



Escola d'Enginyeria de Telecomunicació i  
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# MASTER THESIS

**TÍTULO DEL TFM:**

Continuous integration and monitorization system for code quality and cooperation work

**TITULACIÓN:** Master's degree in Applied Telecommunications and Engineering Management

**AUTOR:** Carles Artigas Morales

**DIRECTOR:** Roc Meseguer

**FECHA:** 4/07/2017

**Título:** Continuous integration and monitorization system for code quality and cooperation work

**Autor:** Carles Artigas Morales

**Director:** Roc Meseguer Pallarès

**Fecha** 4/07/2017

## Resumen

Existen varias herramientas que permiten una mayor gestión de los proyectos de programación e incluso permiten a partir del uso de estas herramientas mejorar la eficiencia y eficacia en el desarrollo de la aplicación así como en las capacidades del desarrollador.

Por tanto en este proyecto se presentaran, implementaran y testearan un conjunto de herramientas para ese propósito. Con lo que este proyecto consiste principalmente sobre la calidad del código y la cooperación entre desarrolladores, para ello se utilizaran diferentes Softwares.

Sonar se utilizara como herramienta de análisis de código ya que provee de un potente feedback que se analizara en el proyecto.

Artifactory se utilizara para aumentar la velocidad de compilación y para salvaguardar la máxima cantidad de ancho de banda posible

Jenkins se utilizara como herramienta de integración continua y será el nexo de unión entre Sonar i Artifactory

Jira se utilizara como herramienta de tracking, de cooperación entre trabajadores y se alimentara del feedback de Sonar.

Nagios será la herramienta que monitorizara que todo esté funcionando correctamente y avisara si algún fallo se produce.

Todo esto se simulara un escenario en un entorno lo más real posible dando máxima importancia a la cooperación y la calidad de código, así como las herramientas que provee el Software implementado.

**Title:** Continuous integration and monitorization system for code quality and cooperation work

**Author:** Carles Artigas Morales

**Director:** Roc Meseguer Pallarès

**Date:**

## **Abstract**

Nowadays it exist a sort of tools that allow a greater management of developing projects and although it allows by using this tools increase the efficiency and the effectiveness of the developing process and the developer knowledge.

Therefore in this project it will be presented, implemented and tested a set of tool for this purpose.

This Project will mainly analyze the code quality and de workers cooperation, in order to do that different software will be used.

Sonar will be used as the tool of code analysis which provides a powerful feedback that will be explained in this project.

Artifactory will be used to speed up the building time and to save as much bandwidth as possible.

Jira will be used as a tracking and cooperation tool and will be fed using sonar feedback.

Nagios will monitorize that everything is working as it has and will warn whether a fail happens.

Everything will be simulated in scenario trying to get close to reality and giving maximum importance to the cooperation and code quality. Furthermore, the implemented software will be analyzed as well



# INDEX

<b>CHAPTER 1. INTRODUCTION.....</b>	<b>1</b>
<b>1.1 Thesis objective.....</b>	<b>1</b>
<b>1.2 Methodology and motivation .....</b>	<b>2</b>
1.2.1 Personal Motivation.....	2
<b>CHAPTER 2: TECHNOLOGIES AND SCENARIO OVERVIEW .....</b>	<b>3</b>
<b>2.1 Scenario Overview .....</b>	<b>3</b>
<b>2.2 Git and Bitbucket.....</b>	<b>5</b>
<b>2.3 Jenkins .....</b>	<b>5</b>
<b>2.4 SonarQube.....</b>	<b>6</b>
<b>2.5 Artifactory.....</b>	<b>7</b>
<b>2.6 Nagios.....</b>	<b>8</b>
2.6.2 NRPE.....	8
<b>2.7 Jira.....</b>	<b>9</b>
2.7.1 Features.....	10
<b>2.8 Required Technologies for using above Software .....</b>	<b>10</b>
2.8 .1 Apache.....	11
2.8 .2 PHP .....	11
2.8 .3 Mysql.....	11
2.8 .4 Java .....	12
<b>CHAPTER 3. SCENARIO .....</b>	<b>13</b>
<b>3.1 Configuration of the scenario .....</b>	<b>13</b>
3.1.1 SmartGit, Jira and Bitbucket configuration.....	13
3.1.2 Jenkins .....	18
3.1.3 Sonar.....	23
<b>3.2 Nagios .....</b>	<b>32</b>
3.2 .1 Soft States .....	35
3.2.2 Hard States.....	36
3.2 .3 Monitored Services .....	36
<b>3.3 Economic Summary.....</b>	<b>38</b>
<b>3.4 Linux Server.....</b>	<b>39</b>
3.4.1 Ports .....	39
<b>3.5 Learned lessons .....</b>	<b>40</b>
<b>CHAPTER 4: QUALITY CODE AND DEVELOPMENT COOPERATION .....</b>	<b>41</b>

<b>4.1 Development cooperation .....</b>	<b>41</b>
<b>4.2 Measures of Quality Code .....</b>	<b>43</b>
4.2.1 Reliability Remediation Effort.....	43
4.2.2 Security Remediation Effort.....	44
4.2.3 Maintainability Remediation Effort.....	45
4.2.4 Complexity .....	45
4.2.5 Duplication.....	46
<b>4.3 Projects Comparative.....</b>	<b>47</b>
<b>4.4 Learned lessons .....</b>	<b>48</b>
<b>CHAPTER 5: CONCLUSIONS AND FORECASTING.....</b>	<b>49</b>
5.1 Conclusions .....	49
5.2 Forecasting .....	49

## FIGURES

Fig. 2. 1 Scenario Overview.....	3
Fig. 2. 2 Communication between the different tools.....	4
Fig. 2. 3 Bitbukeect logo .....	5
Fig. 2. 4 Integration tool diagram .....	6
Fig. 2. 5 Artifactory logo.....	8
Fig. 2. 6 Nagios logo .....	8
Fig. 2. 7 Direct checks .....	9
Fig. 2. 8 Indirect checks.....	9
Fig. 2. 9 Jira logo .....	9
Fig. 2. 10 Jira concept .....	10
Fig. 2. 11 Apache logo.....	11
Fig. 2. 12 PHP logo .....	11
Fig. 2. 13 MySQL logo.....	12
Fig. 3. 1 Binding of users between Jira and Bitbucket.....	14
Fig. 3. 2 Pending tasks from the first sprint .....	14
Fig. 3. 3 Activity pannel of Jira .....	15
Fig. 3. 4 Jira workflow .....	15
Fig. 3. 5 Bitbucket branches.....	16
Fig. 3. 6 SmartGirt project control using branches .....	16
Fig. 3. 7 Branches control in Bitbucket.....	17
Fig. 3. 8 Jenkins log file .....	18
Fig. 3. 9 Jenkins direction access .....	18
Fig. 3. 10 Jenkins connections with different tools .....	19
Fig. 3. 11 Artifactory download library thanks to the parsed pom.xml.....	19
Fig. 3. 12 Creation of new jar files.....	20
Fig. 3. 13 Snapshot from Jenkins.....	21
Fig. 3. 14 Snapshot with different number of artifacts and dependencies.....	21
Fig. 3. 15 Snapshots tree .....	21
Fig. 3. 16 Umate Dashboard Page.....	28
Fig. 3. 17 Adding state of the issue to Blocker, Critical and Major .....	28
Fig. 3. 18 Blocker issues.....	28
Fig. 3. 19 Critical issues.....	29
Fig. 3. 20 Critical issue.....	30
Fig. 3. 21 Result obtained based on the quality gates .....	31
Fig. 3. 22 Complexity found .....	31
Fig. 3. 23 Bugs and Vulnerabilities obtained .....	31
Fig. 3. 24 Nagios communication .....	32
Fig. 3. 25 Nagios access web .....	32
Fig. 3. 26 Nagios tracking services table.....	34
Fig. 3. 27 Nagios message when a critical issue occurred (related with Sonar) .....	35
Fig. 3. 28 App passwords .....	37
Fig. 4. 1 Flow of the scenario.....	41
Fig. 4. 2 New sprint meeting .....	42

Fig. 4. 3 Sprint view.....	42
Fig. 4. 4 Measuring the quality of the code: reliability.....	44
Fig. 4. 5 Rating Reliability of each class .....	44
Fig. 4. 6 Security Remediation Effort.....	45
Fig. 4. 7 remediation effort and security rating .....	45
Fig. 4. 8 Maintainability Remediation Effort.....	45
Fig. 4. 9 Complexity found in each class .....	46
Fig. 4. 10 API calls .....	46
Fig. 4. 11 Duplication analysis result.....	46
Fig. 4. 12 Comparative .....	47



## TABLES

Table 3. 1 Maven example Building time.....	20
Table 3. 2 Umate example Building time.....	20
Table 3. 3 Severity .....	26
Table 3. 4 Reliability .....	26
Table 3. 5 Security .....	26
Table 3. 6 Maintanability .....	26
Table 3. 8 Capex.....	38
Table 3. 9 Opex.....	38
Table 3. 10 Time for setting up.....	39
Table 3. 11 Ports summary .....	40
Table 4. 1 Building, code lines and Complexity .....	48

---

## ***Dedicatoria y agradecimientos***

---

Me gustaría agradecerle a Roc Meseguer todo el tiempo que ha dedicado, quién ha sido mi tutor tanto de este proyecto como el de Final de Grado.

---

Muchísimas gracias.

# CHAPTER 1. INTRODUCTION

This Project will mainly analyze the code quality and de workers cooperation, in order to do that different software will be used. This software will be binded together with the intention to create a continuous integration system with monitorization capabilities

## 1.1 Thesis objective

In this thesis it is described the setting up of a continuous integration system and a monitoring system with an Opensource and Freeware software. Jenkins, SonarQube, Artifactory, Bitbucket and Nagios run in the same server in order to provide to a group of developers the tools to have version control and standardization of the code that they are developing. Also, the system is monitored to know if any of the tools deployed as a service are running or failing in order to act before the fail gets noticed. Consequently, the developers will be forced to follow a set of rules previously defined with the aim of homogenize the progress, the growing and the development of the project.

Things that will be automatized:

- Build of the project when pushing up the code.
- Version control and store previously versions.
- Analysis feedback of the code following some set of rules previously defined.
- Auto-download of the dependencies and stored in the local server for future uses.

Using these tools the developers project will be improved and some metrics test of the code would be ran in order to recover some statistics to check that the standardization of the project. This also improves thing the building time, readability of the code, developers cooperation and commit control.

The project its focused mainly in automatization, tracking and developing whereby a set of tools will be installed and binded together in order to create a continuous integration and monitoring system.

Then the second objective of this thesis will be the analysis of the code with Sonar, to retrieve some metrics of code quality of the different projects and compare them. The analysis would be able to check the cooperation between developers using tracking tools as Jira.

Another objective consisted of checking from a web interface if all the services are running or are failing. In the second case, be warned before the outage comes.

This thesis will not stand in a theoretical point of view. Therefore, everything will be tested as it were a real scenario

In summary:

- Jira will track all the development process and developers cooperation
- Bitbucket will store securely the code and push histogram.
- Sonar will scan the code and will output some metrics and statistics and how to improve it
- Artifactory will store all dependencies previously download and self created or current state snapshots.
- Jenkins is the integration tool, the point of binding.
- Nagios will monitorize that everything is running.

## 1.2 Methodology and motivation

The development process is a key factor in project viability that affects to its planification and its execution. Then, it has a special relevance when it is a corporation project.

In order to get out a product to the market, it is very important to coordinate all the components that are involved. This is the reason to adopt a work methodology that has all the tools, models and methods to improve the process of the development without forgetting the quality.

At the work is advancing, complexity grows and in order to be able to provide viability it is important to automatize the maximum possible task. Using this system, a company will be able to control how the complexity grows. This is why a continuous delivery system will be developed in this thesis.

### 1.2.1 Personal Motivation

Be able to learn and create value for yourself with what you just have learn is very important key factor for me. That will make me grow as engineer and as a person.

What really teach you the university is in fact, to learn and evolve by yourself. In the university they give you the tools (partially), the subjects and they simulate a real problem with certain conditions. At the same time, they want you to seek for yourself the answers to the problems that they gave to you with the tools that they provide. This way, when the real problem face up, we are so habituate to struggling out our minds that we are able to solve it.

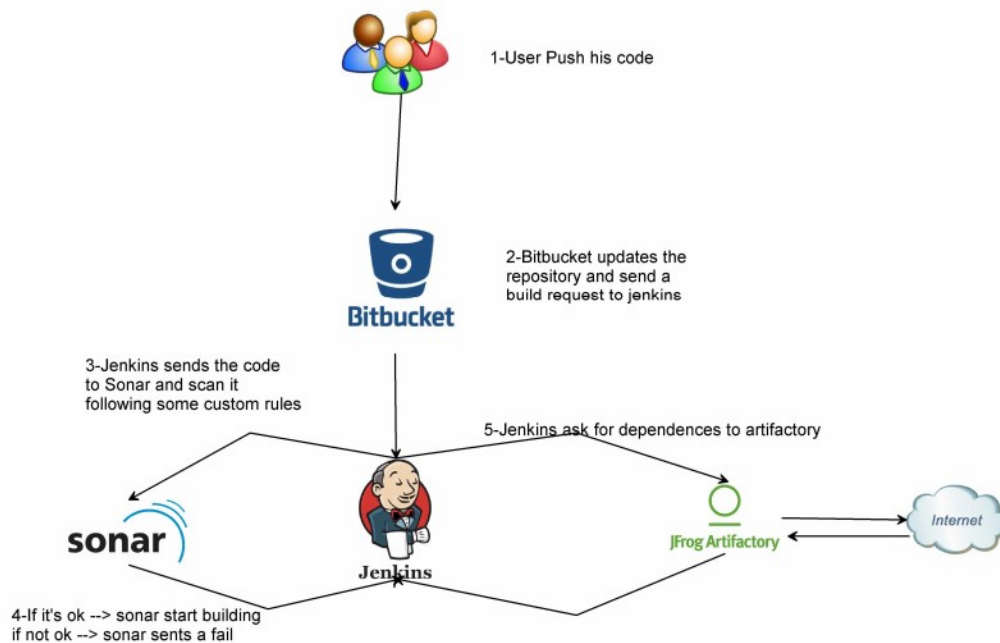
The Master thesis is a way to pose a problem or need and seek a real and effective solution as the want that has been posed in this thesis.

## CHAPTER 2: TECHNOLOGIES AND SCENARIO OVERVIEW

In this chapter all the technologies and software are described in a basic way to help the lector to understand the use of each one of them.

All the software has been installed as a service in the same Linux server, each of them using a different port to allow the access to the web interface.

### 2.1 Scenario Overview



**Fig. 2. 1** Scenario Overview

This scenario showed in Fig. 2. 1 simulates an enterprise that seeks to standardize all the process related to developing.

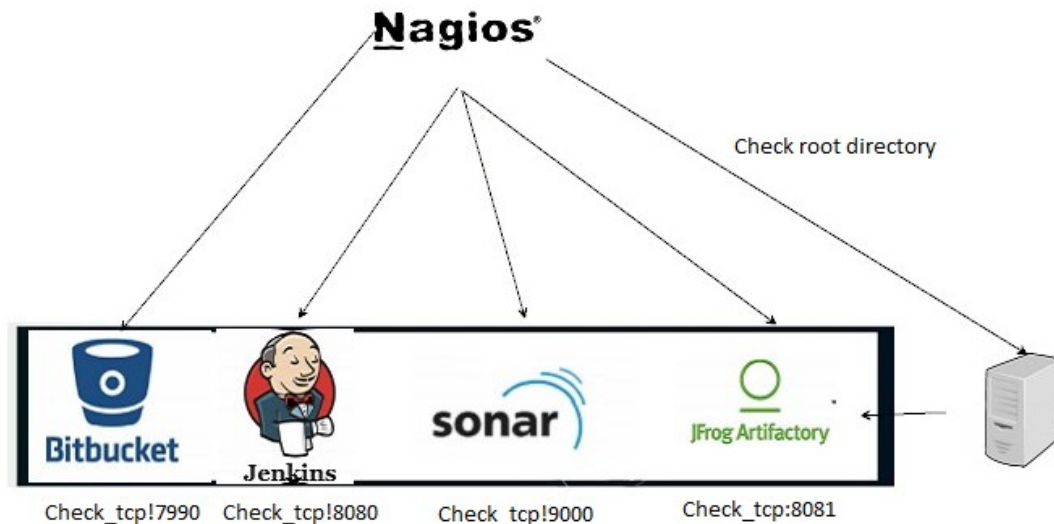
Their main objectives would be to store all the code developed in local server keeping it away from public access as well as storing a history of commits/pushes per developer. Also, the system would be able to tracking all the developing process in an effective manner.

Furthermore, it would be able to automatize all the building process and keep the output generated. Then, use it to create a history of builds to be able to

check what had gone well or had fail in order to know which change has broken the project or generated new bugs.

Nevertheless, analyze the code following a standards previously defined and ease the reading, developing and checking code is another process of the system. Storing all the artifacts/dependencies generated including the own libraries or snapshots of the building state will be located in a local server in order to be able to save bandwidth. This is a critical fact because of in a big corporations were hundreds of employees are using Internet resources simultaneously and to know instantaneously by email if the services deployed are running or have gone down.

In order to test the scenario capabilities different projects were tested. Also, to have variety, a nodejs-example project, java project from DSA subject, a maven-example project, a tunneling project made in python and two android apps had been tested too.



**Fig. 2. 2** Nagios tcp check.

The main idea is to allow the developers to focus their efforts on the important work as developing instead of wasting time in project configuration, sharing code, storing libraries and to store a well documented step-by-step. Histograms of each developer are used to tracking purposes and know the performance of each developer.

As well is vital to know if one of the services is running or down, in other words, improving reacting time, to fix outages before it really damages or slowdown the developing process.

## 2.2 Git and Bitbucket

Bitbucket (Fig. 2. 3) is a web-based hosting service for projects that use either the Mercurial or Git revision control systems. Bitbucket is written in Python using the Django web framework.

Bitbucket can be integrated seamlessly with tracking tools as Jira used in many organizations.



Fig. 2. 3 Bitbucket logo

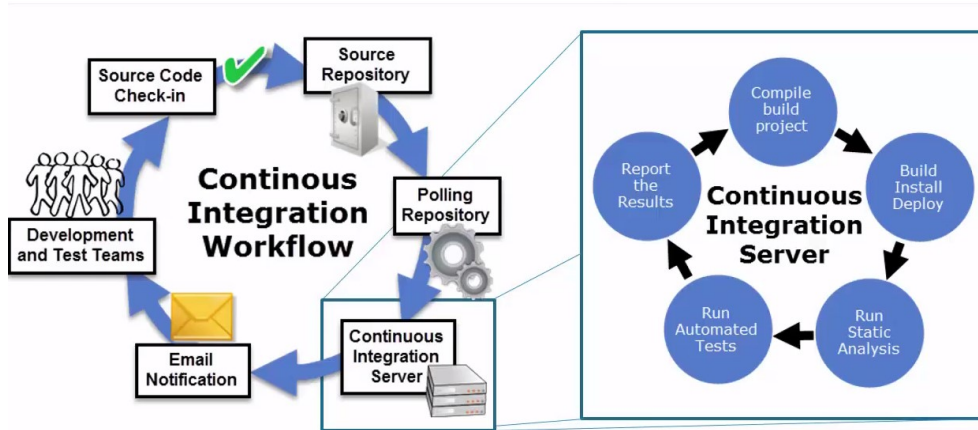
Any programmer worth their salt knows that source control is crucial. The most obvious problem is allowing you to securely store your code in a safe place. Having good source control makes it easier to experiment with new features without worrying about irreparably damaging your program. Source control is something we all should do.

In a lot of organizations you need to keep the projects confidential in order to avoid some leaks or a commercial competitor takes advantage of the project you are developing. With Bitbucket you can have all the private repositories that you want, allowing you to store securely your code.

This software will provide a location to store locally the code that has been tested and to retrieve some data of the developers' cooperation as commits histogram

## 2.3 Jenkins

Jenkins is a Continuous Integration (CI) tool. CI is a process that most developers follow to keep their code base intact. It's mostly a common practice when you work in a group environment. Also, CI system gathers all your code from different developers and makes sure it compiles and builds fine. Basically a CI is the practice of running your tests on a non-developer machine automatically every time someone pushes new code into the source repository.



**Fig. 2. 4** Integration tool diagram

Jenkins is nothing but a man in the middle between your code repository and your build server. It checks for changes on your server every few minutes. If it found them, it gathers them and sends them to your build server.

This has the tremendous advantage of always knowing if all tests work and getting fast feedback. The fast feedback is important so you always know right after you broke the build as for example, when someone introduced changes that made either the compile/build cycle or the tests fail or what you did that failed and how to revert it.

If you only run your tests occasionally the problem is that a lot of code changes may have happened since the last time and it is rather hard to figure out which change introduced the problem. However, when it is run automatically on every push then it is always pretty obvious what and who introduced the problem.

Then developer time is focused in what it matters, increase development speed and code quality is improved. Jenkins can be integrated with a lot of tools like Sonar, Bitbucket, Git, Artifactory and much more. It also provides a lot of plugins to increase its functionality.

This Software is the nexus point or binding point between a lot of software of the scenario, because it binds with Bitbucket, with Sonar and with Artifactory, It makes possible the communication between them.

## 2.4 SonarQube

SonarQube is an open source tool suite to measure and analyze quality of source code. It is written in Java but is able to analyze code in about 20 different programming languages.

Code analysis may be started manually by executing a so-called sonar runner. But SonarQube's full potential is especially revealed when used in combination with continuous integration such as a Jenkins server.



The results of an analysis are shown in a fancy web frontend with green and red lights charts it have an issue lists and an ability to drill down from project level to a single class.

This software is very important for the project because it will analyze all the code of the different projects that have been tested and provide some feedback to allow developers improve their work.

What is code quality?

*A well-written program is a program where the cost of implementing a feature is constant throughout the program's lifetime -- Itay Maman*

There are seven technical axes that should be looked at when doing source code analysis of a project and Sonar is able to support the management of all seven. Non respect of coding standards and best practices:

- Lacking comments in the source code, especially in public APIs.
- Having duplicated lines of code.
- Having complex component or/and a bad distribution of complexity amongst components.
- Having no or low code coverage by unit tests, especially in complex part of the program.
- Leaving potential bugs.
- Having a spaghetti design (package cycles...).

Sonar 7 axes in other words:

- Duplicated code
- Coding standards
- Unit tests
- Complex code
- Potential bugs
- Comments
- Design and architecture

## 2.5 Artifactory

Artifactory is a binary repository manager. We use source control for the source code, Artifactory is version control and more for the binary artifacts (jar/war files, etc). Artifactory is also a place where you can put a shared library so that it is easily accessible in other projects across the enterprise.

This software it is important because it will store all the artifacts previously downloaded, self created artifacts or current build snapshots of the projects. Will allow to the corporation to save bandwidth.



**Fig. 2. 5** Artifactory logo

## 2.6 Nagios



**Fig. 2. 6** Nagios logo

Nagios () monitors your entire IT infrastructure to ensure systems, applications, services, and business processes are functioning properly. In the event of a failure, Nagios can alert technical staff of the problem, allowing them to begin remediation processes before outages affect business processes, end-users, or customers.

This software will allow realizing that something is failing on the system before it can cause major problems and allow developers to fix it before it can cause major damage.

### 2.6.2 NRPE

The NRPE add-on is designed to allow you to execute Nagios plugins on remote Linux/Unix machines.

The main reason for doing this is to allow Nagios to monitor "local" resources (like CPU load, memory usage,etc.) on remote machines.

#### 2.6.2.1 Direct checks

The most straight forward use of the NRPE add-on is to monitor "local" or "private" resources on a remote Linux/Unix Machine. This includes things like CPU load, memory usage, swap usage, current users, disk usage, process states, etc.

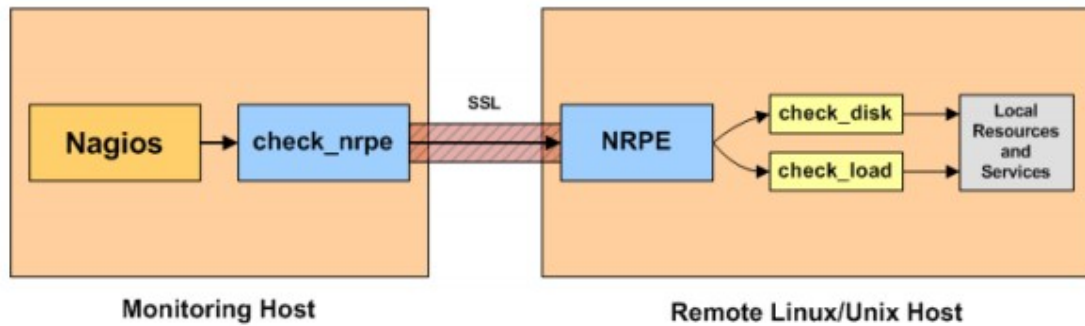


Fig. 2. 7 Direct checks

### 2.6.2.2 Indirect checks

You can also use the NRPE add-on to indirectly check "public" services and resources of remote servers that might not be reachable directly from the monitoring host.

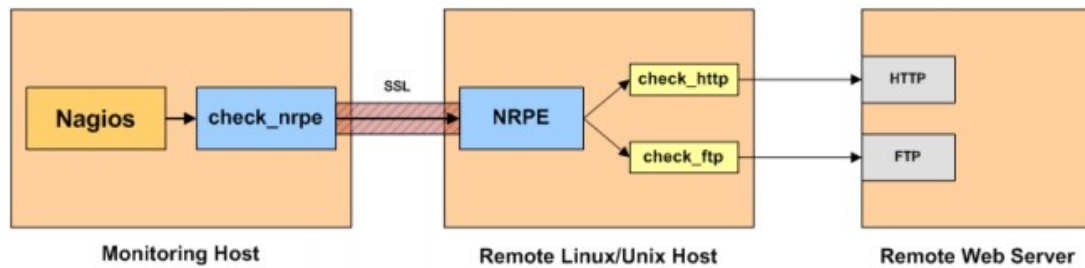


Fig. 2. 8 Indirect checks

## 2.7 Jira

Jira is a proprietary issue tracking product, developed by Atlassian. It provides bug tracking, issue tracking, and project management functions.



Fig. 2. 9 Jira logo

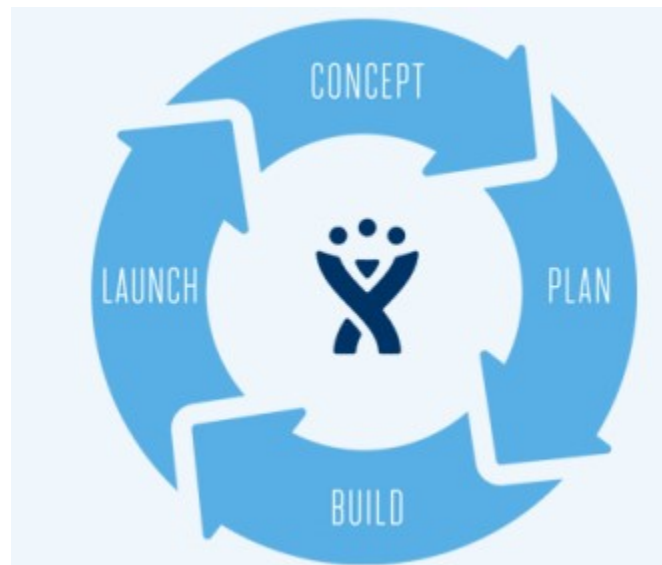
According to Atlassian, over 25,000 customers in 122 countries around the globe use Jira for issue tracking and project management.

Jira is a commercial software product that can be licensed for running on-premises or available as a hosted application. Pricing depends on the maximum number of users.

Jira Software will aid developers to have a better control and organization about the task that are currently being done that is why is very important for this project.

### 2.7.1 Features

- Task Management: The Users can create task with all the information needed as well as adding priority to existing task and assign to others users and keep track on it.
- Workflows: personalize different works to be able to adapt to the company.
- Project plannification and management: Allows to plannify all the work that will be done during a project, assigning or adding priority to different tasks.
- Team Collaboration: it eases the collaboration between teams.



**Fig. 2. 10** Jira concept

As well Jira can be integrate with a lot of software like Github, Microsoft office, pivotal tracker bugzilla, etc.

## 2.8 Required Technologies for using above Software

## 2.8 .1 Apache

Apache (Fig. 2. 11) is the most widely used web server software. Developed and maintained by Apache Software Foundation, Apache is open source software available for free. It runs on 67% of all web servers in the world. It is fast, reliable, and secure. It can be highly customized to meet the needs of many different environments by using extensions and modules.



**Fig. 2. 11** Apache logo

Apache is needed in order to display Nagios web interface and it will run at port 80 in the simulated scenario.

## 2.8 .2 PHP

PHP (Fig. 2. 12) is an acronym for "PHP: Hypertext Preprocessor and is widely-used open source scripting language is free to download and is a popular language.

PHP scripts are executed on the server, It is powerful enough to be at the core of the biggest blogging system on the web (WordPress) and It is deep enough to run the largest social network (Facebook) ,It is also easy enough to be a beginner's first server side language!



**Fig. 2. 12** PHP logo

The reason it is mentioned the Nagios web interface it is developed with PHP

## 2.8 .3 Mysql

MySQL (Fig. 2. 13) is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, used by high profile web properties including Facebook, Twitter, YouTube, Yahoo! and many more.



**Fig. 2. 13** MySQL logo

Oracle drives MySQL innovation, delivering new capabilities to power next generation web, cloud, mobile and embedded applications.

The reason it is mentioned is because it is used to store Sonar and Nagios databases .

#### **2.8 .4 Java**

Java is a programming language and computing platform first released by Sun Microsystems in 1995. There are lots of applications and websites that will not work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

Java allows you to play online games, chat with people around the world, and view images in 3D, just to name a few

Some projects that will be tested in this thesis are programmed in java also Sonarqube and Bitbucket runs on java and without jdk and jre won't start.

## CHAPTER 3. SCENARIO

In this chapter it is explained the configuration, the use and how each piece of software works together as a system and provides to the users a powerful tool to improve their developing. Also it is explained an economic summary and how services are spitted in different ports to access the web interface.

### 3.1 Configuration of the scenario

This scenario use Jenkins, Sonar, Nagios, Artifactory, Jira and Bitbucket as primary software and Mysql, Php, java and apache as prerequisites to be able to use above software. All the software has been installed in an Ubuntu server.

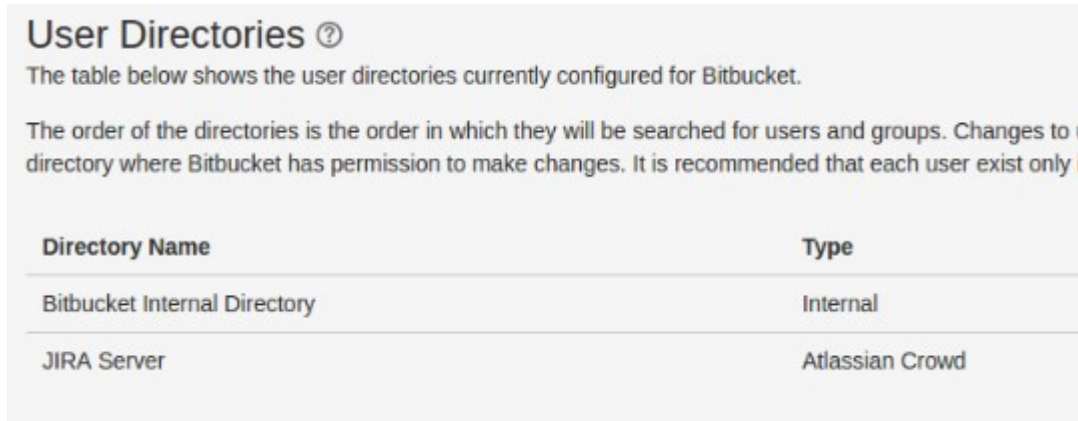
#### 3.1.1 SmartGit, Jira and Bitbucket configuration

SmartGit is a Git client for Windows, Mac and Linux. It is free for non-commercial use, provides a graphical access to Git repositories and can access Subversion repositories. It hides Git's complexity whereby beginners can start quickly and it helps to avoid common pitfalls. Furthermore, it has fine-tuned support for common workflows, making the average Git user more productive and is still powerful enough to provide desired information and functionality to Git experts.

Then, SmartGit is a mature Git client which is on the market since 2009 and has since been improved and fine-tuned during various releases is platform independent. It does look and feel like a native application on Windows, Mac OS X and Linux and offers almost identical functionality on all of these systems. SmartGit integrates with GitHub (Enterprise), BitBucket, GitLab, JIRA and other services. Although, SmartGit will provide to the developers a graphical view to see their collaboration, which are a commit histogram, and a powerful tool to ease merge process.

On the other hand, Jira will help developers to have a planned, ordered and well managed dashboard were all the issues will be drawn there. Keeping all the issues on Jira will aid developers to see their improvements, tracking and changes that they have made during the project. Although, when an app is demanded for a client, sometimes it is not only a mobile application. That is: the project can contain its backend part or front end. Then, they are splitted in different projects in order to keep track on them following scrum methodology. Finally when the project is ready to put it all together, this system allows do merging seamlessly and binding all together.

In order to track the changes by user, all the developer had created a user on Jira or Jenkins and they will be imported from Bitbucket repositories. That means that if a user has permissions in a certain repository, will have as well automatically permissions in the backlog/dashboard of Jira what-ever-the-file-is of related with the same project. As can be seen in Fig. 3. 1 it is possible to assign different roles and permissions to the user.

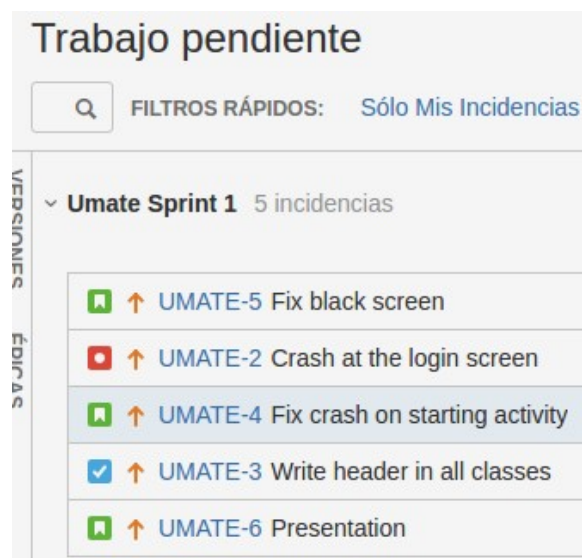


Directory Name	Type
Bitbucket Internal Directory	Internal
JIRA Server	Atlassian Crowd

**Fig. 3. 1** Binding of users between Jira and Bitbucket

Jira will allow developers to create task and assign them to a spring following the scrum methodology, as well developers will be able to assign each other different tasks.

Then, in a real project each team shall meet at least one time at week (1 spring = 1 week more or less) in order to update the project: comment pendent work, plan it, organize it and keep track about what has been done in pervious springs. Using this methodology, there will be always a place where to check when what we want to know what has been done. Fig. 3. 2 shows an example of an weekly meeting where some new tasks are added.

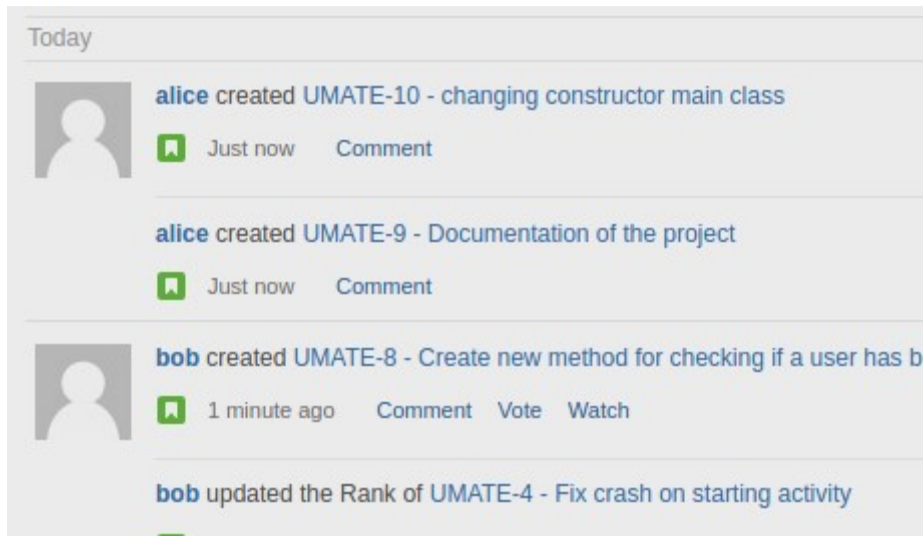


Trabajo pendiente	
Q	FILTROS RÁPIDOS: Sólo Mis Incidencias
VERSIÓN	<ul style="list-style-type: none"> <li>Umate Sprint 1 5 incidencias           <ul style="list-style-type: none"> <li>↑ UDATE-5 Fix black screen</li> <li>↑ UDATE-2 Crash at the login screen</li> <li>↑ UDATE-4 Fix crash on starting activity</li> <li>↑ UDATE-3 Write header in all classes</li> <li>↑ UDATE-6 Presentation</li> </ul> </li> </ul>

**Fig. 3. 2** Pending tasks from the first sprint

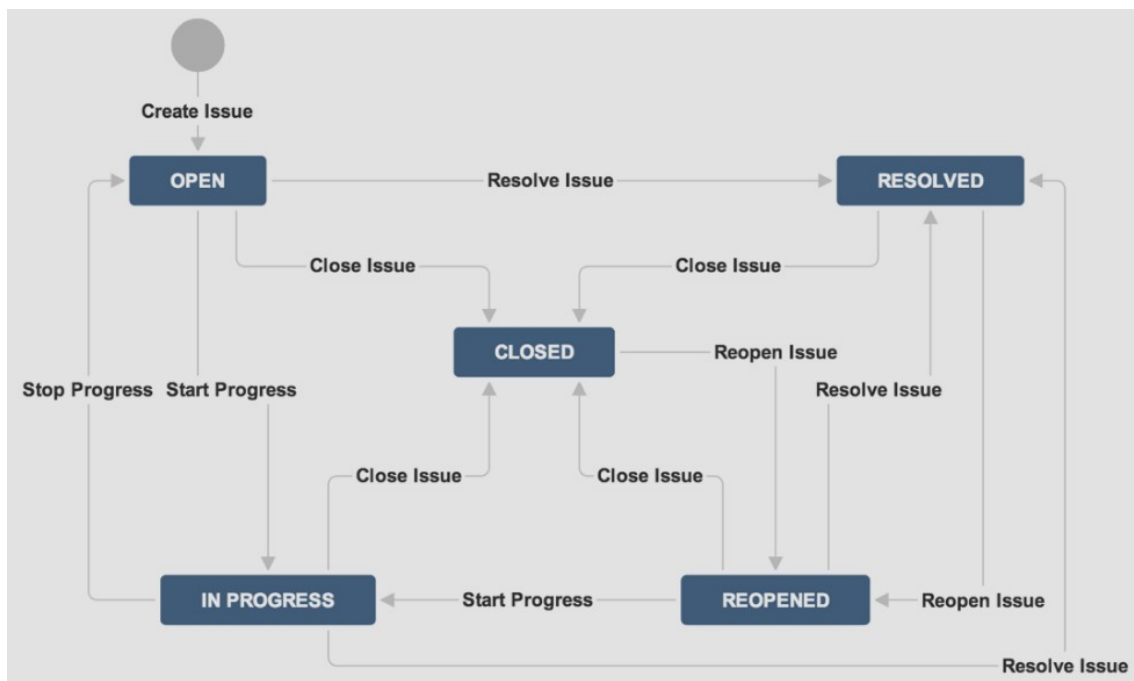


If one developer wants to know what has been the latest things implemented/fixed/done he will only have to check activity panel of Jira. Fig. 3. 3 show and example.



**Fig. 3. 3** Activity panel of Jira

In the Fig. 3. 4 just below it is shown a classical Jira workflow. This workflow will help developers to their jobs. The Jira follows the next steps: in the spring meeting, an issue is created and it will have the state *Open*. Just after the developer started working, he will press *Start* progress and the state will change to *In Progress*. Then, depending on progress of the task (some part of the task is complete), the state will change based on the outcome.



**Fig. 3. 4** Jira workflow

After all the issues have been created and the spring has been started, the developer will check the issues that have assigned and read the description. Thanks to Jira, a link will create a new branch for the project in Bitbucket. Fig. 3.5 shows an example of the new branch created apart from the master one.

### Create branch for UMATE-1

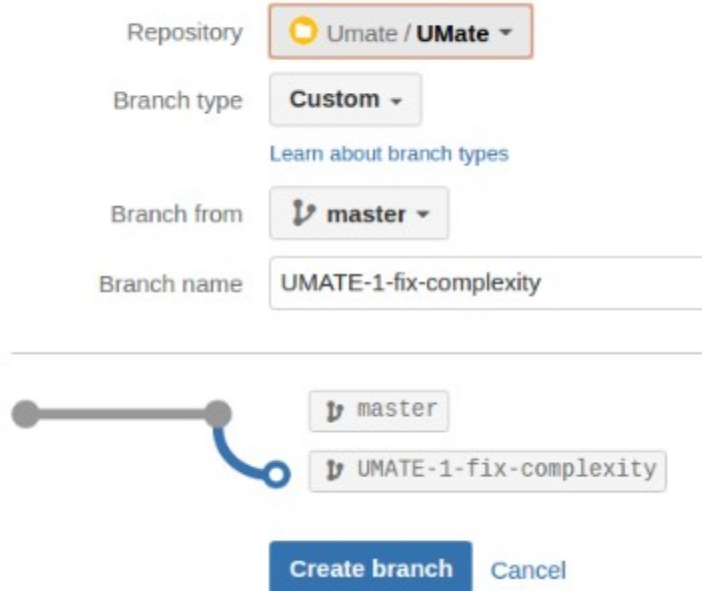


Fig. 3. 5 Bitbucket branches

The developer creates his own branch of the project. Usually coming from the master branch. In the case of fix or develop the issue, a branch is created locally and committed the changes. Then, the developer push to the Bitbucket repository, were the new branch would be created. Finally, when task is finished, the branch would be merged to the master.

Additionally, the merged process can be done manually to avoid inconsistencies of the code or can be done automatically if different files where modified. These actions are made with SmartGit.

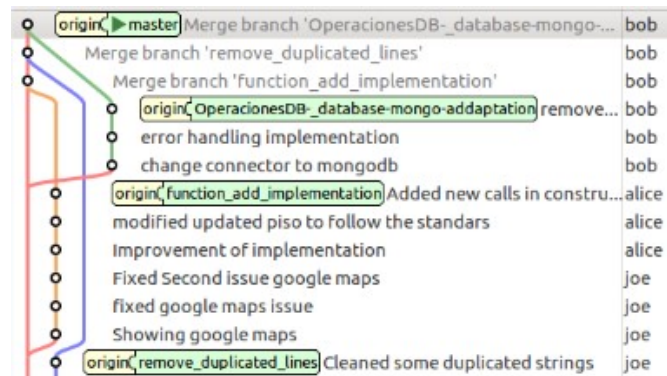








Fig. 3. 6 SmartGirt project control using branches

For showing how it works, the Fig. 3. 6 shows a screenshot about the work progress. The developers, each one of them will work in his own branch and after the work is completed will merge to the master. As shown in the figure, they are working at the same time in different branches without bothering or overlapping themselves. That means if one of them makes a change that could broke the project it will not affect to the work of the others, saving them precious time.

Finally, when each of them had finished working on the branch they can merge with the master branch. Consequently, the master should not be a working branch, and has to be always functional. Jenkins will build from it.

In order to check if the updated code works correctly, the developers could create a new task with the suffix “-wip” which stands for work in progress in order to check if they changes work properly.

Branch	Behind/Ahead	Updated
 master <span style="float: right; border: 1px solid gray; padding: 2px;">DEFAULT BRANCH</span>	12	20 mins ago
 OperacionesDB- <u>database-mongo-addaptation</u> <span style="float: right; border: 1px solid gray; padding: 2px;">BASE BRANCH</span>		22 mins ago
 <u>function_add_implementation</u>	3   6	26 mins ago
 <u>remove_duplicated_lines</u>	3   3	1 hour ago
 <u>coverage_comment</u>	21	4 days ago
 <u>comment_coverage</u>	36	2 hours ago

**Fig. 3. 7** Branches control in Bitbucket

Branches are created locally for each user at the beginning but when they commit and push it to the server, Bitbucket creates automatically the branches in the repository, being accessible for every developer. This process is registered in the current project and has the permission rights. Furthermore, it shows how forwards the development is respect the other branches of the project. In order a user to be able to push to the repository, it should have been granted the write permission; otherwise, they will not be able to push the changes.

Once the code is pushed to the repository a webhook will be launched, triggering an auto build in Jenkins. This will be automatically done sending a json to <http://jenkinsserver:port/webhook/>. Just below, an example of a part of the json is shown. Its important to notice that in order to make the hook work a compatible Jenkins webhook plugin should be added to Bitbucket.

```

{
  "repository":{
    "slug":"umate",
    "id":11,
    "name":"umate",
    "scmId":"git",
    "state":"AVAILABLE",
    "statusMessage":"Available",
    "forkable":true,
    "project":{
      "key":"Umate",
      "id":21,
      "name":"Umate",
      "public":false,
      "type":"NORMAL",
      "isPersonal":false
    }
    ...Continues...
  }
}

```

Although, its very important to highlight the field key because it acts as a binding parameter between Bitbucket, Jenkins and sonar. If this field is the same for these three softwares, the project will be the same, no matter they have different code or they are different projects .

The next image shows hook log file from Jenkins, to see how it reaches successfully and triggers the build of the project.


```

Received commit hook notification for {"scmId":"git","project":{"key":"UMATE","name":"Umate"},"slug":"umate","links":{"self":[{"href":"http://192.168.1.33:7990/projects/UMATE/repos/umate/browse"}]},"ownerName":"UMATE","public":true,"fullName":"UMATE/umate","owner":{"username":"UMATE","displayName":"UMATE"}}

```

**Fig. 3. 8** Jenkins log file

### 3.1.2 Jenkins



S	W	Name ↓	Last Success	Last Failure	Last Duration
		<a href="#">tunneling</a>	27 days - <a href="#">#4</a>	N/A	6 sec
		<a href="#">Umate</a>	1 mo 6 days - <a href="#">#52</a>	1 mo 6 days - <a href="#">#48</a>	8.2 sec
		<a href="#">SampleNodeServer</a>	1 mo 9 days - <a href="#">#15</a>	1 mo 9 days - <a href="#">#20</a>	0.71 sec

**Fig. 3. 9** Jenkins direction access

### 3.2.2.1 Plugins Required and software

In order to make Jenkins work with the following architecture these plugins are needed:

- Nodejs Plugin
- Mercurial Plugin
- Bitbucket Plugin
- Bitbucket pullrequest Plugin
- Artifactory Plugin
- Deploy to a container Plugin
- Pipeline
- Repository connector
- QualityGates
- SonarQube Scanner

This software would be required and installed on the server:

- Maven
- Nodejs
- Mysql
- Java jre8 and jdk8

### 3.1.2.2 Jenkins and artifactory configuration

Jenkins will have a list of all the projects coming from Bitbucket or different repository managers as Github as shown in the Fig. 3. 10. Each time the build has been triggered, Jenkins will clone the code, update the changes to make a new updated build.

Jenkins can connect with different tools as sonar and Artifactory to speed up and test the development.

```
> git config remote.origin.url http://192.168.1.33:7990/scm/mav/project-examples.git # timeout=10
Fetching upstream changes from http://192.168.1.33:7990/scm/mav/project-examples.git
> git --version # timeout=10
> git fetch --tags --progress http://192.168.1.33:7990/scm/mav/project-examples.git +refs/heads/*:refs/remotes/origin/*
~ git rev-parse refs/remotes/origin/master^{commit} # timeout=10
```

**Fig. 3. 10** Jenkins connections with different tools

Jenkins will start parsing the pom.xml. This last process will only happen if from configuration menu of the project you set it on the path file and will send to Artifactory the dependencies that have to retrieve. If Artifactory does not have the dependencies it will start downloading them. These dependencies downloaded will be stored in the local server, to speed up the process in posterior builds. Fig. 3. 11 show an example.

```
- jcenter downloading https://jcenter.bintray.com/org/eclipse/jdt/core/compiler/ecj/4.4.2/ecj-4.4.2.pom 2.02 KB
Indexing archive: jcenter-cache:org/codehaus/mojo/appassembler-maven-plugin/1.1.1/appassembler-maven-plugin-1.1.1.jar
- jcenter downloaded https://jcenter.bintray.com/org/eclipse/jdt/core/compiler/ecj/4.4.2/ecj-4.4.2.pom 2.02 KB at
```

**Fig. 3. 11** Artifactory download library thanks to the parsed pom.xml

Generally the first building of each project takes twice or more time than next builds due to all dependencies gathered. The time will increase depending on the quantity of dependencies in the pom.xml file. After downloading them, Artifactory will store it and Jenkins will build and install each dependency.

```
[INFO] Simple Multi Modules Build ..... SUCCESS [ 0.552 s]
[INFO] Multi 1 ..... SUCCESS [ 5.134 s]
[INFO] Multi 2 ..... SUCCESS [ 0.681 s]
[INFO] Multi 3 ..... SUCCESS [ 0.973 s]
[INFO] .....
[INFO] BUILD SUCCESS
```

**Fig. 3. 12** Creation of new jar files

As shown in the Fig. 3. 12 above, during the building process Jenkins creates some jar, war or js files, depending which language it is used. These are snapshots of the current state of the project and will be stored in a local Artifactory repository to keep track of all the work. As well as snapshots, private libraries self-developed for the corporation will be stored for a future use. In other words, all the corporation that have access to the Artifactory repository will be able to download and integrate easily only by using the Artifactory server. With this way, we avoid old type interchanges of libraries that led it to mistakes as different library version or missing dependencies.

In some corporations with a hundreds of workers this is really important due to it minimize the use of the precious Internet resources. Furthermore it speeds up the building process due to all dependencies are already stored as shown in the below table.

In the tables below it is shown the duration of two projects that use Artifactory and is clearly faster using Artifactory. The reason is because the dependencies are already in the local server, and there is no need to gather it from Internet .

Project : maven-example

	Starting time	Ending time	Total time
With artifactory	12:09:33	12:09:41	8 s
Without artifactory	16:57:43	16:57:58	15s

**Table 3. 1** Maven example Building time

Project: Umate

	Starting time	Ending time	Total time
With artifactory	11:07:33	11:07:52	19 s
Without artifactory	12:08:24	12:08:49	25s

**Table 3. 2** Umate example Building time

In the figure below (Fig. 3. 13) is shown how Jenkins deploys a snapshot in a jar format to the Artifactory repository after a successfully building it. When we realize that our project is ready to release, we only will have to retrieve the snapshot from the server and to deploy it.

```

Artifactory Build Info Recorder: Saving Build Info to '/var/lib/jenkins/workspace/maven-example/maven-example/target/build-info.json'
Deploying artifact: http://192.168.1.33:8081/artifactory/libs-snapshot-local/org/jfrog/test/multi2/3.7-SNAPSHOT/multi2-3.7-SNAPSHOT.jar
Deploying artifact: http://192.168.1.33:8081/artifactory/libs-snapshot-local/org/jfrog/test/multi2/3.7-SNAPSHOT/multi2-3.7-SNAPSHOT.pom
Deploying artifact: http://192.168.1.33:8081/artifactory/libs-snapshot-local/org/jfrog/test/multi1/3.7-SNAPSHOT/multi1-3.7-SNAPSHOT-tests.jar
Deploying artifact: http://192.168.1.33:8081/artifactory/libs-snapshot-local/org/jfrog/test/multi1/3.7-SNAPSHOT/multi1-3.7-SNAPSHOT-sources.jar
Deploying artifact: http://192.168.1.33:8081/artifactory/libs-snapshot-local/org/jfrog/test/multi1/3.7-SNAPSHOT/multi1-3.7-SNAPSHOT.jar

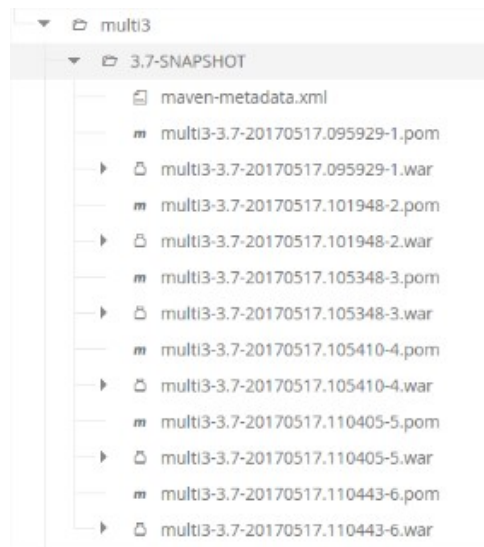
```

**Fig. 3. 13** Snapshot from Jenkins

As said it before and shown in Fig. 3. 14 and Fig. 3. 15 below, for each build and snapshot will be stored.

Module ID	Number Of Arti...	Number Of Dep...
org.jfrog.test:multi1:6.12-SNAPSHOT	4	13
org.jfrog.test:multi2:6.12-SNAPSHOT	2	1
org.jfrog.test:multi3:6.12-SNAPSHOT	2	17
org.jfrog.test:multi:6.12-SNAPSHOT	1	0

**Fig. 3. 14** Snapshot with different number of artifacts and dependencies



**Fig. 3. 15** Snapshots tree

### 3.1.2.3 Jenkins configuration files

For each task, Jenkins will create a configuration file stored in `/var/lib/Jenkins/job/PROJECT_NAME` in a form of an xml file.

This file will contain each step that will be executed during the build. I will show separated parts of Umate project file in order to summarize the most important lines. It is important to keep in mind that this file will change depending on the steps configured or depending on the language.

Bitbucket Connection:

```
<scm class="hudson.plugins.git.GitSCM" plugin="git@3.3.0">
  <configVersion>2</configVersion>
  <userRemoteConfigs>
    <hudson.plugins.git.UserRemoteConfig>
      <url>http://192.168.1.33:7990/scm/umate/umate.git</url>
    </hudson.plugins.git.UserRemoteConfig>
  </userRemoteConfigs>
  <branches>
    <hudson.plugins.git.BranchSpec>
      <name>*/master</name>
    </hudson.plugins.git.BranchSpec>
  </branches>

  <doGenerateSubmoduleConfigurations>>false</doGenerateSubmoduleConfigurations>
  <submoduleCfg class="list"/>
  <extensions/>
</scm>
```

It connects to the Bitbucket repository by the next url:

```
<url>http://192.168.1.33:7990/scm/umate/umate.git</url>
```

And retrieves the master branch. It reads the Pom and tries to install the dependencies stored in it:

```
<concurrentBuild>>false</concurrentBuild>
<rootModule>
  <groupId>EAProject</groupId>
  <artifactId>UMate</artifactId>
</rootModule>

<rootPOM>://192.168.1.33:7990/scm/umate/umate.git/UMate/pom.xml</rootPOM>
<goals>install</goals>
```

Connects to Artifactory and retrieve dependencies from there instead of Internet:

```
<publishers>
```



```
<org.jfrog.hudson.ArtifactoryRedeployPublisher plugin="artifactory@2.10.4">
  <eventIfUnstable>>false</eventIfUnstable>
  <details>
    <artifactoryName>Artifactoryserver</artifactoryName>
    <artifactoryUrl>http://192.168.1.33:8081/artifactory</artifactoryUrl>
    <deployReleaseRepository>
      <keyFromText></keyFromText>
      <keyFromSelect>libs-release-local</keyFromSelect>
    </deployReleaseRepository>
  </details>
</org.jfrog.hudson.ArtifactoryRedeployPublisher>
```

It connects to the sonar-plugin:

```
<deployArtifacts>true</deployArtifacts>
  <resolveSnapshotRepository>
    <keyFromText></keyFromText>
    <keyFromSelect>libs-snapshot-local</keyFromSelect>
    <dynamicMode>>false</dynamicMode>
  </resolveSnapshotRepository>
</deployArtifacts>
```

It deploys Snapshots of the current state of the project (for example a .war or .jar) to the Artifactory server:

```
<builders>
  <hudson.plugins.sonar.SonarRunnerBuilder plugin="sonar@2.6.1">
    <project></project>
    <properties></properties>
    <javaOpts></javaOpts>
    <additionalArguments></additionalArguments>
    <jdk>(Inherit From Job)</jdk>
    <task></task>
  </hudson.plugins.sonar.SonarRunnerBuilder>
</builders>
```

### 3.1.3 Sonar

#### 3.1.3.1 Metrics

A metric is a calculation between two measures. Then, metrics are a method of measuring something, or the results obtained from an analysis. Also, it is important to distinguish measurement concept that is the action of measuring something or the size, length, or amount of something.

##### 3.1.3.1.1 Complexity

It is the complexity calculated based on the number of paths through the code. Whenever the control flow of a function splits, the complexity counter gets incremented by one. Each function has a minimum complexity of 1. This calculation varies slightly by language because keywords and functionalities do.

It will be explained in the case of Java and Node.

In the case of java:

- Keywords and operators incrementing the complexity: if, for, while, case, &&, ||, ?.
- Else, default, and finally keywords do not increment the complexity.
- A simple method with a switch statement and a huge block of case statements can have a surprisingly high complexity value (still it has the same value when converting a switch block to an equivalent sequence of if statements).

The following method has a complexity of 5:

```
public void process(Car myCar){ // +1
    if(myCar.isNotMine()){ // +1
        return;
    }
    car.paint("red");
    car.changeWheel();
    while(car.hasGazol() // +1
        && car.getDriver().isNotStressed()){ // +1
        car.drive();
    }
    return;
}
```

For JavaScript/node

- Complexity is incremented by one for each:
  - Function (i.e non-abstract and non-anonymous constructors, functions, procedures or methods)
  - if statement
  - short-circuit (AKA lazy) logical conjunction (&&)
  - short-circuit (AKA lazy) logical disjunction (||)
  - ternary conditional expressions
  - loop
  - case clauses of a switch statement
  - go to statement (only for PHP)

Complexity have as lower as possible, lowering the number implies faster code due less paths in the code.

### 3.1.3.1.2 Documentation

The documentation included the number of lines containing either comment or commented-out code.

For example:

/**	+0 => empty comment line
*	+0 => empty comment line
* This is my documentation	+1 => significant comment
* although I don't	+1 => significant comment
* have much	+1 => significant comment
* to say	+1 => significant comment
*/	+0 => empty comment line

The density of commented lines will be calculated using the next formula:

$$\text{Density of comment lines} = \frac{\text{Comment lines}}{(\text{Lines of code} + \text{Coment lines}) * 100}$$

### 3.1.3.1.3 Severity

The next table show how Severity will be trait for Sonar:

Severity	Description
Blocker	Operational/security risk: This issue might make the whole application unstable in production. Ex: calling garbage collector, not closing a socket, etc.
Critical	Operational/security risk: This issue might lead to an unexpected behavior in production without impacting the integrity of the whole application. Ex: NullPointerException, badly caught exceptions, lack of unit tests, etc.
Major	This issue might have a substantial impact on productivity. Ex: too complex methods, package cycles, etc.
Minor	This issue might have a potential and

	minor impact on productivity. Ex: naming conventions, Finalizer does nothing but call superclass finalizer, etc.
--	--

**Table 3. 3** Severity**3.1.3.1.4 Reliability**

The next table show how reliability will be tag for Sonar:

Reliability Rating	Number of bugs
A	0 Bugs
B	1 Minor Bug
C	1 Major Bug
D	1 Critical Bug
E	1 Blocker Bug

**Table 3. 4** Reliability

Reliability remediation effort is the effort to fix all bug issues. The measure is stored in minutes in the DB.

**3.1.3.1.5 Security**

The next table show how security will be tag for Sonar:

Security Rating	Number of vulnerabilities
A	0 Vulnerabilities
B	1 Minor Vulnerability
C	1 Major Vulnerability
D	1 Critical Vulnerability
E	1 Blocker Vulnerability

**Table 3. 5** Security**3.1.3.1.6 Maintainability**

The next table show how Maintainability will be tag for Sonar:

Security Rating	Number of vulnerabilities
A	0 Vulnerabilities
B	1 Minor Vulnerability
C	1 Major Vulnerability
D	1 Critical Vulnerability
E	1 Blocker Vulnerability

**Table 3. 6** Maintainability

### 3.1.4 Jenkins and SonarQube configuration

For Sonar to work on Jenkins it is mandatory the configuration of the file sonar-scanner.properties. Otherwise, sonar is not able to analyze the code because it is not able to bind the Jenkins project with the code. Also it is not able to find it because the path it is not set. Then, it is needed as well to install sonar plugin on Jenkins. This file should be place in the root directory of each project, and will be used to sonar and Jenkins to two ways bind the project.

Sample of sonar-scanner.properties:

```
sonar.projectKey= <PROJECT KEY>
sonar.projectName=<PROJECT_NAME>
sonar.projectVersion=1.0

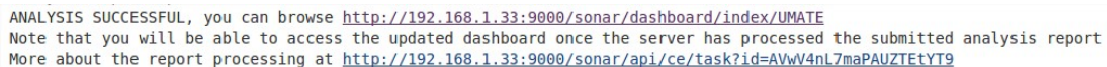
sonar.sourceEncoding=UTF-8
sonar.sources=.<SOURCE_CODE>
sonar.exclusions=<FILES_EXCLUDED>
sonar.java.binaries=bin/classes

#in case of android project , how will be called the report >
sonar.android.lint.report=lint-report.xml

sonar.branch= master
```

I picked Umate project to show how sonar works. The reason to pick this project it is because it was done during DSA classes and it contains a lot of bad habits, code repetition and missing comments. That will make the project unable to pass the quality gates of sonar, but at same time will give us a huge output with all the bugs, vulnerabilities and errors that does not complaint with a good a habit of developing.

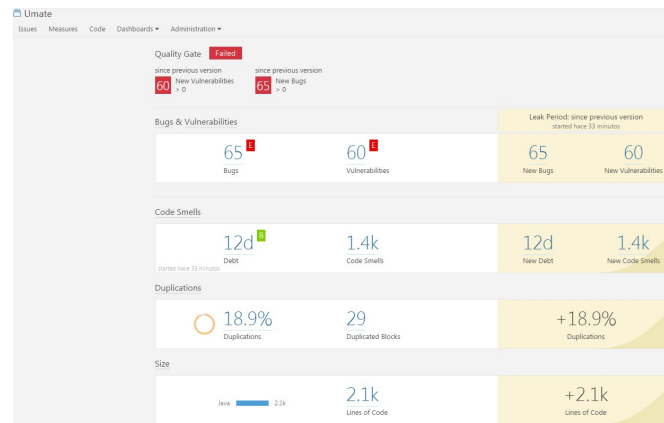
If the analysis is successful, in other works, sonar could read and analyze all the code a link would be shown from Jenkins. From this link, we could access to the sonar web page, where all the results are presented in a dashboard. If the sonar finds more errors than allowed will mark the project as failure and will assign a letter as a rate from A to F, being F the worth and A perfect.



```
ANALYSIS SUCCESSFUL, you can browse http://192.168.1.33:9000/sonar/dashboard/index/UMATE
Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
More about the report processing at http://192.168.1.33:9000/sonar/api/ce/task?id=AVwV4nL7maPAUZTEtYT9
```

**Fig 3.16b.** Jenkins to Sonar link

In the dashboard showed in the Fig. 3. 16 below we can see and example of how Sonar will show us a view with our possible bugs, the quantity of code lines, how much coverage of comments do we have and how to fix them.



**Fig. 3. 16** Umate Dashboard Page

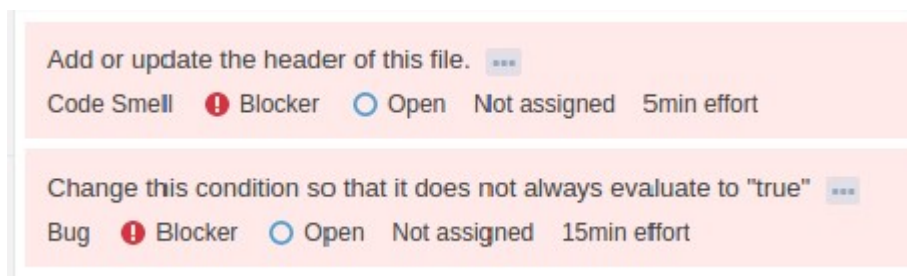
The fixing clues that sonar will give will depend on which rules we have previously defined or activated. It is possible to have a huge set of rules depending on the situation or depending on the project. In other words, you can activate or deactivate the rules that you want providing a flexible way to work and change standard habits.

You can define which will be the conditions called quality gates in order to pass the build using a formulary as below. Blocker, Critical and Major issues must be set to 0, the reason will be explained latter. In any case we don't want our application to go out the market with continuous crashes or a big vulnerability. This kind of issues can broke the trust that consumers or clients will have on us. In the case of minor issues we can be more permissive as I will explain later.

Blocker issues	Value	is greater than	0
Critical issues	Value	is greater than	0
Coverage on new code	Δ since previous version	is less than	80

**Fig. 3. 17** Adding state of the issue to Blocker, Critical and Major

The main reason of this election is due to the features of each issue, a blocker issue implies an application crash that must not happen in any commercial application. This crash can cost money or reputation of the enterprise. For example, in Umate project are cataloged as blockers the next issues (Fig. 3. 18):



**Fig. 3. 18** Blocker issues

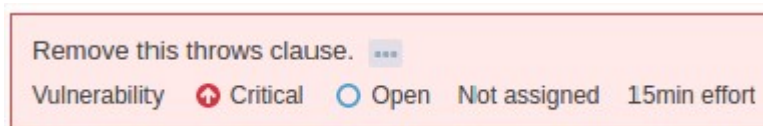
In Fig. 3. 18 you can see a blocker issue detected in PisoResource.java, if you

click on “Change this condition so that it does not always evaluate to "true"”. It will browse you until the line of code where the error is found and you will get some tips and examples to how to fix it, for example in this case will be:

```
private void compute(int foo) {  
    if (foo == 4) {  
        doSomething();  
        // We know foo is equal to 4 at this point, so the next condition is always false  
        if (foo > 4) {...}  
        ...  
    }  
    ...  
}
```

Another example of a blocker issue is the “Add or update the header of this file” which you get the tip “Each source file should start with a header stating file ownership and the license, which must be used to distribute the application.” That is because the main function of sonar is to standardize and ease the process of developing. In the case of several developers, they are always creating code is to remember them with a header what is the purpose of each class. That can be as well a method of security. Imagine that you are using a commercial class/library, there is no license at the start of the file, and you think you can use it as you want in your own project. That must not happen.

Null pointer Exceptions will be as well categorized as blockers issues because it can crash the application. For critical issues we have the followings errors:



**Fig. 3. 19 Critical issues**

In this case the error was caused because the declaration of the class was “public static void main (String []args) throws IOException”, for sonar that is a critical issue because there is no reason for a main method to throw anything. After all, *what's going to catch it?*

Instead, the method should itself gracefully handle any exceptions that may bubble up to it, attach as much contextual information as possible, and perform whatever logging or user communication is necessary. For that reason, sonar shows you the next tip:

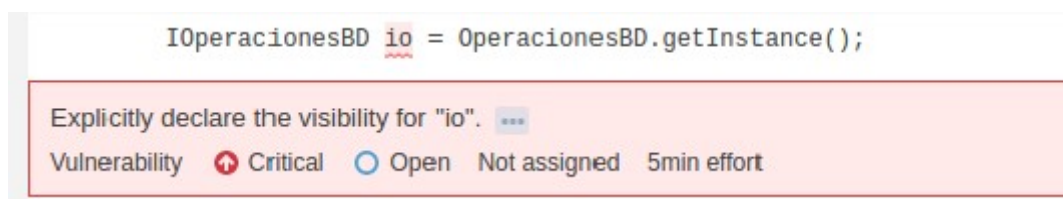
Compliant Solution

```

public static void main(String args[]) {
    try {
        doSomething();
    } catch (Throwable t) {
        log.error(t);
        System.exit(1); // Default exit code, 0, indicates success. Non-zero value
        means failure.
    }
}

```

Another example of critical issue is the following one (Fig. 3. 20):



**Fig. 3. 20** Critical issue

This will be the tips given by sonar, as you can see with the example bellow is that the variable was not set as private and could be accessible from outside. This can be a potentially leave it open to unexpected modification by other classes.

#### Noncompliant Code Sample

```

class Ball {
    String color="red"; // Noncompliant
}
enum A {
    B;
    int a;
}
Compliant Solution

class Ball {
    private String color="red"; // Compliant
}
enum A {
    B;
    private int a;
}

```

For Majors issues we have for example:



- “Method has 12 parameters, which is greater than 7 authorized”.
- “Add a private constructor to hide the implicit public one”.
- “2 duplicated blocks of code must be removed”.
- “This file has 1,072 lines, which is greater than 1,000 authorized. Split it into smaller files”.

For Minors issues we have for example:

- Document this public method.
- Add a new line at the end of this file.
- Assign this magic number 800.0 to a well-named constant, and use the constant instead.
- Document this public class.
- Remove this unused import 'EAPProject.UMate.transaction.PisoT'.

Depending if it complains with the quality gates the build will be marked as failure or as successful. In Fig. 3. 21 it is possible to observe that at the end of the line it is marked as failure.

```
16:17:58 PostBuild-Step: Quality Gates plugin build passed: FALSE
16:17:58 Build step 'Quality Gates' marked build as failure
16:17:58 [ERROR] Scanning build for known causes
```

**Fig. 3. 21** Result obtained based on the quality gates

Continuing with the description, a case of Tunneling project made in python. Its complexity is 168 but the complexity is so high because there is a lot of bash scripts gathered in three python files, which gathers all the functionalities of the project, which means a lot of operators. This operator increases the complexity function.

fabfilewifi.py	61
fabfile.py	61
plot.py	46

**Fig. 3. 22** Complexity found

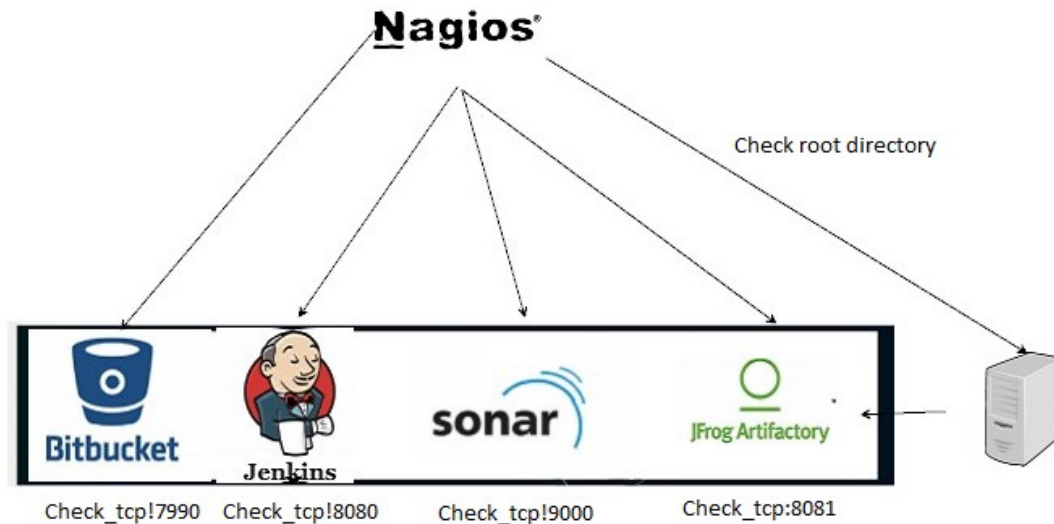


**Fig. 3. 23** Bugs and Vulnerabilities obtained

As you can see in Fig. 3. 22 and Fig. 3. 23, there are 5 bugs but it comes from a deprecation of a previous version of python, and it has some minor issues/vulnerabilities related to some addresses that are not configurable, , in other words , hard coded or a missing/surplus comments .

There is no support for bash in sonar due to that I only could analyze python code.

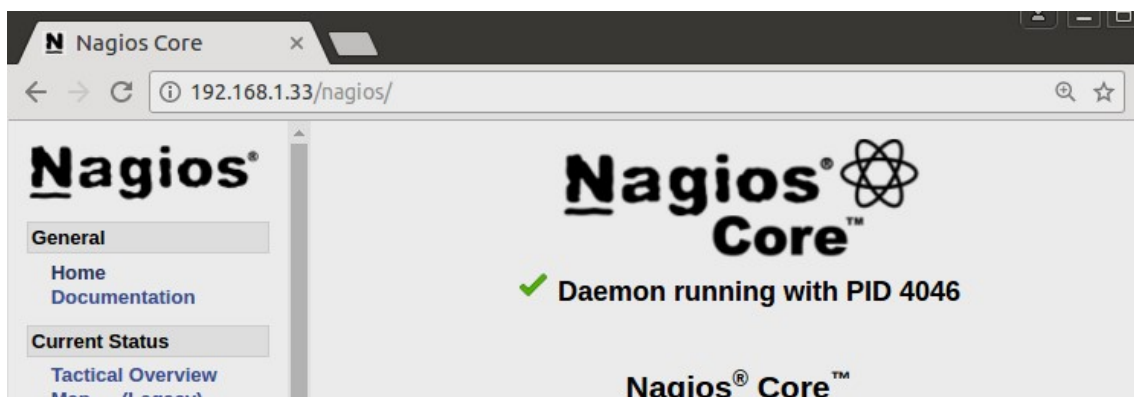
### 3.2 Nagios



**Fig. 3. 24** Nagios communication

First thing to make Nagios (Fig. 3. 24) work in this scenario is to open the firewall for the nrpe add-on in order to be able to monitor the services. The file /etc/iptables.conf should be modified with the following line: \_

```
iptables -I RH-Firewall-1-INPUT -p tcp -m tcp --dport 5666 -j ACCEPT
```



**Fig. 3. 25** Nagios access web

The main reason to use Nagios is to allow to the corporation to manage in a real time scenario, which services are currently running or disabled, and be able to react before the outage has been notified for the workers.

Nagios configuration it is located in a different configuration files, there are 4 types:

- Main configuration File  
Nagios.cfg →Contains all the directives and location of other files(users, delays ,resources...)
- Resource File  
For defining where scripts are, defining services
- Object Definition File  
Definition of other hosts or things that have to be monitored, who has to be mailed..

Each of the configuration files are added to the Main configuration File, then the daemon checks the main configuration file to know which others configuration files has to load.

In the case of this scenery, here is a summary of the hosts.cfg file.

```
define host {
  Use          server      ; Name of the template where gets his properties
  Host_name    Tesis       ;Name of the host
  Alias        Tesis Server ;Name
  Address      192.168.1.33 ;network address.
}
```

For Nagios to know what have to check is mandatory. In order to define “the mandatory services” , just below is presented a case. In the case of Jenkins, Artifactory , Bitbucket and Sonar, as they are services that use http the server will establish TCP like connections. Then, for monitoring each of them will be necessary to change the port in the check-command as below.

```
define service{
  use          generic-service
  host_name    localhost ;hostgroup can be used instead as well
  service_description      Artifactory
  check_command      check_tcp!8081 ;          the          sintaxis          is
  COMMAND!PORT
  notifications_enabled      1          ; 1 means enabled
}
```

In order to allow Nagios to notify to the administrator, Nagios has to know an e-mail direction. This should be defined in the file contacts.cfg.

```
define contact{
    contact_name      nagiosadmin      ; Short name of user
    use                generic-contact  ; Inherit default values from
generic-contact template (defined above)
    alias              Nagios Admin    ; Full name of user

    email              kcxarokkc@gmail.com
    ;
}
```

The last file that will be mentioned is commands.cfg where each of the commands will be defined.

```
define command{
    command_name      check_tcp
    command_line      $USER1$/check_tcp -H $HOSTADDRESS$ -p $ARG1$
$ARG2$
}
```

The line command\_line is the script that will be executed with the variables before \$. It is possible to add your own script to personalize what you want.

After setting up all the services as explained before, Nagios will show us a web page like the image below. Fig. 3. 26 show the table that appeared in the web.

Service	Status	Last Check	Duration	Attempt
Artifactory	OK	06-22-2017 19:20:35	0d 0h 2m 6s	1/3
Bitbucket	OK	06-22-2017 19:21:08	0d 0h 1m 33s	1/3
HTTP	OK	06-22-2017 19:18:48	5d 23h 56m 57s	1/4
Jenkins	OK	06-22-2017 19:19:22	5d 23h 34m 17s	1/3
PING	OK	06-22-2017 19:19:55	5d 23h 56m 19s	1/4
Root Partition	OK	06-22-2017 19:20:28	5d 23h 53m 24s	1/4
SSH	OK	06-22-2017 19:21:02	5d 23h 53m 22s	1/4
Sonar	CRITICAL	06-22-2017 19:16:30	5d 23h 33m 44s	3/3
Swap Usage	OK	06-22-2017 19:21:27	5d 23h 52m 24s	1/4
Total Processes	OK	06-22-2017 19:19:58	5d 23h 56m 42s	1/4

**Fig. 3. 26** Nagios tracking services table

As you can see each of the services, has been configured and a intentionally fail has been introduced to the Sonar service, in order to produce a mail error:

Subject: **\*\*PROBLEM** host Alert: Sonar is DOWN **\*\***

To: [kcxarokkc@gmail.com](mailto:kcxarokkc@gmail.com)

---

\*\*\*\*\* Nagios \*\*\*\*\*

Notification Type : PROBLEM  
 Host: 192.168.1.33  
 State: DOWN  
 Adress: 192.168.1.33  
 Info: CRITICAL – Host Unreachable (192.168.1.33)

.....

When the Problem will be fixed a RECOVERY email will be send it

The screenshot displays the Nagios service status for 'Sonar'. The current status is 'CRITICAL' (for 5d 23h 46m 9s) and has been acknowledged. The status information indicates a connection refused to address 127.0.0.1 and port 9000. Performance data shows 3/3 attempts in a HARD state, with the last check time being 06-22-2017 19:26:30. The check type is ACTIVE, and the latency/duration is 0.000 / 0.002 seconds. The next scheduled check is at 06-22-2017 19:36:30. The last state change was on 06-16-2017 19:48:57, and the last notification was on 06-22-2017 18:36:30 (notification 2). The service is not flapping (4.80% state change) and is not in scheduled downtime. The last update was on 06-22-2017 19:35:02 (0d 0h 0m 4s ago). All monitoring features (Active Checks, Passive Checks, Obsessing, Notifications, Event Handler, Flap Detection) are enabled.

**Fig. 3. 27** Nagios message when a critical issue occurred (related with Sonar)

Nagios introduces as well two different states soft and hard states. In order to prevent false alarms from transient problems, Nagios allows you to define how many times a service or host should be (re)checked before it is considered to have a "real" problem. This is controlled by the `max_check_attempts` option in the host and service definitions. Understanding how hosts and services are (re)checked in order to determine if a real problem exists is important in understanding how state types work.

### 3.2 .1 Soft States

Soft states occur in the following situations...

- When a service or host check results in a non-OK or non-UP state and the service check has not yet been (re)checked the number of times specified by the `max_check_attempts` directive in the service or host definition. This is called a soft error. When a service or host recovers from a soft error. This is considered a soft recovery.
- The following things occur when hosts or services experience SOFT state changes:
  - The SOFT state is logged.
  - Event handlers are executed to handle the SOFT state.

SOFT states are only logged if you enabled the `log_service_retries` or `log_host_retries` options in your main configuration file.

The only important thing that really happens during a soft state is the execution of event handlers. Using event handlers can be particularly useful if you want to try and proactively fix a problem before it turns into a HARD state.

The `$HOSTSTATETYPE$` or `$SERVICESTATETYPE$` macros will have a value of "SOFT" when event handlers are executed, which allows your event handler scripts to know when they should take corrective action. More information on event handlers can be found [here](#).

### 3.2.2 Hard States

Hard states occur for hosts and services in the following situations:

- When a host or service check results in a non-UP or non-OK state and it has been (re)checked the number of times specified by the `max_check_attempts` option in the host or service definition. This is a hard error state.
- When a host or service transitions from one hard error state to another error state (e.g. WARNING to CRITICAL).
- When a service check results in a non-OK state and its corresponding host is either DOWN or UNREACHABLE.
- When a host or service recovers from a hard error state. This is considered to be a hard recovery.
- When a passive host check is received. Passive host checks are treated as HARD unless the `passive_host_checks_are_soft` option is enabled.

Then, the following things occur when hosts or services experience HARD state changes:

- The HARD state is logged.
- Event handlers are executed to handle the HARD state.
- Contacts are notified of the host or service problem or recovery.

### 3.2 .3 Monitored Services

#### 3.2.3.1 Services

Artifactory, Bitbucket, Jenkins and sonar are the services that we want to control, if anything fails will change to hard state and be mailed in three minutes, one minute for each check.

Also, the service http is checked to know if apache is running following the next steps:

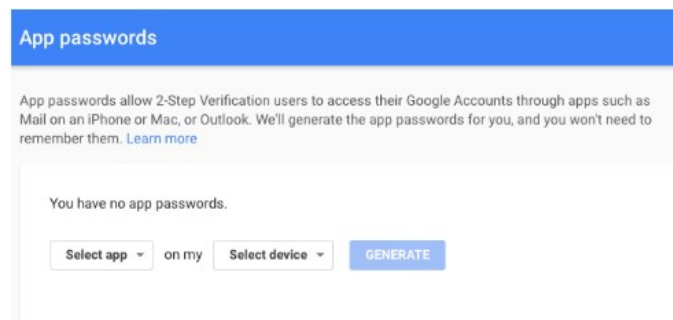
- Service ping is used to check if the host keeps alive.
- SSH service is checked to know if it is possible to login from outside.
- Swap and root partition are checked because is for some reason, a log file growing to much or a disk saturation, some services can stop working because there wont be enough space to write on the disk.

### 3.2.3.2 Post-fix configuration

Postfix is a free and open-source mail transfer agent (MTA) that routes and delivers electronic mail, intended as an alternative to Sendmail MTA.

After install Postfix and libsasl2-modules, it is mandatory to configure gmail. Because it use a Two-Factor Authentication, Gmail will refuse a email coming from a postfix software, because postfix doesn't provide the second step of authentication.

You can configure your Gmail account to accept it, but it will be necessary to generate an app password:



**Fig. 3. 28** App passwords

After clicking generate it will appear a yellow box with the password

It is mandatory as well to provide Gmail credentials in the next file `/etc/postfix/sasl/sasl_passwd`.

```
smtp.gmail.com]:587 username@gmail.com:password
```

To not store the password in a plain text a hash must be done.

```
sudo postmap /etc/postfix/sasl/sasl_passwd
```

After writing this line a `sasl_passwd.db` will be created. The lines shown below are from `main.cf` file .We set the relay as Gmail and enable authentication with the hash:

```
relayhost = [smtp.gmail.com]:587  
# Enable SASL authentication
```

```

smtp_sasl_auth_enable = yes
# Disallow methods that allow anonymous authentication
smtp_sasl_security_options = noanonymous
# Location of sasl_passwd
smtp_sasl_password_maps = hash:/etc/postfix/sasl/sasl_passwd
# Enable STARTTLS encryption
smtp_tls_security_level = encrypt
# Location of CA certificates
smtp_tls_CAfile = /etc/ssl/certs/ca-certificates.crt

```

### 3.3 Economic Summary

This scenario is really cheap to implement, because almost all the software is open source or it have its free version. It is not counted the physical server, because a lot of corporation have this resources virtualized and generally will not be necessary to buy new one.

#### Capex

Software/Resource	Cost
Nagios	0 \$
Jenkins	0 \$
Sonar	0 \$
Artifactory Free version	0 \$
Bitbucket	0 \$
Apache	0 \$
Mysql	0 \$
Php	0 \$
Jira	0 \$
Total	0 \$

**Table 3. 7** Capex

#### Opex

Software/Resource	Cost
Nagios	0 \$
Jenkins	0 \$
Sonar	0 \$
Artifactory Free version	0 \$
Bitbucket	10 \$ up to 10 users
Apache	0 \$
Mysql	0 \$
Php	0 \$
Jira	10 \$ to 10 users
Total	20 \$

**Table 3. 8** Opex



The cost will be 20 \$ if we have less than 10 users, in case of having more than 10 the cost will escalate very fast because when talking to 25 users the price of Bitbucket will increase to 1800 \$ and Jira to 150\$. The cost can escalate very fast if we take maximum specs , for example in the case of Bitbucket the maximum cost will be 240.000 \$ for 10000 users, and for Jira will be 1500 \$ per month.

If any corporation doesn't want to spend so much money , can change bitbucket for a self-hosted git and won't notice any difference and Jira for a Open source tracking tool as pivotal tracker.

The above tables do not take in count the human cost. If we add the human cost but from an expert point of view, in others words, who is doing the job, know what is doing, it is not the first time it does it.

Setting up the Linux machine from scrap	8h
Installing all the required packets, including Mysql, java ,apache and php	6h
Configuring Sonar and installing	20h
Configuring Jenkins and installing	30h
Configuring Artifactory and installing	14h
Configuring Bitbucket and installing	5h
Configuring Jira and Installing	10h
Configuring Nagios and Installing	15h
Binding all Software together	10h

**Table 3. 9** Time for setting up

As we can see it is very cheap to implement this monitoring and integration system from economical point of view.

## 3.4 Linux Server

### 3.4.1 Ports

In the Internet protocol suite, a port is an endpoint of communication in an operating system.

A port is always associated with an IP address of a host and the protocol type of the communication, and thus completes the destination or origination network address of a communication session. A 16-bit number identifies a port for each address and protocol. This is known commonly as the port number.

Port numbers are from 0 to 65535. Ports 0 to 1024 are reserved for use by certain privileged services. For the HTTP service, port 80 is defined as a default and it does not have to be specified in the Uniform Resource Locator.

In order to have all services in the same machine it was a must to reconfigure all the application ports to be sure that was used only one time

Software	Port
Jenkins	8080
Bitbucket	7990
Sonar	9000
Apache	80
Artifactory	8081
Jira	8001 and 8002
Nagios	80/nagios

**Table 3. 10** Ports summary

### 3.5 Learned lessons

The first thing that I improved doing this thesis was my Linux knowledge , the fact of configuring and setting up from scratch everything make me learn how to set up and configure and how to bind together different software's from different developers using a Ubuntu system. It was indeed very important to learn which packets were missing in order to make work Jenkins or Sonar in my system. Its important to remark that Jenkins has to be configured for each project differently , it is not the same a maven project made on java that a ant project or a griddle project despite they have the same language.

As well I realized that Jenkins have a lot of more capabilities that the one that I implemented, some of them I will explained in the forecasting. It's important to remark that all the software was made to have the possibility to work together but they can work independently , whereby it's not necessary to use Jenkins to run an analysis code with sonar, but all software together creates a powerful continuous integration and monitorization Software really usable and recommendable for small , medium and big companies.

For my point of view is an indispensable system that should be implemented in order to improve the learning curve of workers and leave them more time to focusing in important task as developing

## CHAPTER 4: QUALITY CODE AND DEVELOPMENT COOPERATION

In this chapter is presented the cooperation between workers and how they divide the task in order to improve their developing and work as a team, also it will be explained some code quality metrics related to the projects that have been tested

### 4.1 Development cooperation

In a corporation it is very important the cooperation between workers. If they have tools that allow them to manage as optimum as possible the development process can increase the productivity as well as the income generated.

This section will explain how developers will cooperate, divide task, divide work and time scheduling in order to increase quality code. To summarize and avoid to get extended Umate project has been picked and will be used in order to show how developers are cooperating, but it can be extensible for each of the projects presented.

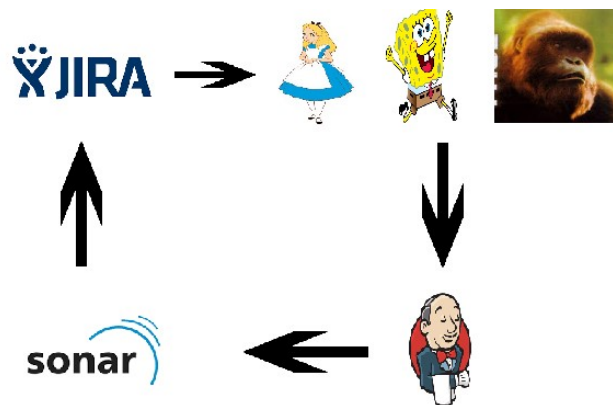


Fig. 4. 1 Flow of the scenario

Let's say that in a corporation they are developing an application. This corporation has a few hundred of workers. One of these workers is Charlie a technical leader and project manager who has at his orders some developers teams. Charlie as technical leader has to check and evaluate how their teams are going forward during the developer process, how they are spending their times, how they are improving their developing skills, how efficient they are, if they finish their job in the scheduling, etc.

One team, called Leonidas is developing the Umate project, this team is formed by 3 developers each of them having different skills when programming, one of them is Alice a senior developer, the other is called Bob is a junior developer and Joe, who has just join the team and has never worked on programming.

Max is the managing guy, who will be in charge of all the infrastructure, his main job related with the developer process will be to check Nagios services, in case of a service fail, he will fixed it before the others notice it, configure Jenkins if needed when a new project enters.

At the beginning, before starting the project, when the project has no code .The Leonidas Team and Charlie meets in order to split tasks. As the tool used to divide work is Jira and the work methodology used is scrum, the team leader will fix for the first spring the starting tasks (Fig. 4. 2).

Nombre de sprint:

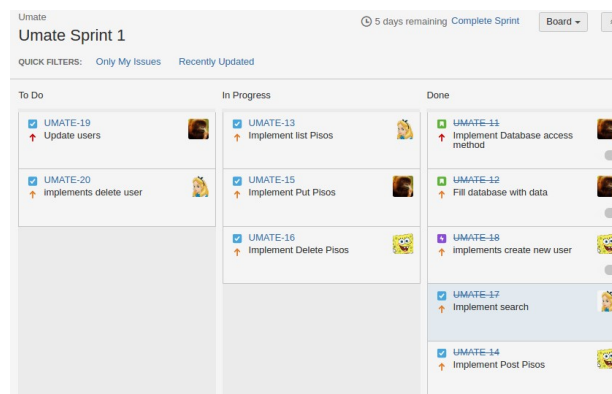
Duración:

Fecha de Inicio:

Fecha de Fin:

**Fig. 4. 2** New sprint meeting

Developers will start working. The first thing that they will do is to create a new branch as explained in the previous chapter in order to avoid bothering them selves. After finishing their work, they will merge with the master if everything works as expected. All developers will start task depending on their priority, which is marked with the red arrow (higher priority). As Alice is the senior developer, she will receive the most complicate task of the project, followed by Bob and Joe who will get the most simplistic tasks.



**Fig. 4. 3** Sprint view

When the last day of the spring reaches, the Team Leader will have a meeting with Leonidas Team. He will check each one of the issues, if everything is as

expected, he will switch the resolved state (done) to the close state.

The issues that could not be resolved will pass to the second spring, and their priority will change to highest priority.

Each time a developer finishes an issue and pushes the code to Bitbucket triggers an auto-build in Jenkins. As explained before, if the implementation requires an artifact/dependency, Artifactory will provide it to Jenkins. If the dependency is self-done it will be updated manually by infrastructure guy Max, in order to make it accessible by everyone in the corporation.

Jenkins will ask Sonar to scan the project, in order to retrieve some metrics of the code and be able to improve or change inconsistencies, bugs or vulnerabilities.

In previous Sonar and Jira versions, there was a plugin which takes all the issues created by sonar and put it on Jira automatically, but for the latest versions, SonarQube(5.5) and Jira 7 there is no support yet.

As well Charlie the technical leader will check the commits done in Bitbucket by the team, and using sonar metrics will get a close idea about how well his developers are working, he will know how much Reliable is the code, its security, Maintainability, Duplications, Size, Complexity and Documentation and as well how many issues have sonar retrieved from the code by developer.

Let's say that before finishing the first spring, there is a problem and the sonar service stop working. Max the infrastructure guy will receive an email at his account after the third check of Nagios and he will be warned before the developers notice the problem, that means that Max will fix the problem before the issue could delay the project release date.

This process will keep going until the final release of the project and will allow less experienced developers to get well programming habits and standardize their programming to the company needs . Improving the quality code and the cooperation process.

## **4.2 Measures of Quality Code**

For the measures of quality it will be used Umate project.

### **4.2.1 Reliability Remediation Effort**

As explained before Joe has never worked on programming, and he has being asked to implement a class who will access to the database .



**Fig. 4. 4** Measuring the quality of the code: reliability

As you can see in the Fig. 4. 4 above, Joe is an inexperienced developer and the class that he has created with 705 lines of code has 27 bugs with all the ranges of severity. This provoked that his part of the project will not pass the quality gates.

After this results. Joe using this results will try to fix it using sonar tips, but as he don't have any experience, to fix the issues that he is not able to do it alone Bob or Alice will have to sit with Joe in order to give him some tips to how to improve his part of the code. Sonar says that if he wants to fix all the issues he will have to spend more or less 4 hours 50 minutes.

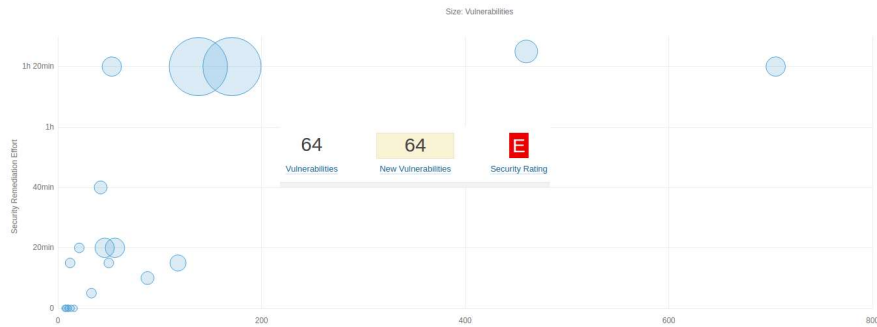
JMate/src/main/java/EAPProject/UMate/PisoResource.java	E
JMate/src/main/java/EAPProject/UMate/OperacionesBD.java	E
JMate/src/main/java/EAPProject/UMate/security/SHA1.java	D
JMate/src/main/java/EAPProject/UMate/transaction/UsuarioT.java	A
JMate/src/main/java/EAPProject/UMate/model/Usuario.java	A
JMate/src/main/java/EAPProject/UMate/uMateApplication.java	A
JMate/src/main/java/EAPProject/UMate/transaction/PisoT.java	A

**Fig. 4. 5** Rating Reliability of each class

The Team Leader will have the option to check the rank of each class to verify the effectiveness of each developer and see in a graphical view how they develop.

#### 4.2.2 Security Remediation Effort

In the case of security remediation there are much more issues due it's nature. Variables or methods with an incorrect visibility will be tag as vulnerabilities, Exceptions using throwables instead of catch will be tag as vulnerabilities, useless throw clause too.



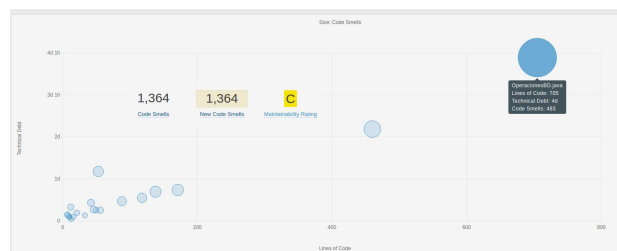
**Fig. 4. 6** Security Remediation Effort

/Mate/src/main/java/EAPProject/UMate/PisoResource.java	E	1h 25min
/Mate/src/main/java/EAPProject/UMate/transaction/PisoT.java	D	1h 20min
/Mate/src/main/java/EAPProject/UMate/model/Piso.java	D	1h 20min
/Mate/src/main/java/EAPProject/UMate/OperacionesBD.java	D	1h 20min
/Mate/src/main/java/EAPProject/UMate/IOperacionesBD.java	D	1h 20min
/Mate/src/main/java/EAPProject/UMate/security/SHA1.java	D	40min

**Fig. 4. 7** remediation effort and security rating

### 4.2.3 Maintainability Remediation Effort

Sonar introduces the concept of Code Smell. Code smell, also known as bad smell, in computer programming code, refers to any symptom in the source code of a program that possibly indicates a deeper problem. Code smells are usually not bugs, they are not technically incorrect and do not currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future.



**Fig. 4. 8** Maintainability Remediation Effort

Unneeded comments will be tag as code smells, missing header comments too, misallocation of code will be tag as code smell (not tabled properly).

### 4.2.4 Complexity

The complexity of Umate project is 272, it is a number really high what it means that the code it is not really good, it has a lot of paths to walk before reach the function that it needs.

It is as well provided the complexity per file, that give you a clue which class has to be reviewed in order to reduce his complexity and make the code faster.

OperacionesBD.java	96
PisoResource.java	49
transaction/PisoT.java	38
model/Piso.java	38
model/Usuario.java	12
transaction/UsuarioT.java	10
security/SHA1.java	7
HibernateUtil.java	5
transaction/ComentarioT.java	4
model/Comentario.java	4
LoginRegisterResource.java	2

**Fig. 4. 9** Complexity found in each class

OperacionesBD is a java class, which its main functionality was to “talk” with the database. It has so high complexity due to a un-optimized code that has repeated for each time that want to access the database the same lines with a minor changes to implement the functionality. Then is a repetitive method, which does similar things, but changing the value to retrieve at the end.

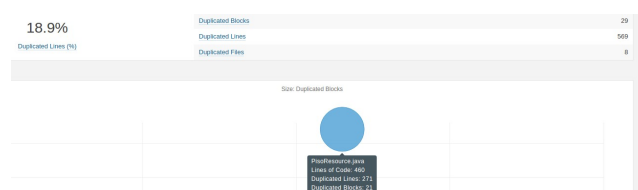
PisoResource has as well a lot of complexity due all API calls are saved in this file and there is one for each type.

```
@GET
@Path("/{Listarfav/{userid}")
@Produces(MediaType.UMATES_PISO)
```

**Fig. 4. 10** API calls

Piso.java and PisoT.java have same complexity due to both classes are similar, one is the model and the other is the transaction class.

## 4.2.5 Duplication



**Fig. 4. 11** Duplication analysis result



Duplication is the worst coding sin because it perpetuates all the others. For instance, copying and pasting a block of code duplicates its Bugs and Potential Bugs, its Coding Standards Breaches, and so on. To help you detect and eliminate duplication, SonarQube uses its own copy/paste detection engine, which can detect duplications:

- Within a source file
- Across multiple files in a project
- Across modules of a project
- Across multiple projects

A piece of code is considered as duplicated as soon as there is the same sequence of 10 successive statements regardless of the number of tokens and lines.

### 4.3 Projects Comparative

NAME ^	VERSION	LOC	BUGS	VULNERABILITIES	CODE SMELLS
Mario	1.0	549	0	2	309
Node	1.0	51,176	2,418	143	5,520
Tuneling	1.0	806	5	36	27
UMATE	1.0	2,057	29	64	1,364
WorldBirths	1.0	1,506	34	20	993

**Fig. 4. 12 Comparative**

Mario App is an application that provides a guide for Super Mario Game for Android , in others words , tips for beat the game. This application was programmed as fast as possible, the main intention was to bring up to the playstore as fast as possible the app without focusing on maintainability and reusability , because the goal was to profit the Boom that was caused by the game and earn money as fast as possible. It worked at the beginning but the account was banned due to bad PEGY rating for pub. This App has 2 vulnerabilities, because it has 2 intentionally test in order force the application crash to find possible bugs. The App got a rating of A related to bugs, D for security rating(due to critical vulnerabilities that are explained before) , B for Maintainability due to 308 code smells and 56 of duplicated code, because as I explained before the code was made fast to profit the Boom.

Node App is a test application that I found on internet in order to test different things like maven, graddle , express and that Jenkins was able to running it an provide it a valid output, it has so many bugs and vulnerabilities due almost every piece of the code is deprecated because it was developed 4 years ago Tunneling application , is an application to test sproud protocol, sonar found it 36 vulnerabilities due to the python part of the code was made in order to join every functionality in 3 different files, one for 802.3, one for plotting and one for the Wifi protocol, this has brought about that there is a lot of repeated

structures. The bash part of the code could not be analyzed due to sonar does not support it.

World births is an app to check real time births in the world , almost 34 bugs that it have are about Lazy initialization of "static" fields should be "synchronized", this is because this app use multithreading and synchronized field avoids writing at the same time at the same variable, and others bugs are related to use of BigDecimal instead of double. Because the addition of to float variables save it in another float variable is not right , because is not the accurate result.

For the result to be compliant, the addition must be in a double variable. The majority of bugs of this projects are due to visibility mistakes, where some final, private where not added. The case of Umate Project in the previous sections.

It is important to remark that the building times of each project not only depends on the complexity also on the language and code lines.

Project name	Building time	Code lines	Complexity
Umate	14.2 s	2057	272
Tuneling	6 s	806	168
SuperMario	8.5 s	549	85
WorldBirths	8.5 s	1506	223
Node-example	19 s	51176	18006

**Table 4. 1** Building, code lines and Complexity

#### 4.4 Learned lessons

Jira is a powerful tool that ease cooperation between workers , cooperation is a main pillar in a successfully development process it is a fundamental to achieve success.

Sonar gave me the opportunity to realize how many bad programming habit I had acquired during all these years since I learned programming and provided to me a way to locate all the errors that I did in a fast and efficient way. Sonar as well provide a way to create your own rules, to help others to acquire good habits or to follow some kind of standardization in a corporation.

All the system together could be implemented in any enterprise that develops and increase their speed when developing because tedious process have been automatized and standardized.

As well you could track the improvement curve of any worker using the output provide it by the system, giving him a lot of tools to change how he is used to develop.

## CHAPTER 5: CONCLUSIONS AND FORECASTING

### 5.1 Conclusions

Cooperation between workers is one of the main pillars of successfully enterprise. If the cooperation increases, the workers will become more effective and the enterprise will get more revenues and will expend less money.

Revenues will be increased because as they cooperate more and they become more effective, they will finish projects early, they will grow faster in the as will learn from each other much more.

Less money because as they finish faster, they will be able to pass to the next project early what it gets translated to more number of projects per year

As well the ability of keeping track of what a worker does, force them to keep constant effort and avoid them less productivity periods with a well splitted work assignation

As the team grows , and gets bigger , this system will increase it's effectiveness because will standardize and make it reproducible every step of the process.

As well it is important to keep the code clean, organized and well structured in order to increase Maintainability , Security and Reliability and in order to help others who did not worked yet on the project to understand faster what each piece of code does and to become profitable earlier.

This system that has been tested in this project will provide the developers a standardized and automated way to focus their developing showing that there are faster and better manners to develop when a team is involved

Work cooperation between workers, deployment of the infrastructure and configuration , analysis of the code and monitorization system and simulation of a real environment, all these objectives were accomplished.

### 5.2 Forecasting

This project could be improved by enabling the autodeploy capability of Jenkins. In other works Jenkins provide a way to communicate with an external server previously configured and to deploy the projects that are Java, Phyton or Node on the fly, whereby, when the developer press the push bottom the jar, js or py that is created after the build it is transferred to the external server and deployed automatically without human interaction and providing a testing or release environment faster and effortlessly than in the older way.

It is important to remark that for each language the autodeploy works differently, for example for java projects that use maven and tomcat, there will be

necessary in the external server to install maven, tomcat and java and to configure properly, whereby same software versions as the project needs.

