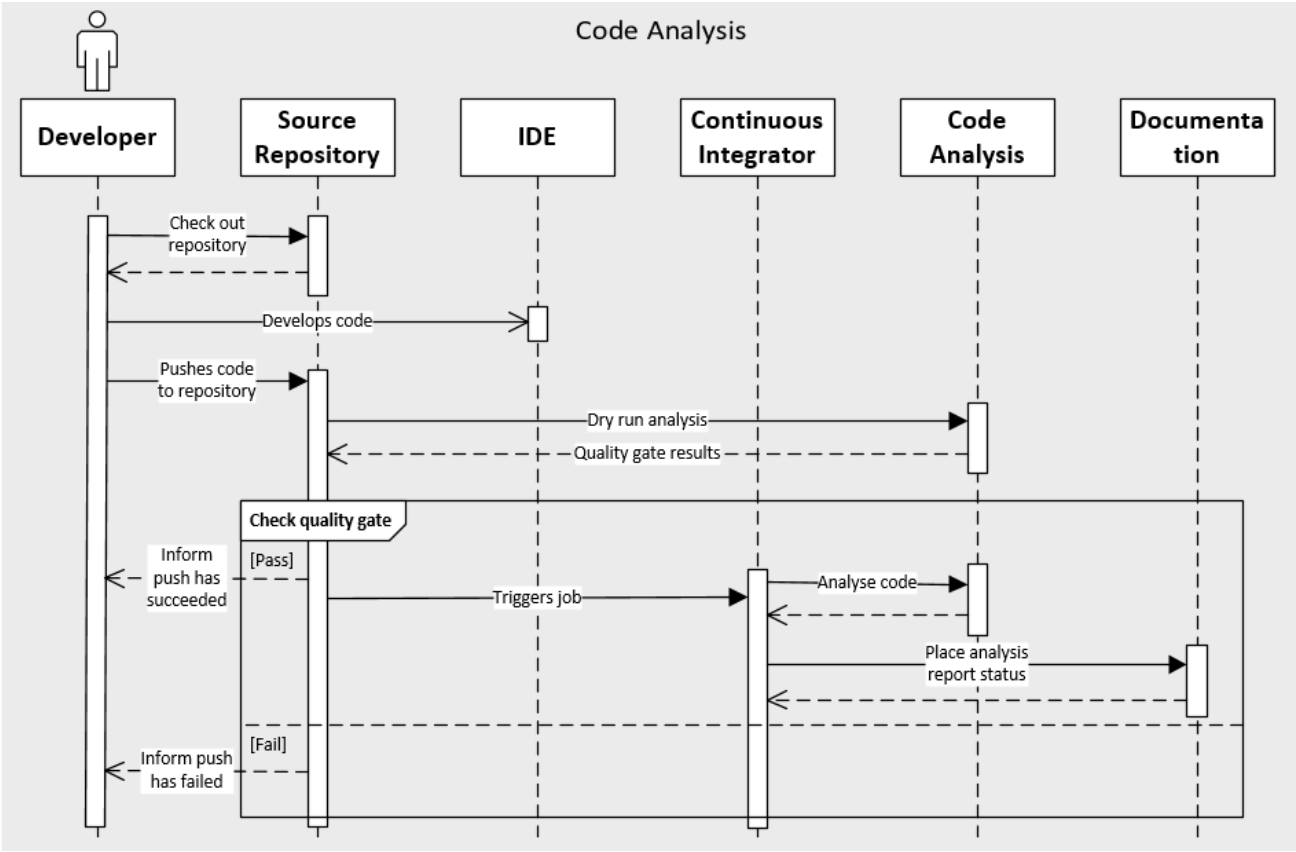


Figure 10.: Code Analysis Flow

Code analysis is used by developers in their IDEs to catch problems while they are coding, meaning that the code must follow a certain set of rules and pass the defined quality gate. Illustrated in Figure 10, fast incremental code analysis checking can be included in *Continuous Integration* to catch problems immediately after code has been submitted to the source repository, and deeper analysis checks can also be done to ensure that code will not be deployed with any critical issues or security vulnerabilities.

A successful code analysis should run quickly and provide an easy way to alert developers of any potential issue. The information should be clear and unambiguous, and indicate where, why and how to fix the issue.

3.2.1.2 *Automated Tests*

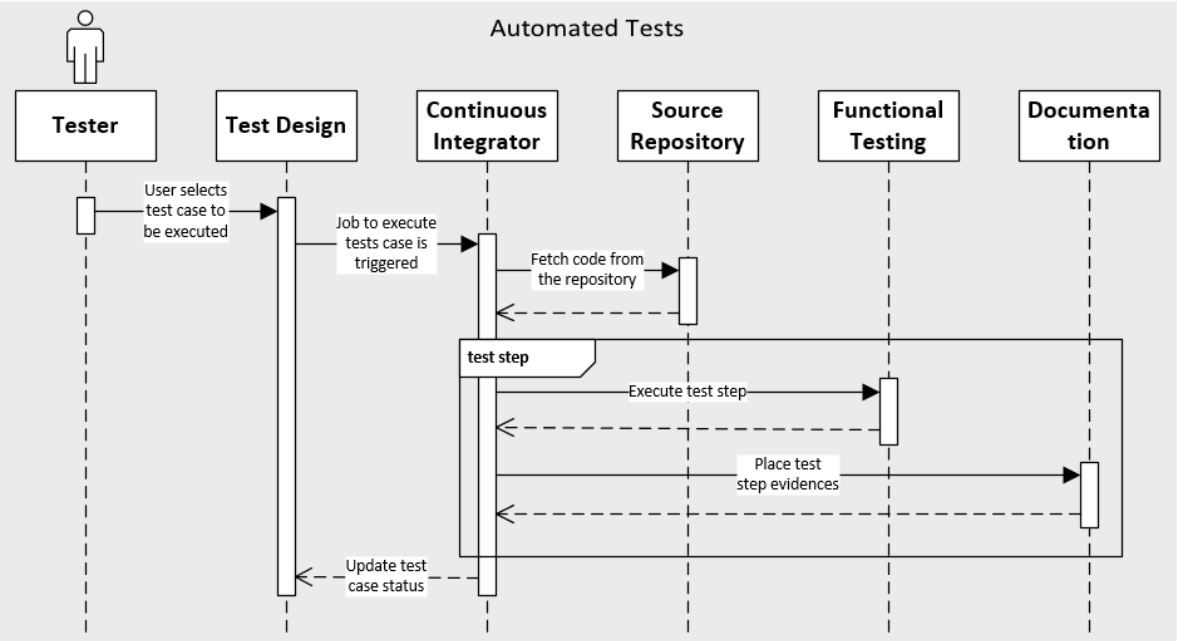


Figure 11.: Automated Tests Flow

Tests play a very important role in any organisation and it is where most of the time is being spent. Additionally, in the DevOps philosophy where deployments are made continuously, it is crucial to automate all tests as much as possible and reusing whatever can be reused. Manual testing uses up precious resources and introduces other variable such as the human error.

Illustrated in Figure 11, this process allows less time spent worrying about covering all tests scenarios, as well as cutting the costs of having dedicated testers for the job, quickly verifying that any code change still passes the defined tests and lastly reducing drastically the possibility of human error.

3.2.1.3 *Code Review*

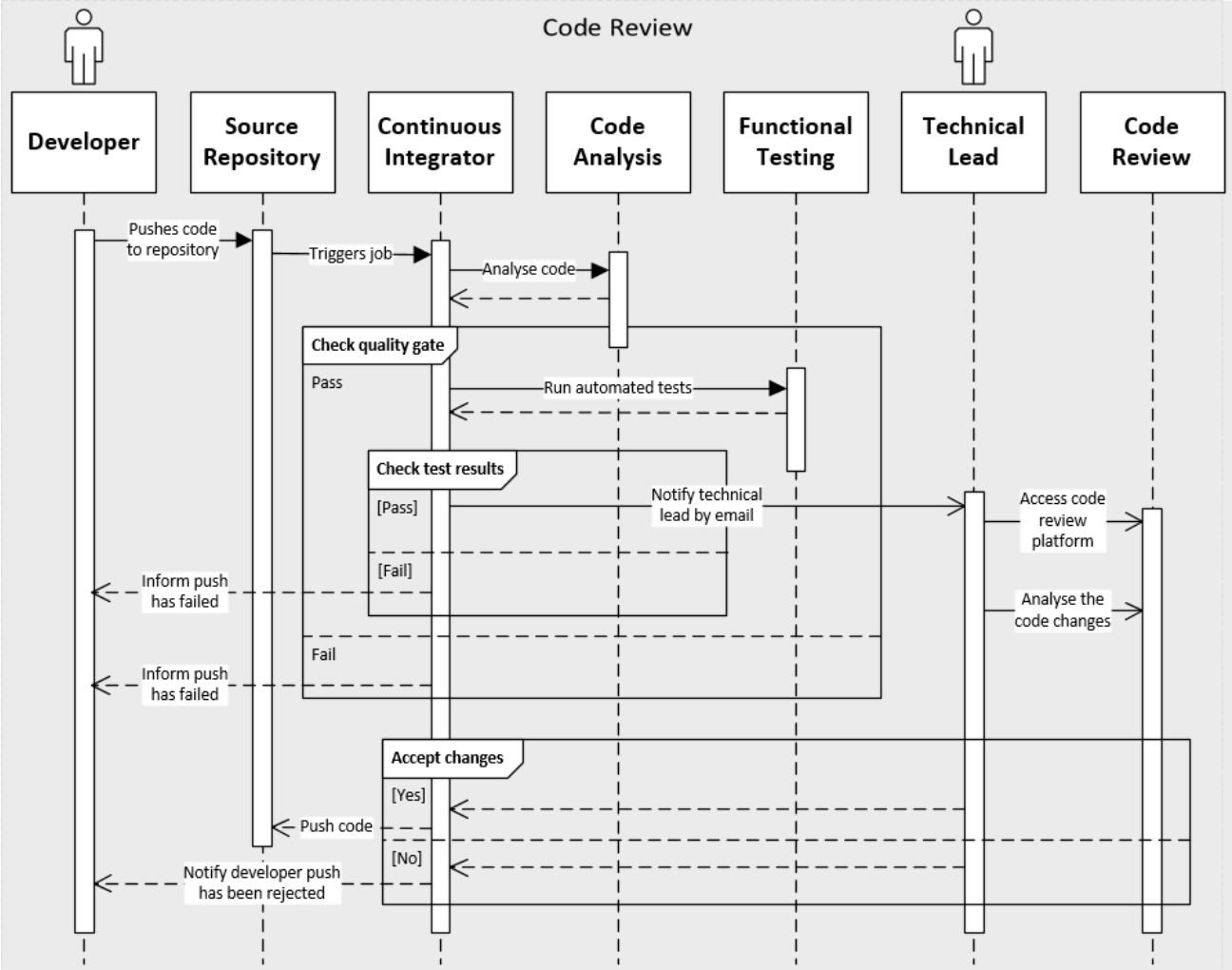


Figure 12.: Code Review Flow

Illustrated in figure 12, this process should be in every organisation even if it is not automated as it in a DevOps philosophy. Peer code reviewers generally focus on making sure that the code is understandable, maintainable and that it follows all of the organisation's code conventions. Furthermore, they can also suggest to developers ways to improve the code efficiency as well as how to write it using the best practices.

Code review process is a good way to improve security or reliability, because it usually happens early in development. Therefore, the risk of finding a bug or a security breach at a later stage is very remote. Due to *Continuous Delivery*, most code changes are small and incremental, making the code review process quicker, especially if the code being reviewed has already passed through a static analysis to filter the most obvious mistakes.

3.2.1.4 *Deployment*

Illustrated in fugure 13, the deployment process is a central part of *Continuous Delivery*, since one of the main goals is to be able, at any point of time, to deploy a software package or a simple configuration file to any of the environments. Due to this need, it is required a fast process to create the build packages ready for deployment in any environment. In addition, and only when applicable, the software must pass the quality gate and the automated tests.

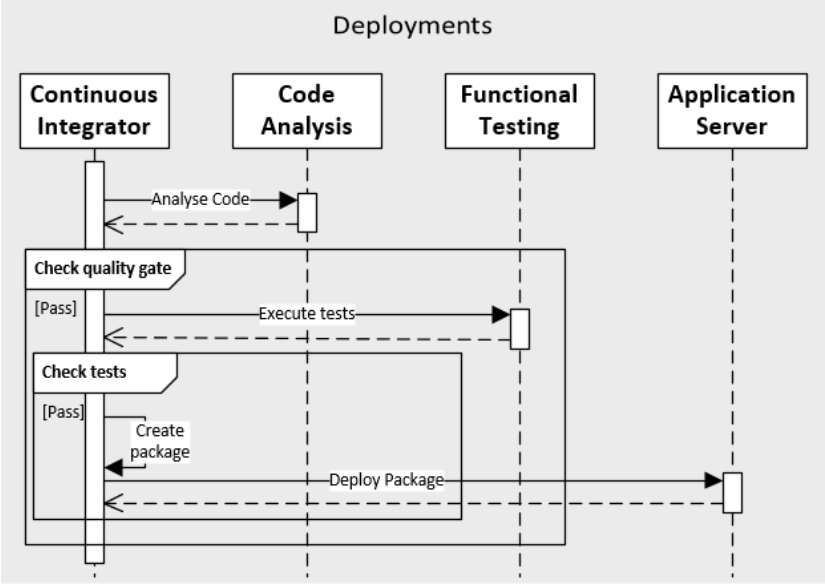


Figure 13.: Deployment Flow

3.2.1.5 *Monitoring*

In a DevOps world nothing is more ubiquitous than the monitoring process, as it adds value to increase awareness during each phase of the delivery pipeline. As it is spread across multiples phases, there is no unique process, but rather a set of primary categories to focus on. Some of these categories are performance, capacity, throughput, SLAs, user metrics and log file analysis, which are detailed below in Table 3.2.1.5.

Table 9.: Monitoring Categories[2]

Category	Description
Performance	Analyse page loads, query times, response times and upload/download speeds.
Capacity	Analyse disk space, memory, CPU and bandwidth.
Throughput	Analyse the throughput of the web, cache, database, network and application layers.

Table 9.: Monitoring Categories[2]

Category	Description
SLAs	Analyse availability, reliability and security.
User Metrics	Analyse registrations, page views, bounce rates and click rates.
Log File Analysis	Predictive analytics and pattern recognition.

2 http://devops.com/2014/04/15/nagios-monitoring-strategy/

4

PROCESS AUTOMATION DEVELOPMENT

In this chapter, it will be discussed the main decisions that were taken along the way, all of the scripts, jobs and artefacts that were created and the outcome of the flows that were automated.

Regarding the suite of tools that will be used to simulate the end to end DevOps flow, Table 10 summarises it.

Table 10.: DevOps Tools Suite

DevOps Lifecycle	Tool	Selected Product
Requirements	Project Tracking	Jira
Requirements	Test Design	Zephyr
Requirements	Documentation	Confluence
Development	Source Management	Bitbucket
Development	Code Review	Phabricator
Development	Repository Manager	Artifactory
Testing	Code Analysis	SonarQube
Testing	Functional Testing	Selenium
Testing	Performance Testing	JMeter
Deployment	Configuration Management	Puppet
Deployment	Continuous Integration	Jenkins
Monitoring	Monitoring	Dynatrace

The decisions on which tool to pick for each of the phases of the DevOps Lifecycle were based on three main criteria:

1. Functionalities of each tool;
2. How easy it would be to integrate with the rest of the tools of the suite;

3. The cost associated with each tool.

During the core of the dissertation, some of these tools will be used in the implementation of the automated flows. The flows that will be implemented will be the automated code analysis process, the automated functional tests process and an automated deployment process to an application server. Finally, this will demonstrate a portion of the tools suite in action, on a real industry use case.

4.1 DECISIONS

The following subsections detail the functionalities of each individual tool that was selected for the DevOps suite. The table below is used to measure the necessity of each documented feature. The contents from Figure 14 to Figure 24 were obtained from an analysis conducted by an internal Deloitte project.

Table 11.: Tool Analysis

Symbol	Description
●	Feature is essential
●	Would be "nice to have" this feature
●	Feature is available
●	Feature is missing
●	Feature is partially available

4.1.1 *Project Tracking*

For project tracking, the selected tool was Jira. Jira is a proprietary issue tracking product, developed by Atlassian. It provides bug tracking, issue tracking, project management functions among other features. Figures 14 and 15 show Jira functionalities in more detail.

Functionality		Description	Status	Jira	Jira API access
Project Management	Plan Sprints	Choose which issues are to be included in the next sprint	●	●	●
	Use Scrum	A Scrum board is a board that was created using the "Scrum" preset. Scrum boards are for teams that plan their work in sprints	●	●	N. A.
	Use Kanban	Kanban boards are for teams that focus on managing and constraining their work-in-progress. Because work is not planned in advance using discrete time periods or versions, Kanban boards do not have a Backlog screen	●	●	N. A.
Notifications	Email notifications	Define personal and project level email notification schemes	●	●	●
	Email subscriptions	Subscribe to custom search filters to receive regular updates of changes	●	●	●
Resource Management		Allows product owners to plan capacities and manage allocations for projects, track resource consumption	●	●	●
Budgeting		Take control of project costs, revenues and resources - Plan, track, and forecast finances and capacity in real-time in JIRA	●	●	●
Issue Tracking	Create user stories & issues	A story is represented as an issue, and individual tasks within the story are represented as sub-tasks	●	●	●
	Watch issues	Add yourself or others as a watcher to any issue to receive email updates whenever that issue changes	●	●	●
	Custom Issue Types	Add and remove fields to define custom issue types to match development	●	●	●
	Bulk Issue Modifications	Bulk Operations enable operations to be performed on multiple issues at once	●	●	●
	Linking to Other Issues	Issue linking allows you to create an association between two or more existing issues	●	●	●
	Issue Level Time-Tracking	Specify work estimates for each issue and log work as you go. Roll up time tracking data across sub-tasks and maintains the time spent and remaining estimate for tracking and reporting	●	●	●
	Issue Level Security	Restrict access to individual issues within a project	●	●	●

Figure 14.: Project Tracking - part 1

Functionality		Description	Status	Jira	Jira API access
Installation & Configuration	Suite Integration	Able to connect from/to multiple applications from a suite	●	●	N. A.
	Remote APIs	Provides a full set of remote APIs (e.g. REST, SOAP, XML-RPC, etc.)	●	●	●
	On Premises	Can be installed locally	●	●	N. A.
	Standalone application	Has a pre-packaged database and application server	●	●	N. A.
	Connect LDAP & Active Directory	Manage users and groups from any number of directories using the User Directories manager	●	●	N. A.
	Bundled Database Drivers	Includes database drivers to connect to database	●	●	N. A.
	Plugin Ecosystem	Plugins to extend application potential	●	●	N. A.
Task Management	Dashboards	Allows you to neatly organize related information by context	●	●	●
	Custom Workflow	A workflow is the set of statuses and transitions that an issue goes through during its lifecycle. Workflows typically represent business processes	●	●	●
	Attachments	Users will be allowed to attach files and screenshots to JIRA issues	●	●	●
	Moving & Copying Issues	Easily move an issue from one project to another by using the Move Issue wizard	●	●	●
	Sub-tasks	Sub-task issues are useful for splitting up a parent issue into a number of smaller tasks that can be assigned and tracked separately	●	●	●
Reporting	Custom charts created from filters	Create a chart gadget from any search result or saved search filter and add it directly to the dashboard	●	●	●
	Release Notes	Generate release notes containing details of all issues fixed in a specific version	●	●	●
	Road Map	The project Road Map shows the issues scheduled for upcoming unreleased versions of a project	●	●	●
	Change Log	The project Change Log shows the resolved issues for the recently released versions of a project	●	●	●
	Information Radiator	An information radiator is a large, highly visible display used by software development teams to track progress	●	●	●

Figure 15.: Project Tracking - part2

4.1.2 *Test Design*

For test design, the selected tool was Zephyr. Zephyr for Jira is a native application that resides in Jira and bring test management capabilities to any Jira project. With this tool, tests can be created and viewed in any Jira project, linked to other issues, executed immediately or as part of a test cycle. ZAPI exposes Zephyr for Jira via REST APIs, which allows to build custom integration with Zephyr for Jira. Figure 16 show Zephyr functionalities in more detail.

Functionality		Description	Status	Zephyr	ZAPI
Specification		Testing is integrated into the project cycle now, allowing users to track software quality and make empowered go/no-go decisions	●	●	●
Resource allocation		Plan and assign test execution cycles	●	●	●
Test Preparation	Create and modify tests	Create and modify tests with test steps and attachments. Create, view, edit and clone richly formatted tests	●	●	●
	Build test plans	Build test execution cycles	●	●	●
Defect Management		File, link and track defects	●	●	●
Dashboard and Reporting		Configure and track quality metrics, share dashboards	●	●	●
Testing Specification	Manual Testing	Be able to manually validate that a test has passed / failed execution	●	●	●
	Automated Testing	Be able to start tests on automated testing tools	●	●	●

Figure 16.: Test Design

4.1.3 *Documentation*

For documentation, the selected tool was Confluence. Confluence is a team collaboration software, developed by Atlassian. In its essence, Confluence can be seen as a wiki, making it very easy to create and edit content. Figure 17 show Confluence functionalities in more detail.

Functionality	Description	Status	Confluence	Confluence API
Page editor	Create meeting notes, project plans, product requirements, etc.	●	●	●
Inline comments	Leave the feedback on the work itself on any page	●	●	●
File collaboration	Feedback directly on existing files	●	●	●
Version tracking	Keep track of versions automatically (Pages and files)	●	●	N. A.
Organized workspaces	Create a space for every team, department, or major project to share information and keep work organized.	●	●	●
Blogs and discussions	Share news and announcements, and have active discussions with comments, @mentions and likes.	●	●	●
User and content permissions	Keep the site and content secure with granular permissions	●	●	●
JIRA integration	Connect to JIRA to provide insight into the development work with automatic linking, quick issue creation and reports in Confluence.	●	●	N. A.
Previews	View many file types right from the page with a file previewer	●	●	N. A.
Dedicated task view	Automatically link requirements and retrospectives to issues, epics and sprints in JIRA.	●	●	N. A.

Figure 17.: Documentation

4.1.4 *Source Management*

For source management, the selected tool was Bitbucket. Bitbucket is a web-based hosting service, owned by Atlassian, for projects that use either the Mercurial or Git revision control systems. Figure 18 show Bitbucket functionalities in more detail.

Functionality		Description	Status	Bitbucket	Bitbucket API
Hosting	Cloud-based	No installation required	●	●	●
	On premises	Can be installed locally	●	●	●
Accessibility	Shell	Allows users to add their own SSH keys to the server and use those SSH keys to secure Git operations between their computer and the server instance	●	●	N. A.
	API	Use the REST plugin module to expose services and data entities as REST APIs	●	●	●
Team Management		Granular user management options ensure only the right people have the right access to the code, all the way down to the branch level.	●	●	●
Code review		Integrated code review tool	●	●	●
Bug tracking		Have issue tracking/bug reporting on the source management tool	●	●	●

Figure 18.: Source Management

4.1.5 *Code Review*

For code review, the selected tool was Phabricator. Phabricator is a suite of web-based software development collaboration tools, including the Differential code review tool and the Herald change monitoring tool. Figure 19 show Phabricator functionalities in more detail.

Functionality	Description	Status	Phabricator	Phabricator API
Automatic code review on commit to branch	Have code review tool track changes in GIT repositories and automatically trigger actions	●	●	N. A.
Peer review trigger after branch commit	Specify a group of people for review and reviews will show up in the dashboard of individuals in that group.	●	●	N. A.
Rule based code reviews	Build Custom Rules for Pre-Commit Code Reviews	●	●	●
Integrated Issue tracking	Have issues info integrated with code review	●	●	●
Access Control	Granular user management options ensure only the right people have the right access to your code, all the way down to the branch level.	●	●	●
Integration with Continuous integration tools	Start CI builds and allow CI tool to report build status and coverage information	●	●	●
Code review result notifications	Notify users of result notifications	●	●	N. A.

Figure 19.: Code Review

4.1.6 *Code Analysis*

For code analysis, the selected tool was SonarQube. SonarQube is an open source quality management platform, dedicated to continuously analyse and measure source code quality. Figure 20 show SonarQube functionalities in more detail.

Functionality	Description	Status	SonarQube	SonarQube API
Static code analysis	Analyze typical developer errors: Duplications, Coding standards, Lack of coverage, Potential bugs, Complexity, Documentation and Design	●	●	●
Integrate with continuous integration tools	Fully automate the process. Let users easily put in place a process of Continuous Inspection.	●	●	●
Integrate with version control system	Automatically detect changes in code and trigger code analysis	●	●	●
Connect LDAP & Active Directory	Manage users and groups from any number of directories using the User Directories manager.	●	●	●
Pre-commit check	Check source code before pushing changes to the SCM	●	●	●
Drilling down analysis	Quickly go from a quality indicator to a corrective action in the source code	●	●	●
Metrics evolution	Ensure that progress gets tracked over time	●	●	●

Figure 20.: Code Analysis

4.1.7 *Functional Testing*

For functional testing, the selected tool was Selenium. Selenium is a tool used to create robust, browser-based regression automation suites and tests. JUnit is the testing framework that is used, designed to cover all category of tests. Figure 21 show Selenium functionalities in more detail.

Functionality	Description	Status	Selenium
Integration with continuous integration tools	Create test cases and publishes test results in design tools	●	●
Support for mobile	Connect to VCS, checkout code and build artifacts	●	●
Test report generation	Start builds from IDEs	●	●
Manual testing	Able to store the built artifacts locally	●	●
Automated testing	Connect jobs in a dependency chain	●	●
Recoding functionality	Use SSH to run shell commands on a remote machine	●	●
Playback functionality	Allows users to quickly playback recorded tests	●	●
Object Identification	Able to identify the objects such as Links, Buttons, Edit boxes, Drop downs	●	●
Trigger test executions	Have a way to automatically trigger test executions	●	●

Figure 21.: Functional Testing

4.1.8 *Continuous Integration*

For continuous integration, the selected tool was Jenkins. Jenkins is a continuous integration server that can accelerate the software development process through automation. It manages and controls development lifecycle processes such as build, static analysis, tests, packaging and deployment. Figure 22 show Jenkins functionalities in more detail.

Functionality	Description	Status	Jenkins	Jenkins API
Integration with test design tools	Create test cases and publishes test results in design tools	●	●	●
Version control system integration	Connect to VCS, checkout code and build artifacts	●	●	●
IDE integration	Start builds from IDEs	●	●	N.A.
Job artifact persistence	Able to store the built artifacts locally	●	●	●
Pipeline definition	Connect jobs in a dependency chain	●	●	●
SSH support	Use SSH to run shell commands on a remote machine	●	●	N.A.
API / Rest support	Provide machine-consumable remote access API to its functionalities	●	●	●
Deployment capabilities	Deploy successfully built artifacts to external storages	●	●	●
Notifications	Notify users of build results	●	●	●

Figure 22.: Continuous Integration

4.1.9 *Repository Manager*

For repository manager, the selected tool was Artifactory. Artifactory integrates with all major continuous integration/deployment and DevOps tools, providing an end-to-end, automated solution for tracking artefacts from development to production staged. Figure 23 show Artifactory functionalities in more detail.

Functionality	Description	Status	Artifactory	Artifactory API
Local cache	Remote artifacts are cached locally for reuse, so that you don't have to download them over and over again.	●	●	●
Access control	Advanced security features give you control over who can access your artifacts, and where they can deploy them.	●	●	●
Automation	Automate aspects of artifact management using a REST API.	●	●	●
On-Premise installation	Can be installed locally	●	●	●
Fine-grained build access	Provides information on all builds	●	●	●
Manage builds	Manage builds (e.g. Upload builds, delete builds)	●	●	●
Retrieve Artifacts	Retrieves artifacts from the server	●	●	●

Figure 23.: Repository Manager

4.1.10 *Monitoring*

For monitoring, the selected tool was Dynatrace. Dynatrace manages the availability and performance of software applications and the impact on user experience. Figure 24 show Dynatrace functionalities in more detail.

Functionality	Description	Status	Dynatrace
Infrastructure Monitoring	Be able to monitor servers, network elements, applications, system metrics, distributed monitoring	●	●
Alerting	Email, mobile phone, per-user notifications	●	●
Reporting	Performance Graphs	●	●
	SLA Reports	●	●
	Scheduled Reporting, Executive Summary Report, Bandwidth Report	●	●
	Custom Report Creation	●	●
Instant Remote Host Access	Provides users with instant remote access to any device they are monitoring (e.g. via RDP, VNC, Telnet, or SSH)	●	●
Heartbeat Monitoring	Provides essential monitoring of critical hosts and applications on remote machines.	●	●
User Interface	Advanced Dashboards	●	●
	Sharable/Deployable Dashboards	●	●
Configuration	Configuration Wizards, Auto-Discovery & Auto-Decommissioning,	●	●
Send & Receive SNMP Traps	SNMP traps enable agents to notify the management station of significant events using an unsolicited SNMP message	●	●
Maintenance Tools	Automated Back-Up Scheduler, Upgrade Via Web Interface, External map integration (e.g Google Maps)	●	●
Network Maps	Basic Map, Network Replay	●	●

Figure 24.: Monitoring

4.2 IMPLEMENTATION

4.2.1 *Code Analysis*

During this code analysis flow, the following applications from the DevOps suite will be used: Jira, Bitbucket, SonarQube, Jenkins and Confluence. The flow illustrated in Figure 25 is divided into 4 main steps which will be detailed in the sub chapters below:

- Developer associates a Bitbucket branch to his task in JIRA;
- Developer codes a certain feature and performs any number of commits;
- Developer tries to push his code to the repository and a Bitbucket hook is triggered. The hook performs a dry run to SonarQube to analyse the pushed code. In case of success it calls a Jenkins job to continue the automated flow;
- Jenkins calls SonarQube to generate the analysis report and publishes the report in a Confluence page.

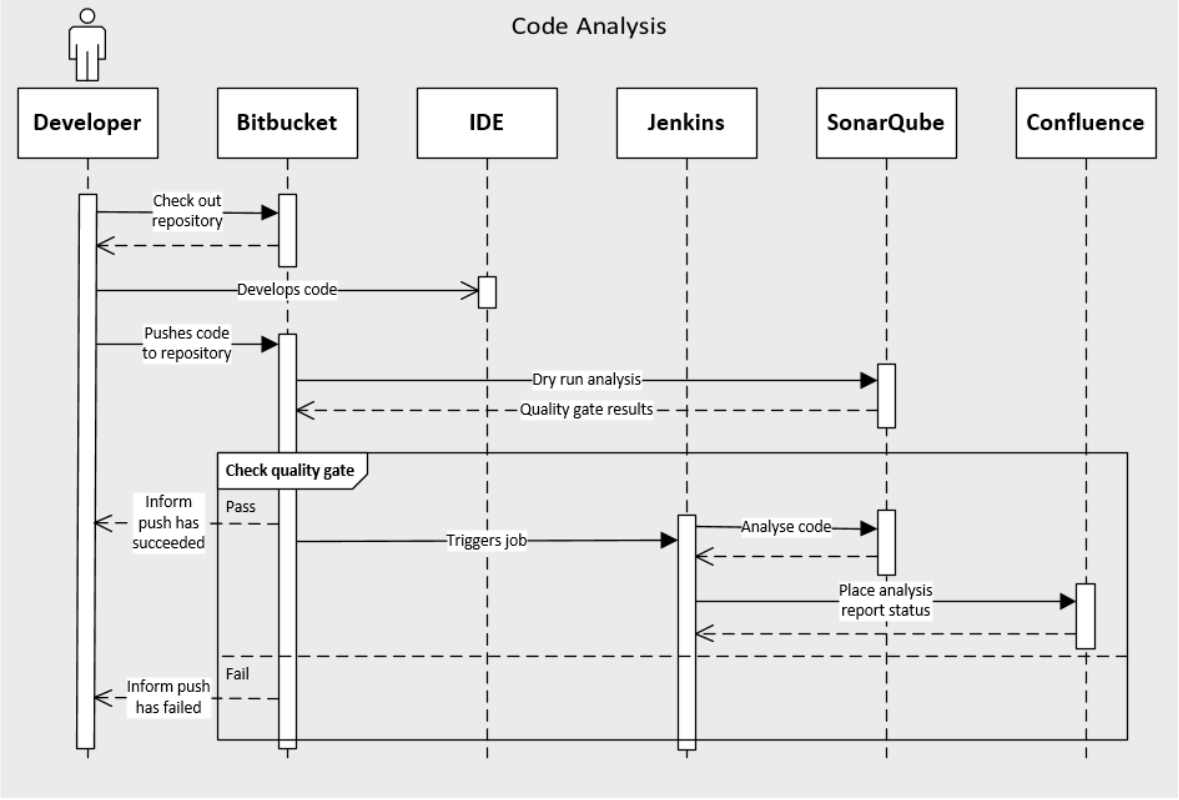


Figure 25.: Code Analysis Flow

4.2.1.1 *Associate Bitbucket branch to a task in JIRA*

In this JIRA installation, it is configured to send emails to the issue assignee every time it is created or modified. Therefore, after receiving an email from JIRA informing that a new issue has been opened, the developer logs into JIRA and associates a new Bitbucket branch to that issue. All of the necessary developments to solve the issue will be committed to that branch, and afterwards merged to the master branch.

4.2.1.2 *Develops code and commits to branch*

After associating the branch, the developer clones that branch to his local machine using the script *script_to_clone.sh*. This script has the following structure:

```
bash script_to_clone.sh <username> <password> <hostname> <port> <project> <
    repository>
```

Table 12.: Script to clone

Input Parameter	Description
username	User name to authenticate in Bitbucket
password	Password for the username provided
hostname	Hostname or IP address of the machine where Bitbucked is installed
port	Port number where Bitbucket is listening
project	Name of the Bitbucket project
repository	Name of the repository being cloned

Following that, the developer creates a new Java project in the IDE in order to start developing and committing his code. In order to guarantee that every developer follows the same commit structure, it should be the company's best practise to implement a client-side hook for that.

Hooks can be defined as a set of scripts that GIT executes when certain events occur in a repository. They can reside in either local or server-side repositories, and they are only executed in response to actions in that repository.

Local Hooks affect only the repository in which they reside. This means that if a change is performed in a Hook in the local repository, it will not affect other team members that also works with the repository. For this reason, it is very important to maintain Hooks configuration aligned between all the team members. Hooks can be used as a way to implement a certain policies to be followed by all team members. The most known local hooks types are:

- **pre-commit:** This Hook is executed every time a commit event happens, and before GIT asks the developer to insert a commit message;
- **prepare-commit-msg:** This Hook is called after a pre-commit with the objective of populating the text editor with predefined commit message;
- **commit-msg:** This Hook is called after commit message has been entered by the developer, and it should be a good place if you want for example to enforce a certain commit message policy, that follows a certain structure;
- **post-commit:** This Hook is called after commit-msg, and it is only used for notification purposes, because it will not change the outcome of commit operation.

In this code analysis flow, the *commit-msg* hook has the following structure:

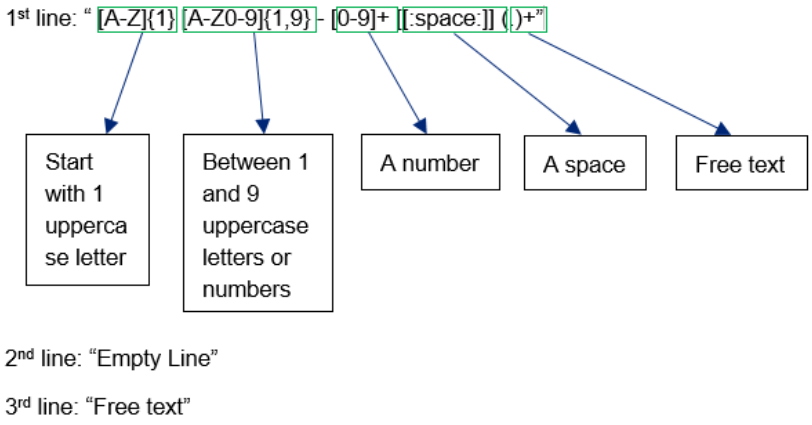


Figure 26.: Commit Message Structure

4.2.1.3 *Pushes code to repository*

Whenever the developer decides to push the code to the repository, he can use the IDE or use the script *script_to_push.sh*. This script has the same structure described in Table 12.

```
bash script_to_push.sh <username> <password> <hostname> <port> <project> <
    repository>
```

Before being able to push the same changes to the master branch, the developer must first take care of the merging operation between those branches. This push operation on the master branch will trigger a server-side hook. Hooks can be set in order to validate changes performed before a user pushes them into the project repository. The pre-receive hook is executed every time somebody uses git push to push commits to the repository and will accept or reject a push to the repository if certain conditions are fulfilled or not.

To enable the Pre Receive Hook, the user must access the Repository Settings page and select the option "Hook". In that page, the user can configure and enable several hooks. After clicking in the edit button for the Pre Receive Hook, the user must configure it. Figure 27 illustrates how the hook can be configured.

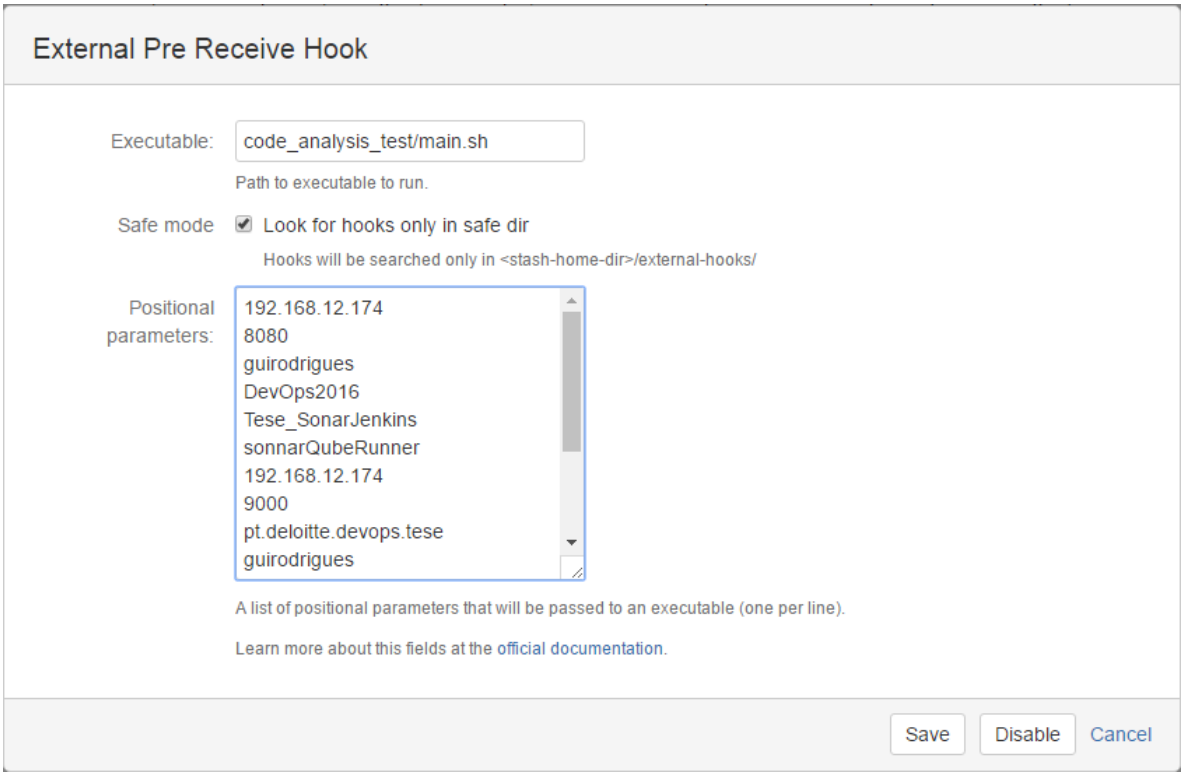


Figure 27.: Pre Receive Hook

The Executable file can be set only by Bitbucket system administrator for security purposes. However, users can set executable relative if safe mode is enabled. Safe mode can be used by non-privileged users for specifying as executable any script that sysadmin will create in <bitbucket_home>/external-hooks/ directory. Positional arguments will be passed to the selected executable as is, one per line and they should be specified without any quotes.

The *main.sh* script execute the following subtasks:

- **Generate pushed files:** First the modified files that are trying to be pushed are generated so they can be analysed by the SonarQube;
- **Check the branch:** Only the defined branches will be analysed, so a verification of the branch will be performed to check if the analysis will proceed or not;

- **Create the sonar-project.properties file:** The sonar-project.properties is a mandatory configuration file on the root directory of the project that set some parameters of the SonarQube analysis;
- **Run SonarQube analysis:** The next step is run a SonarQube analysis. This analysis is incremental, which means that only the modified files are analysed. At the end a file is created with the results of the analysis named *sonar-report.json*. This file presents the list of issues found in the files changed, each one with a level of severity;
- **Get the quality gates:** The next step is to get the quality gates defined for the project from the SonarQube API. The quality gates includes the limit values for each level of severity and also the actual value of issues found for each severity. The values are saved in a file and will be used to validate the changes;
- **Validate the changes:** The issues generated by the changes are grouped by severity and added to the actual number issues found (before this changes). So for each severity the sum will be compared to the limits defined in the quality gates file.

The script will accept or reject a push performed by the user considering the limit values defined to the project. If the push is successful, a Jenkins job with a SonarQube complete analysis will be triggered. The full shell script can be found in Appendix A.1.

To run SonarQube analysis, it is assumed that previously it was created a new project and associated a quality gate to it. In this case, the quality gate is meant to analyse only Java code and its metrics are reflected in Table 13.

Table 13.: SonarQube quality gate

Metric	Condition	Warnings	Errors
Blocker Issues	is greater than	N/A	1
Critical Issues	is greater than	5	10
Info Issues	is greater than	250	N/A
Major Issues	is greater than	10	25
Minor Issues	is greater than	100	200

In case the SonarQube dry run analysis fails, the push will automatically fail as well. Figure 28 illustrates one case when this happens, where the developer tried to push some code to the master branch that had 70 out of 25 major issues allowed.

As a Post Receive Hook, it has also been configured in Bitbucked to send an email to a list of recipients whenever someone pushes to the repository. This hook is called *Push Email Notifier Hook*.

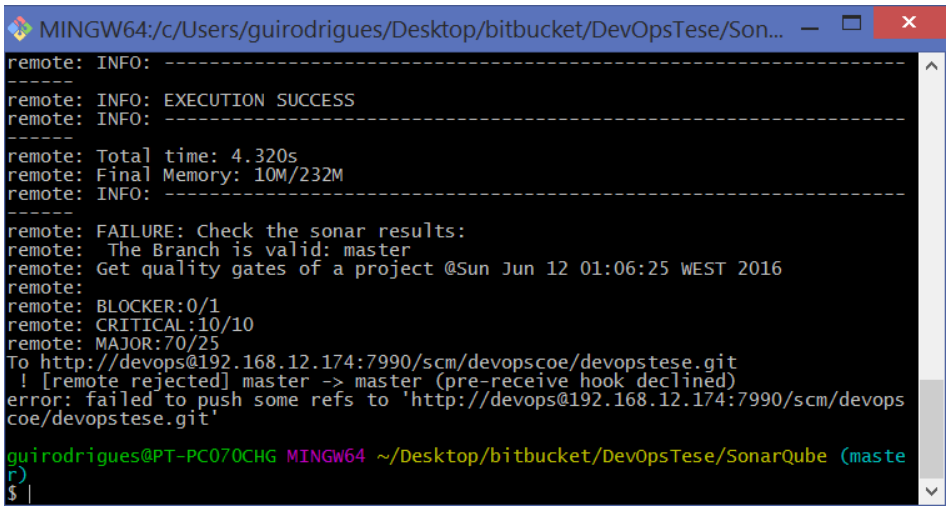


Figure 28.: Pre Receive Hook Failure

4.2.1.4 *Runs full code analysis and publishes report in Confluence*

In case the Jenkins job gets triggered, it means that the dry run analysis passed with success. This job has two main build steps:

- Issues a complete analysis of the repository to SonarQube who will produce a report with the results;
- Store the reports with the analysis in a Confluence page. The main Confluence page, named *SonarQube Analysis* will be created in the first execution. For every analysis execution, it will be created a child page and in the attachment the complete issue report analysis generated by the SonarQube build step. Figure 29 illustrates an example of a generated Confluence page.

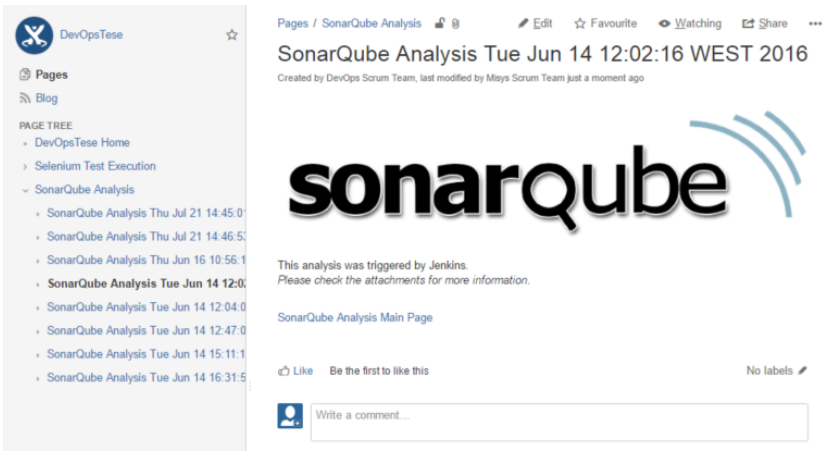


Figure 29.: Confluence Page

4.2.2 *Automated Tests*

During this automated tests flow, the following application from the DevOps suite will be used: Jira, Bitbucket, Zephyr, Selenium, Jenkins and Confluence. The flow illustrated in Figure 30 is divided into 4 main steps which will be detailed in the sub chapters below:

- Tester selects the test issue in JIRA, changing its status to "Automate Test", that will trigger a job in Jenkins;
- Jenkins will call a custom jar that will interact with Zephyr through ZAPI to get the information regarding each individual test in the test suite;
- Jenkins will clone the Bitbucket repository where the tests are coded and start its execution;
- Jenkins will publish the evidences of the automated tests in a Confluence page.

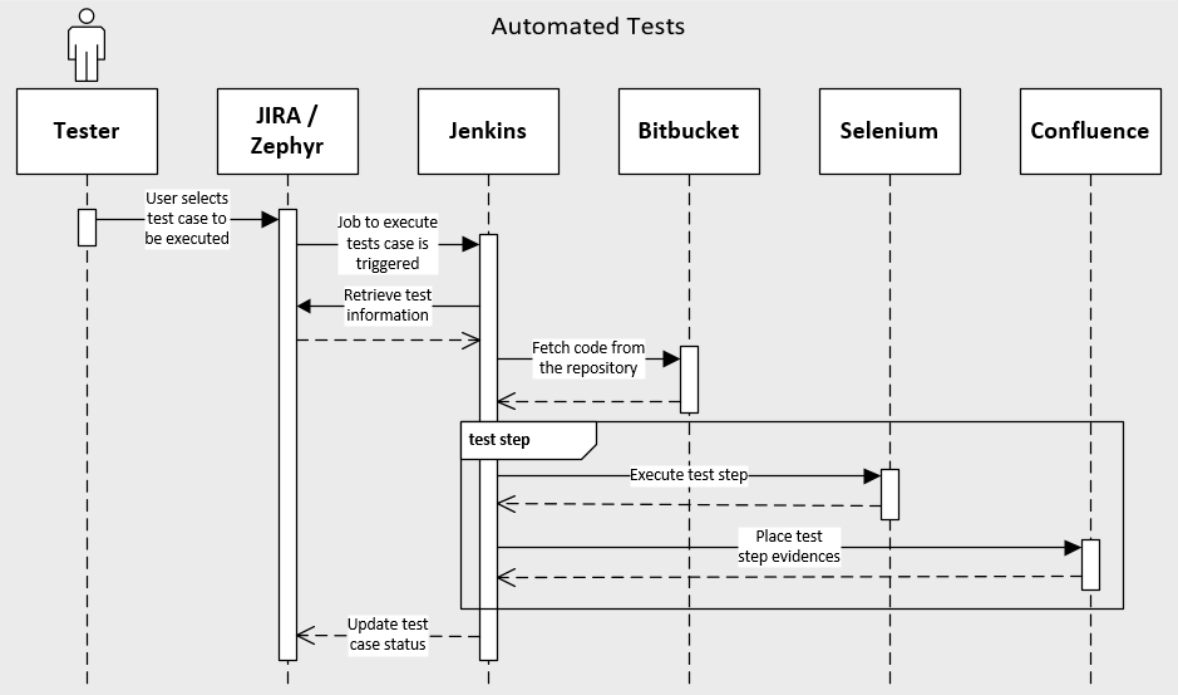


Figure 30.: Automated Tests Flow

4.2.2.1 *Select issue in JIRA*

Having installed the Zephyr plugin for JIRA, it is possible to create an issue of type *Test* and associate it to an existent issue like a user story or a task. Although the purpose of this

issue in its essence is only for test design, it will be used in this flow as the starting point to the automated tests.

On JIRA's system administration panel, it can be configured WebHooks, which are user-defined HTTP POST call-backs. They provide a lightweight mechanism for letting remote applications receive push notifications from JIRA, without requiring polling. To register a WebHook in JIRA, it is necessary to provide the following information:

- **Name** for the WebHook created;
- **URL** where the callback should be sent;
- **Scope** of the WebHook, either "all issues" or a limited set of issues specified by JQL;
- **Events** to post to the URL, either "all events" or a specific set of events.

Figure 31 illustrates an example of a WebHook configured in JIRA. In this example, when the status of an issue of type *Test* is updated to *Automate Test* in JIRA project *DevOps Tese*, it triggers a webhook.

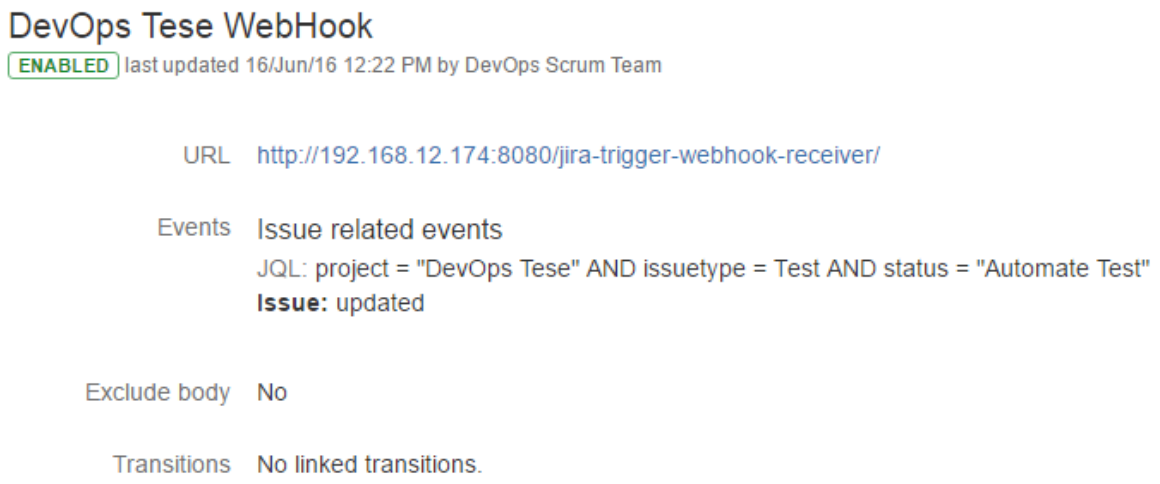


Figure 31.: JIRA WebHook

Every issue in JIRA has a workflow associated, and for the *Test* issue it has been defined the workflow illustrated in Figure 32. In this workflow, the WebHook is triggered whenever the issue status changes from *Open* to *Automate Test* through transaction *Automate Test* or from *Automate Test* to itself through transaction *Repeat Test*.

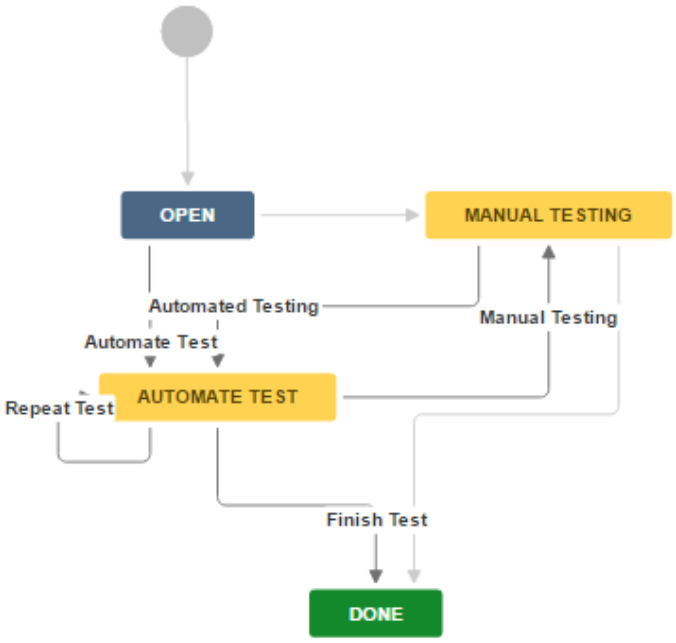


Figure 32.: Test issue workflow

Due to JIRA limitations, the POST call-back does not support basic authentication required by Jenkins to trigger the build. Therefore, a plugin has to be added to Jenkins, which works as a receiver that will be responsible to trigger the build upon the reception of the request from JIRA. When configuring the job in Jenkins, the same JQL filter used in JIRA WebHook needs to be used, as illustrated in Figure 33. There is also the possibility of extracting information from the JIRA issue who triggered Jenkins and store it in variables.

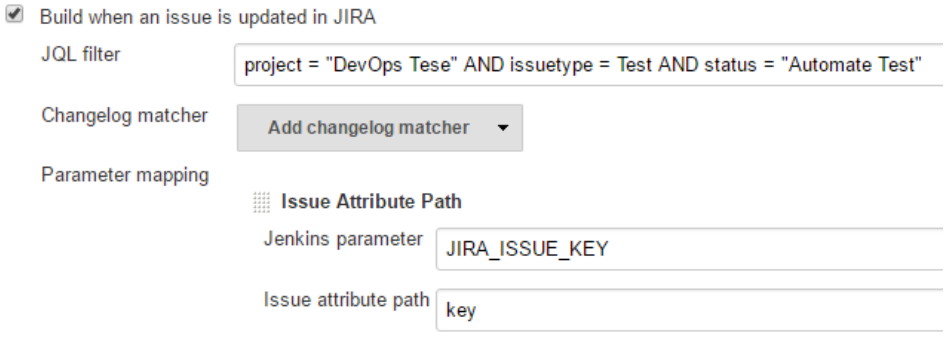


Figure 33.: Jenkins build trigger

4.2.2.2 *Retrieve test information*

In order to retrieve the necessary information about each test step, it has been developed a custom jar file *ZapiJsonParser-1.0.jar* that takes the following as input parameters:

- *JIRA_USERNAME*;
- *JIRA_PASSWORD*;
- *JIRA_SERVER_HOST*;
- *JIRA_SERVER_PORT*;
- *JIRA_ISSUE_KEY*.

This parser has the purpose of posting a new test execution in JIRA, under the test issue, as well as to generate a file, named *Zephyr_test_info.txt* containing information about each individual test step following the structure below (each line in the file corresponds to a test step).

```
projectID#issueID#testExecutionID#issueSummary#issueDescription#testStepID#
    testStepExecutionID#testStepName#testStepData
```

- *projectID*: JIRA project identifier;
- *issueID*: JIRA issue identifier;
- *testExecutionID*: JIRA test execution identifier;
- *issueSummary*: JIRA issue name;
- *issueDescription*: JIRA issue description;
- *testStepID*: JIRA test step identifier;
- *testStepExecutionID*: JIRA test step execution identifier;
- *testStepName*: JIRA test step name;
- *testStepData*: JIRA test step details.

4.2.2.3 *Run Selenium scripts*

When the Jenkins job is triggered, the Bitbucket branch where the tests are coded gets cloned to the Jenkins workspace. Then, and once the JSON information gets parsed by the jar file, generating a file with information about each test step, a shell script is executed to begin Selenium tests. Since these tests follow JUnit annotations, for each line in the *Zephyr_test_info.txt* file, the following maven command is executed:

```
mvn -Dtest=$test_summ#$step_name -DprojectID=$project_id -DtestID=$test_id -
    DtestExecID=$test_exec_id -DtestSum=$test_summ -DtestDesc=\'$test_desc\'
     -DtestStepID=$step_id -DtestStepExecID=$step_exec_id -DtestStepData=\'
    $step_data\' test
```

There are, however, some considerations to bear in mind with this approach. They are as follows:

1. The JIRA issue name must match with the JAVA class name;
2. The JAVA class must have one method with the @Test annotation for each JIRA test step name, and the name must match.

The example below illustrates the necessary skeleton code to execute the test *testStep* of JIRA issue *TestCase*. It consists in a login action performed on the Misys core banking system Essence.

```
public class TestCase {

  /**
   * Log In test step.
   *
   */
   @Test
   public void testStep() {
     WebDriver driver = new FirefoxDriver();
     driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);
     driver.manage().window().maximize();
     driver.get("http://localhost:8080/uxp/rt/html/login.html?locale=en-gb
         ");

     //username
     setValueById(driver, "username", "myUser");

     //password
     setValueById(driver, "password", "myPass");

     //Sign In button
     driver.findElement(By.id("login").click();
   }
```

```
  /**
   * Sets element value by ID
   *
   * @param driver WebDriver in focus
   * @param id Element ID
   * @param value Value to be set
   */
   public static void setValueById(WebDriver driver, String id, String
       value) {
     WebElement element = driver.findElement(By.id(id));
     new Actions(driver).moveToElement(element).perform();
     element.clear();
     element.sendKeys(value);
   }
}
```

Despite the tests being executed in the machine where Selenium server is running, all tests evidences are stored in Jenkins workspace. For that purpose, a folder called *reports* get created in each build, containing a PDF file for each JIRA test step with the test evidences, and also a CSV file is generated containing the following information (one test per line):

```
projectID#issueID#testExecutionID#testStepID#testStepExecutionID#testStatus#
    pdfPath#
```

- *projectID*: JIRA project identifier;
- *issueID*: JIRA issue identifier;
- *testExecutionID*: JIRA test execution identifier;
- *testStepID*: JIRA test step identifier;
- *testStepExecutionID*: JIRA test step execution identifier;
- *testStatus*: Status of test execution. 0 - Failure; 1 - Success;
- *pdfPath*: Path to PDF file containing test evidences.

4.2.2.4 *Publish test evidences*

Once the tests have been executed, it is necessary to place the evidences in a Confluence page and update the status of both the test case and test steps in JIRA. For this 2 goals,

there are 2 shell scripts that run in the Jenkins job. These scripts can be found in Appendix A.2

1. *publish_to_confluence.sh*: For every test step execution a PDF file gets created with the evidences. It is, therefore, necessary to store this evidences in Confluence. The main Confluence page, named *Selenium Test Execution* will be created in the first execution. For every test execution, it will be created a child page and in the attachment the test evidences generated by Selenium. Figure 34 illustrates an example of a generated page.

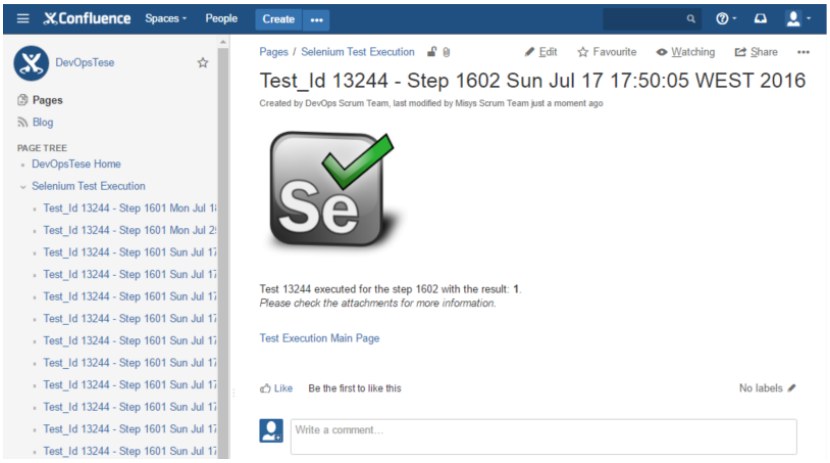


Figure 34.: Selenium Confluence Page

2. *update_tests.sh*: This script analyses the CSV file generated by Selenium in order to decode the result of each test step. It then updates the status based on that information, having in mind that for a test case to fail, it is only necessary that one of its test steps fails. In addition to updating the status of each test step, it also adds a comment containing the hyperlink to the Confluence page containing the test evidences. Below in Figure 35, is an example of both the test case and test steps status in case the execution is successful.

Test Executions

Version	Test Cycle	Status	Defects	Executed By	Executed On
Unscheduled	Ad hoc	PASS	-	DevOps Scrum Team	Yesterday 6:43 PM

Test Details

	Test Step	Expected Result	Status	Comment
1	matchBirthDatePass	The age should match the test data birth date	PASS	192.168.12.220:8090/pages/viewpage.action?pageId=3343801
2	matchBirthDateFail	The age should NOT match the test data birth date	PASS	192.168.12.220:8090/pages/viewpage.action?pageId=3343803

Figure 35.: Test Status

4.2.3 *Deployment*

During this deployments flow, it will be taken into consideration a functionality available in Jenkins, which is basically a post-action of a job that gives the opportunity to immediately begin the execution of a new job, in case the previous one had been successful.

Therefore, the deployment flow will be triggered whenever the job containing the automated tests is completed with success, creating the software package with the automated tests code and deploying it to an Application Server. The flow illustrated in Figure 36 is divided into 2 main steps which will be detailed in the sub chapters below:

- Execute the automated tests process;
- Jenkins will create a war file and deploy it to a Wildfly application server.

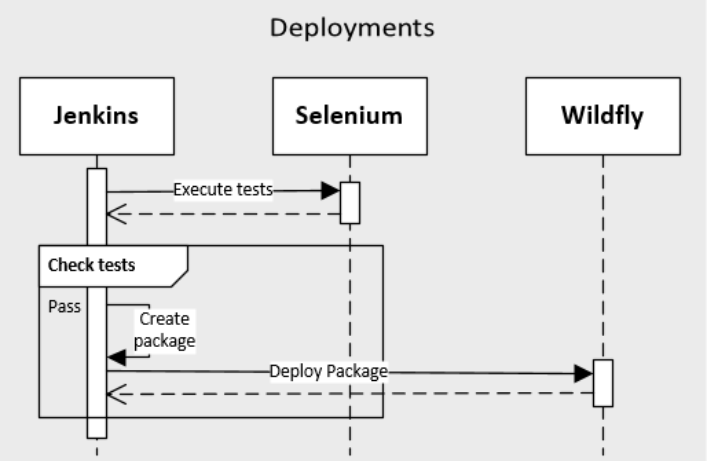


Figure 36.: Deployment Flow

4.2.3.1 *Execute the automated tests process*

Process detailed in section 4.2.2.

4.2.3.2 *Create and deploy the package*

Once the automated tests have been successfully executed, the packaging and deployment job will be triggered by a jenkin's post-build action. With the post-build action illustrated in Figure 37, it is possible to guarantee that the deployment job is only triggered when the automated tests build is stable.

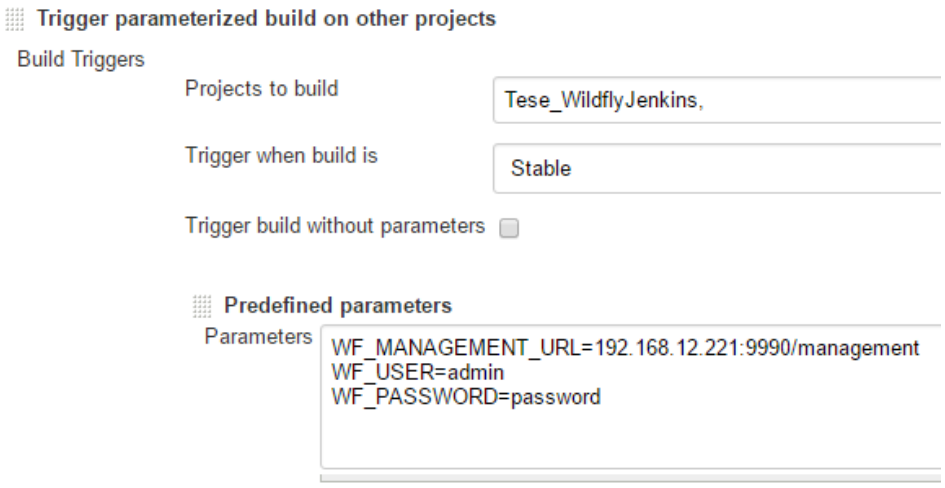


Figure 37.: Post-build Action

The packaging and deployment scripts, which can be found in Appendix A.4, is composed of the following steps:

- Use the maven package command to create the new war file;
- Undeploy the existent war file, if it exists, from the application server;
- Remove the existent war file, if it exists, from the application server;
- Upload and enable the new war file to application server.

5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSIONS

In this dissertation project, there were mainly five objectives:

- Describe the development lifecycle of a solution under the DevOps philosophy;
- Describe an automated code analysis strategy;
- Describe an automated testing strategy;
- Describe a strategy to promote software packages through multiple environments and deployments to production environment;
- Analyse a selection of tools, used to support the previously enumerated strategies.

Of the above goals, only the fourth one has been partially accomplished, where the deployment strategy does not cover multiple environments. Apart from that all of the other proposed goals have been successfully accomplished and the results exceeded the initial expectations, especially due to the fact that it is already being used on a real project scenario. The automated code analysis and automated tests flows highly diminish the time a developer/tester takes during the development phase, while reducing the probability of encountering a problem at a later stage of the delivery pipeline.

By integrating the tools identified in the DevOps suite, creating the flow of continuous delivery, automating the processes of code analysis, functional tests and deployment, it is clear the benefits it can bring to any organisation. In the scope of this dissertation, it can be highlighted one benefit for each the Development and the Operations team:

- Team Dev: The ability of triggering the code analysis process and generating reports for each push to the repository master branch;
- Team Ops: The ability to trigger the test scripts from the Project Tracking tool and, in case of success, create and deploy a software package to an application server.

In addition to defining and developing the automated processes of code analysis, automated tests and deployments, it has all been put to practise in a concrete live scenario. The demo was performed on the Misys core banking application Essence with the following inputs/outputs:

- The requirement was to add a new custom field to a screen;
- The field had some business logic behind, coded in Java;
- The code was analysed, before being pushed to the repository;
- An automated test was triggered in order to validate the functionality of the requirement;
- A software package was built after the tests finished with success.

Globally, the outcome of this dissertation was very well received within Deloitte, and a set of actions have already been taken and others planned:

- Nearly 60% of the DevOps suite is already being successfully used on a daily basis in a project with over 40 people, covering the tools Jira, Zephyr, Confluence, Bitbucket, SonarQube, Selenium and Jenkins;
- Several sessions were held to educate those people on the DevOps philosophy and how to take the best possible value out of the tools;
- The test automation strategy was the main driver to win a project on a major Portuguese Bank;
- There is the intention of expanding this DevOps philosophy to other projects across the firm and complete the full integration of the rest of the tools.

5.2 PROSPECT FOR FUTURE WORK

In the context of this dissertation, a considerable amount of follow-up work could be carried out. First of all, the conclusion of the automated deployment strategy, with the help of the configuration management tool identified for the DevOps suite, Puppet. Additionally, there are other opportunities that could be further explored:

- Automate the code review process;
- Automate the performance tests and continuous monitoring processes;

- Make use of a repository manager tool, such as Artifactory, to centralize all of the dependency packages;
- Observe the behaviour of a real client adoption scenario, and analyse the results of embracing the DevOps philosophy.

REFERENCES

D. Androcec and Z. Dobrovic. Creating hybrid software engineering methods by means of metamodels. In *Information Technology Interfaces (ITI), Proceedings of the ITI 2012 34th International Conference on*, pages 481–486, June 2012.

Lianping Chen. Continuous delivery: Huge benefits, but challenges too. *Software, IEEE*, 32(2):50–54, Mar 2015.

D.A. Dewi and M. Muniandy. The agility of agile methodology for teaching and learning activities. In *Software Engineering Conference (MySEC), 2014 8th Malaysian*, pages 255–259, Sept 2014.

R.E.D. Fairley, P. Bourque, and J. Keppler. The impact of swebok version 3 on software engineering education and training. In *Software Engineering Education and Training (CSEE T), 2014 IEEE 27th Conference on*, pages 192–200, April 2014.

Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison Wesley, August 2010.

Jez Humble, Joanne Molesky, and Barry O'Reilly. *Lean Enterprise: How High Performance Organizations Innovate at Scale*. O'Reilly Media, Inc., August 2010.

S. Jantunen, L. Lehtola, D.C. Gause, U.R. Dumdum, and R.J. Barnes. The challenge of release planning. In *Software Product Management (IWSPM), 2011 Fifth International Workshop on*, pages 36–45, Aug 2011.

Gene Kim, Kevin Behr, and George Spafford. *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*. IT Revolution Press, January 2013.

T.J. Lehman and A. Sharma. Software development as a service: Agile experiences. In *SRII Global Conference (SRII), 2011 Annual*, pages 749–758, March 2011.

IT Revolution Puppet Labs. 2015 state of devops report. Technical report, Puppet Labs, 2015.

M. Rossi, M. Taisch, and S. Terzi. Lean product development: A five-steps methodology for continuous improvement. In *Engineering, Technology and Innovation (ICE), 2012 18th International ICE Conference on*, pages 1–10, June 2012.

Sanjeev Sharma and Bernie Coyne. *DevOps for Dummies*. John Wiley & Son, Inc., 2015.

M. Virmani. Understanding devops bridging the gap from continuous integration to continuous delivery. In *Innovative Computing Technology (INTECH), 2015 Fifth International Conference on*, pages 78–82, May 2015.

Mandi Walls. *Building a DevOps Culture*. O'Reilly Media, Inc., April 2013.

A

LISTINGS

A.1 MAIN.SH

```
#!/bin/sh

#Jenkins arguments
JENKINS_HOST=$1
JENKINS_PORT=$2
JENKINS_USER=$3
JENKINS_PWD=$4
JOB_NAME=$5
TOKEN_NAME=$6

#Sonar arguments
SONAR_HOST=$7
SONAR_PORT=$8
PROJECT_KEY=$9
SONAR_USER=${10}
SONAR_PWD=${11}
PROJECT_NAME=${12}
VERSION=${13}
SOURCE=${14}
LANGUAGE=${15}
#List of branches to be tested, the others will be discarded
BRANCHES_LIST=${16}

#name of the folder with the code changed and that will be analysed
CODE_DIR=analysis
#Name of the files with some reports
```

```
FILE_RESULT="results.txt"
#Variable with final result
FINAL_RESULT=-1

#check if all the parameters are passed
if [ $# -ne 16 ]; then
  echo "------------------"
  echo "ERROR: Not enough arguments!"
  echo "USAGE bash main.sh <JENKINS_HOST> <JENKINS_PORT> <JENKINS_USER>
  <JENKINS_PWD> <JOB_NAME> <TOKEN_NAME> <SONAR_HOST> <SONAR_PORT> <
     PROJECT_KEY>
  <SONAR_USER> <SONAR_PWD> <PROJECT_NAME> <VERSION> <SOURCE> <LANGUAGE>
  <BRANCHES_LIST>"
  echo "------------------"
  exit -1
else
  echo ${PWD}
  HOOK_DIR=/var/atlassian/application-data/bitbucket/deloitte/external-
   hooks/code_analysis

  #create folder with changed files in push and return the branch name
  BRANCH_NAME=$(bash $HOOK_DIR/create_pushed_files.sh $CODE_DIR)

  #check if this branch is suposed to be tested
  RESULT_BRANCH=$(bash $HOOK_DIR/validate_branch.sh $BRANCH_NAME $
     BRANCHES_LIST)

  if (($RESULT_BRANCH == 0)); then
       echo -e "The Branch is valid: $BRANCH_NAME" >> $CODE_DIR/
          $FILE_RESULT
  else
      echo "No analysis was performed! BRANCH:$BRANCH_NAME BRANCHES_VALID:
         $BRANCHES_LIST"
      exit 0
  fi

  #generate sonar-project.properties
```

```
  bash $HOOK_DIR/generate_sonar_properties.sh $CODE_DIR $PROJECT_KEY $
      PROJECT_NAME $VERSION $SOURCE $LANGUAGE

  #run the sonarqube analysis, if the sonar-report.json not exists the
   sonar analysis went wrong
  cd $CODE_DIR && /opt/sonar-runner-2.4/bin/sonar-runner -Dsonar.analysis.
      mode=incremental -Dsonar.issuesReport.json.enable=true

  if [ ! -f ${PWD}/.sonar/sonar-report.json ]; then
    echo "ERROR: File JSON not created!"
    exit -1
  fi

  #get quality gates report from the Sonarqube API: if result != 0
      something went wrong with the connection with the Sonar API
  RESULT_SONAR_METRICS=$(bash $HOOK_DIR/get_project_metrics.sh $SONAR_HOST
      $SONAR_PORT $PROJECT_KEY $SONAR_USER $SONAR_PWD $FILE_RESULT)

  if (($RESULT_SONAR_METRICS != 0)); then
    echo -e "ERROR: Metrics file not created! Check the error:\n $(cat
        $FILE_RESULT)"
    exit -1
  fi

  #validate push
  RESULT_SONAR_ANALYSIS=$(bash $HOOK_DIR/validate_push.sh "${PWD}/.sonar/
      sonar-report.json" "resources.json" "$HOOK_DIR")
  if (( $RESULT_SONAR_ANALYSIS == 0)); then
    curl $JENKINS_HOST:$JENKINS_PORT/job/$JOB_NAME/build?token=$TOKEN_NAME
        --user $JENKINS_USER:$JENKINS_PWD
    echo -e "SUCCESS: Check the sonar results:\n $(cat $FILE_RESULT)"
 exit 0
  else
    echo -e "FAILURE: Check the sonar results:\n $(cat $FILE_RESULT)"
    exit -1
  fi
fi
```

src/main.sh

A.2 PUBLISH_TO_CONFLUENCE.SH

```
#!/bin/bash

#file CSV with the tests information
#<PROJECT_ID>#<TEST_ID>#<TEST_EXEC_ID>#<STEP_ID>#<STEP_EXEC_ID>#<
    STEP_STATUS>#<PDF_PATH>#
FILE_CSV=$1
USERNAME=$2
PASSWORD=$3
CONFLUENCE_SERVER_HOST=$4
CONFLUENCE_SERVER_PORT=$5

JSON=$(curl -H "Content-Type: application/json" -u $USERNAME:$PASSWORD -X
   GET $CONFLUENCE_SERVER_HOST:$CONFLUENCE_SERVER_PORT/rest/api/content?
   title=Selenium+Test+Execution)
ANCESTOR_PAGE_ID=$(echo $JSON | grep -Po '(?<={"id":")[^"]*')
ANCESTOR_PAGE_TITLE="Selenium Test Execution"

#Create Ancestor Page if it does not exists
if [ "$ANCESTOR_PAGE_ID" = "" ]; then

    JSON=$(curl -u $USERNAME:$PASSWORD -X POST -H 'Content-Type:
        application/json' -d'{"type":"page","title":"'"$ANCESTOR_PAGE_TITLE
        "'", "space":{"key":"DEVTESE"},"body":{"storage":{"value":"This is
        the main page where Selenium test evidences will be saved.","
        representation":"storage"}}}' $CONFLUENCE_SERVER_HOST:
        $CONFLUENCE_SERVER_PORT/rest/api/content/)
    ANCESTOR_PAGE_ID=$(echo $JSON | grep -Po '(?<={"id":")[^"]*')

fi

#get new ancestor page id by the json response of the POST
ANCESTOR_PAGE_ID=$(echo $JSON | grep -Po '(?<={"id":")[^"]*')

#for each line - each line has test information
while read line;
do
```

```
SAVE_IFS=$IFS
IFS='#'

read -r -a array <<< "$line"

#get test information separated by #
PROJECT_ID=${array[0]}
TEST_ID=${array[1]}
TEST_EXEC_ID=${array[2]}
STEP_ID=${array[3]}
STEP_EXEC_ID=${array[4]}
STEP_STATUS=${array[5]}
PDF_PATH=${array[6]}
#today's date
now=$(date)

#information that will be set in confluence page
TITLE="Test_Id $TEST_ID - Step $STEP_ID $now"
PAGE_TEXT="<ac:image ac:height='125'><ri:url ri:value='http://squeeze3.
    deloittedigital.pt/devops' /></ac:image>"
PAGE_TEXT+="<p><br />Test $TEST_ID executed for the step $STEP_ID with the
     result: <strong>$STEP_STATUS</strong>. <br /> <em>Please check the
    attachments for more information.</em></p>"
PAGE_TEXT+="<br /><ac:link><ri:page ri:content-title='$ANCESTOR_PAGE_TITLE
    ' /><ac:plain-text-link-body><![CDATA[Test Execution Main Page]]></ac:
    plain-text-link-body></ac:link>"

#create new page in confluence
JSON=$(curl -u $USERNAME:$PASSWORD -X POST -H 'Content-Type: application/
    json' -d'{"type":"page","title":"'"$TITLE"'", "ancestors":[{"id":"'"$
    ANCESTOR_PAGE_ID"'"}], "space":{"key":"DEVTESE"},"body":{"storage":{"
    value":"'"<p>$PAGE_TEXT</p>"'","representation":"storage"}}}' $
    CONFLUENCE_SERVER_HOST:$CONFLUENCE_SERVER_PORT/rest/api/content/)

#get new page id by the json response of the POST
PAGE_ID=$(echo $JSON | grep -Po '(?<=^{"id":")[^"]*')
```

```
 #upload file in $PDF_PATH to the page with the id $PAGE_ID
 curl -v -S -u $USERNAME:$PASSWORD -X POST -H "X-Atlassian-Token: no-check"
      -F "file=@$PDF_PATH" -F "comment=Test evidence for step $STEP_ID" "
     $CONFLUENCE_SERVER_HOST:$CONFLUENCE_SERVER_PORT/rest/api/content/
     $PAGE_ID/child/attachment"

 #set new path of the pdf in confluence
 NEW_PATH="$CONFLUENCE_SERVER_HOST:$CONFLUENCE_SERVER_PORT/pages/viewpage.
     action?pageId=$PAGE_ID"

 #write to file the new information to new file
 echo "$PROJECT_ID#$TEST_ID#$TEST_EXEC_ID#$STEP_ID#$STEP_EXEC_ID#
     $STEP_STATUS#$NEW_PATH" >> tests.txt
done <$FILE_CSV
```

src/publish_to_confluence.sh

A.3 update_tests.sh

```
#!/bin/sh

#Analyze automated tests

#USAGE bash update_tests.sh <RESULT_FILE> <JIRA_USER> <JIRA_PASS> <
    JIRA_SERVER> <JIRA_PORT>
#Example: bash update_tests.sh tests.txt JiraUser JiraPwd 192.168.12.100
    8080

RESULT_FILE=$1
JIRA_USER=$2
JIRA_PASS=$3
JIRA_SERVER=$4
JIRA_PORT=$5

# jira codes for pass and fail status
PASS=1
FAIL=2
```

```
TEST_STATUS=$PASS

#check if all the parameters are passed
if [ $# -ne 5 ]
then
    echo "------------------"
    echo "Not enough arguments!"
    echo "USAGE bash update_tests.sh <RESULT_FILE> <JIRA_USER> <JIRA_PASS>
        <JIRA_SERVER> <JIRA_PORT>"
    echo "------------------"
    exit -1
else
    while read line;
    do
     SAVE_IFS=$IFS
        IFS='#'

        read -r -a array <<< "$line"

   project_id=${array[0]}
   test_id=${array[1]}
        test_exec_id=${array[2]}
        step_id=${array[3]}
   step_exec_id=${array[4]}
        step_status=${array[5]}
        pdf_link=${array[6]}

   # if a single test step fails, the whole test cycle fails as well
   if [ $TEST_STATUS -eq $PASS ] && [ $step_status -eq $FAIL ]
   then
       TEST_STATUS=$FAIL
   fi

   # update the test step
   ./change_test_step_status.sh $JIRA_USER $JIRA_PASS $JIRA_SERVER $
       JIRA_PORT $step_id $step_exec_id $step_status $pdf_link

   IFS=$SAVE_IFS
```

```
    done < $RESULT_FILE

 # update the test cycle
 ./change_test_status.sh $JIRA_USER $JIRA_PASS $JIRA_SERVER $JIRA_PORT $
     test_exec_id $TEST_STATUS

    exit 0
fi
```

src/update_tests.sh

A.4 DEPLOY_PACKAGE.SH

```
#!/bin/sh
cd DevOps

mvn package -Dmaven.test.skip=true

echo "Undeploy old war"
curl -S -H "content-Type: application/json" -d '{"operation":"undeploy", "
    address":[{"deployment":"DevOps-1.0-SNAPSHOT.war"}]}' --digest http://$
    WF_USER:$WF_PASSWORD@$WF_MANAGEMENT_URL
echo ""

echo "Remove old war"
curl -S -H "content-Type: application/json" -d '{"operation":"remove", "
    address":[{"deployment":"DevOps-1.0-SNAPSHOT.war"}]}' --digest http://$
    WF_USER:$WF_PASSWORD@$WF_MANAGEMENT_URL
echo ""

echo "Upload new war"
BYTES_VALUE=`curl -F "file=@target/DevOps-1.0-SNAPSHOT.war" --digest http
    ://$WF_USER:$WF_PASSWORD@$WF_MANAGEMENT_URL/add-content | perl -pe 's
    /^.*"BYTES_VALUE"\s*:\s*"(.*)".*$/$1/'`
echo ""

JSON_STRING_START='{"content":[{"hash": {"BYTES_VALUE" : "'
```

```
JSON_STRING_END='"}}], "address": [{"deployment":"DevOps-1.0-SNAPSHOT.war"
    }], "runtime-name":"DevOps-1.0-SNAPSHOT.war", "operation":"add", "
    enabled":"true"}'
JSON_STRING="${JSON_STRING_START}${BYTES_VALUE}${JSON_STRING_END}"

echo "Deploy new war"
RESULT=`curl -S -H "Content-Type: application/json" -d "${JSON_STRING}" --
    digest http://$WF_USER:$WF_PASSWORD@$WF_MANAGEMENT_URL | sed -ne "s/.*
    outcome\" *: *\"\([a-zA-Z]\+\).*/\1/p"`
echo "Deployment result: ${RESULT}"
echo ""

if [ "$RESULT" != "success" ]; then
  exit -1
fi
```

src/deploy_package.sh