Final degree project

COMPUTER ENGINEERING DEGREE

Faculty of Mathematics
University of Barcelona

Quality Assurance implementation in the Software Development Unit of Eurecat

Author: Juan Manuel Morales Garzón

Directors:	Dra. Petia Radeva
	Stefan Dauwalder
Realised:	Departament de Matemàtiques i Informàtica
	Facultat de Matemàtiques i Informàtica
	Eurecat Centre Tecnològic
Barcelona,	26 of January of 2017

Abstract

The QA (Quality Assurance) process is involved in the software development from the birth of the idea to the implementation of the software itself. In the early stages, it has the aim to verify that the objectives are well exposed and the requirements are accurate. On the design and codification phases, it monitors compliance with the set standards. Finally, it verifies that the software in operation respects the requested requirements and that the delivery to the client is done under the right conditions. The QA procedure includes a set of quality tests to ensure the correct functionality such as unit tests, integration tests, regression tests, acceptance tests, functional tests, load tests, stress tests, system tests and smoke tests.

Nowadays, there is a growing number of companies that are realising the importance of the Quality Assurance process and starts to implement it, as this process in long term results in saving money and building better software in terms of quality, time and robustness.

This project consists of the implementation of everything related to the QA procedure on the SDU (Software Development Unit), concretely on the Health products. This implies the study and application of different techniques, tools and methodologies that are currently used on the QA world, and the implementation of automated tests with the objective to speed up work.

This project was carried out with the collaboration of Eurecat Centre Tecnològic of Catalonia, in the offices of Barcelona, which helped to fulfill this project’s objective by allowing to apply the different Quality Assurance techniques, tools and methodologies on two health products: eKauri and eKenku. Both products are based on providing remote assistance to elder and chronic patients, with data retrieved from a mobile app, environmental and physiological sensors, which help to reduce costs, waiting lists, and grants continuous analysis of the patient at the time that the patient can live longer independently.

Resumen

El proceso de QA (Aseguramiento de la Calidad) está involucrado en el desarrollo de software desde el nacimiento de la idea hasta la implementación del propio software. En las primeras etapas, tiene el objetivo de verificar que los objetivos están bien expuestos y los requisitos son precisos. En las fases de diseño y codificación, supervisa el cumplimiento de las normas establecidas. Por último, verifica que el software en funcionamiento respeta los requisitos solicitados y que la entrega al cliente se realiza en las condiciones adecuadas. El procedimiento de QA incluye un conjunto de pruebas de calidad para garantizar el correcto funcionamiento, como pruebas de unidad, pruebas de integración, pruebas de regresión, pruebas de aceptación, pruebas funcionales, pruebas de carga, pruebas de esfuerzo, pruebas del sistema y pruebas de humo.

Hoy en día, hay un creciente número de empresas que se están dando cuenta de la importancia del proceso de QA y empiezan a implementarlo, ya que este proceso a largo plazo se traduce en un ahorro de dinero y la creación de un mejor software en términos de calidad, tiempo y robustez.

Este proyecto consiste en la implementación de todo lo relacionado con el procedimiento de QA en la UDS (Unidad de Desarrollo de Software), concretamente en los productos de Salud. Esto implica el estudio y la aplicación de diferentes técnicas, herramientas y metodologías que se utilizan actualmente en el mundo de QA, y en la implementación de pruebas automatizadas con el objetivo de agilizar el trabajo.

Este proyecto se ha llevado a cabo con la colaboración de Eurecat Centre Tecnològic de Cataluña, en las oficinas de Barcelona, que ha ayudado a cumplir el objetivo de este proyecto permitiendo aplicar diferentes las técnicas, herramientas y metodologías de QA en dos productos de salud: eKauri y eKenku. Los dos productos están basados en la prestación de asistencia remota a pacientes ancianos y crónicos, con datos obtenidos desde la aplicación móvil, sensores ambientales y fisiológicos, que ayudan a reducir costes, listas de espera y ofrece un análisis continuo del paciente a la vez que el paciente puede vivir más tiempo independientemente.

Resum

El procés de QA (Assegurament de la Qualitat) està involucrat en el desenvolupament de programari des del naixement de la idea fins a la implementació del propi programari. En les primeres etapes, té l'objectiu de verificar que els objectius estan ben exposats i els requisits són precisos. En les fases de disseny i codificació, supervisa el compliment de les normes establertes. Finalment, verifica que el programari en funcionament respecta els requisits demanats i que el lliurament al client es realitza en les condicions adequades. El procediment de QA inclou un conjunt de proves de qualitat per garantir el correcte funcionament, com a proves d'unitat, proves d'integració, proves de regressió, proves d'acceptació, proves funcionals, proves de càrrega, proves d'esforç, proves del sistema i proves de fum.

Avui en dia, hi ha un creixent nombre d'empreses que s'estan donant compte de la importància del procés de QA i comencen a implementar-lo, ja que aquest procés a llarg termini es tradueix en un estalvi de diners i la creació d'un millor programari en termes de qualitat, temps i robustesa.

Aquest projecte consisteix en la implementació de tot el relacionat amb el procediment de QA a la UDS (Unitat de Desenvolupament de Programari), concretament en els productes de Salut. Això implica l'estudi i l'aplicació de diferents tècniques, eines i metodologies que s'utilitzen actualment en el món de QA, i en la implementació de proves automatitzades amb l'objectiu d'agilitzar el treball.

Aquest projecte s'ha dut a terme amb la col·laboració de Eurecat Centre Tecnològic de Catalunya, a les oficines de Barcelona, que ha ajudat a complir l'objectiu d'aquest projecte permetent aplicar les diferents tècniques, eines i metodologies de QA en dos productes de salut: eKauri i eKenku. Els dos productes estan basats en la prestació d'assistència remota a pacients ancians i crònics, amb dades obtingudes des de l'aplicació mòbil, sensors ambientals i fisiològics, que ajuden a reduir costos, llistes d'espera i ofereix una anàlisi contínua del pacient alhora que el pacient pot viure més temps independentment.

Acknowledgements

The completion of this thesis and its implementation would not have been possible without the collaboration of Eurecat Centre Tecnològic and many people that work in there. Although it is difficult to review everyone, I would like to mention some of them.

First of all I would like to thank my tutor, Petia, for introducing me to Eurecat and for giving me that first vote of confidence in selecting me for the practicum. Really thank you very much. I would also like to especially thank my tutor at Eurecat, Stefan, for all the support and advice he has given me throughout these months, as well as the entire eKauri and eKenku team, who have answered all my questions and have made me feel integrated in the team.

Finally, I would like to thank my girlfriend, Maria Rosa, for being always by my side, giving me encouragement when I was overwhelmed and for enduring my "I do not arrive" in the last stage of the project.

Table of contents

1 Introduction

Nowadays, many companies release their products with a significant number of errors. After releasing the product, fixing those errors means the company has to waste an unexpected amount of money and time, not to mention that the customer reliance on the company may decrease. For this reason, there is a growing concern about the QA (Quality Assurance) process.

Moreover, in computer engineering degree, Quality Assurance, from now on referred as QA, is barely mentioned. In others words, during many exercises that students do during the degree, students try to avoid bugs in their programs, but techniques or processes to detect and prevent bugs are not taught.

This project tries to deal with these fields. In order to accomplish that, it is divided in two main sections: theoretical and practical.

The first section, the theoretical one, is a study about what is QA, which are its techniques, tools and methodologies to avoid the maximum number of bugs and problems on the product release.

In the second section, the practical, it is analysed how to bring to practice the information learned in the section, implementing the QA procedure in the Software Development Unit, from now on referred as SDU, in Eurecat Centre Tecnològic, hereinafter referred to as Eurecat, and it will focus on automation of the tests.

1.1 Motivation

The motivation to do this project is, as said before, that more and more people (companies, developers, students) realise the usefulness of the QA process, and in this field there is a lack of information regarding its importance.

The Eurecat Center at the moment, has no QA procedure yet, so this project will study and explore the one that suits the best for them. To do this, this project would have to cover the next points:

1. Investigation of the QA world (methodologies, tools and techniques)
2. Software development models (specifically Scrum)
3. Implementation of the QA process in a real project
4. Tests automation

1.2 Goal of the thesis

The aim of this project, consists on the implementation of the QA process in the SDU in the Eurecat Center, concretely on the health products. The implementation will consist on different techniques, tools and methodologies that exists on the QA world, in order to establish a new or existent methodology that suits the best for the center.

Besides of all different QA techniques, the project will focus on the implementation of automation tests.

2 Viability analysis

2.1 Project description

The objective of this proposal is to use or create a methodology of developing software according to the available team in order to perform better. The methodology will consist on a set of tools and techniques to make developers aware of the possible bugs and problems that can appear at developing certain tasks or the errors they have committed. This will lead to a series of habits that will make developers better at their job. Besides, the project will focus on the automation tests, so the most quantity of tests could, and some of them should, be performed automatically.

2.2 Identification of Risks and contingency plans

The main risks that can arise in this project are, when putting into practice, if some of the methodologies or techniques prove to be useless, either because it delays the development too much or because it is not possible to find any of the existing errors. To avoid this problem to happen, methodologies and techniques will be discussed and putting in practice one by one.

Another problem that can happen is when developing tests, if a flaw is committed without noticing, so it can lead to wrong execution results and skip errors on the release. To try to avoid that, we will define and develop the test in the most impartial possible way with the expected output and then test with correct and incorrect data to check that it works as supposed. Of course, if a flaw on the test definition or development is found, it is mandatory to be solved.

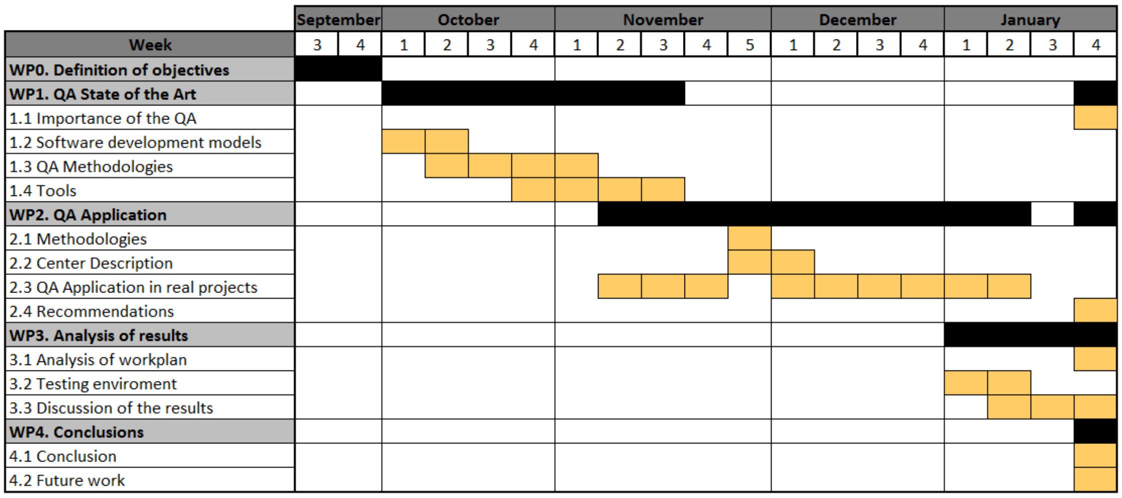
2.3 Is it viable?

The viability of this project depends on time and manpower. As it only will be one person fully dedicated to the QA process and the time for this project is four months, it would be difficult to complete all automation tests and techniques on time. Although, with the techniques and methodologies to be implemented, the methodology of the QA process for the Eurecat case will be defined and most of the automation tests will be performed.

Finally, if any of the tests or methodologies are not performed in time, it will be done in the future. Therefore, in the end, the main goal of this project to build a QA Methodology for Eurecat will be accomplished in time. For that reason this project is considered viable.

2.4 Workplan and timetable

The following figure shows a Gantt Diagram of the workplan to develop this project. It is divided into five main sections: The definition of all the work to be done during these months, the current state of the QA world, the application of the QA process in real projects, the analysis of work done and the conclusions.



	September		October				November					December				January			
Week	3	4	1	2	3	4	1	2	3	4	5	1	2	3	4	1	2	3	4
WP0. Definition of objectives	■	■																	
WP1. QA State of the Art			■	■	■	■	■	■	■										■
1.1 Importance of the QA																			🟨
1.2 Software development models			🟨	🟨															
1.3 QA Methodologies				🟨	🟨	🟨	🟨												
1.4 Tools						🟨	🟨	🟨	🟨										
WP2. QA Application								■	■	■	■	■	■	■	■	■	■		■
2.1 Methodologies											🟨								
2.2 Center Description											🟨	🟨							
2.3 QA Application in real projects								🟨	🟨	🟨		🟨	🟨	🟨	🟨	🟨	🟨		
2.4 Recommendations																			🟨
WP3. Analysis of results																■	■	■	■
3.1 Analysis of workplan																			🟨
3.2 Testing enviroment																🟨	🟨		
3.3 Discussion of the results																	🟨	🟨	🟨
WP4. Conclusions																			■
4.1 Conclusion																			🟨
4.2 Future work																			🟨

Figure 1. Gantt Diagram for the QA Process

3 QA State of the Art

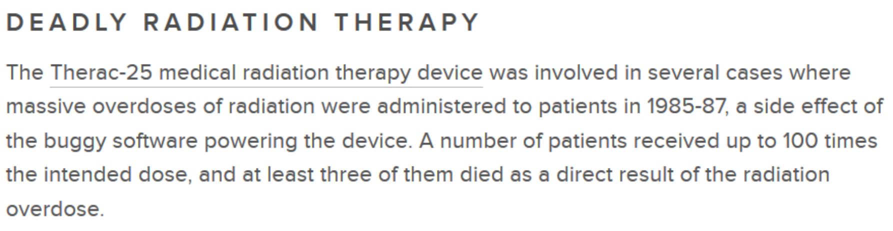
The QA process is a set of techniques, methodologies and methods applied at software development models in order to prevent bugs from appearing. Applying the QA process, software developed will have more quality as it is tested constantly in order to find and report bugs that have not been prevented. Although the work of a QA engineer is to ensure the quality of the software, quality is a responsibility of everybody on the team.

3.1 Importance of the QA

This section explains the importance of QA in software development. There are many reasons that could explain the importance of QA, so here, will be explained some of the points that have been considered the most important.

Moreover, the software developed, are for customers who pay for it, so users have to be able to use the software without any problem. In addition, if customers experience errors, their confidence in the company that developed it will decrease.

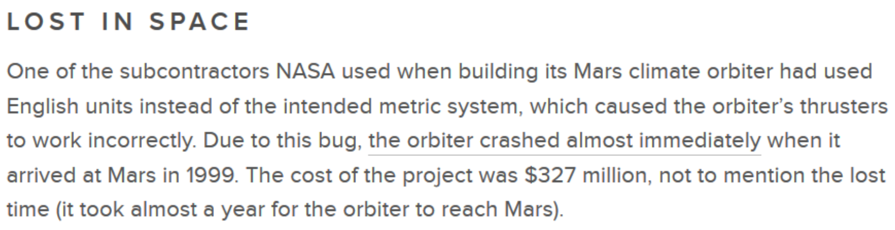
Another very important reason is that, the incorrect operation of a software for whatever reasons, depending on the product can cause serious accidents. For example, a pair of historical software bugs that caused extreme consequences can be seen in the following figures.



DEADLY RADIATION THERAPY

The Therac-25 medical radiation therapy device was involved in several cases where massive overdoses of radiation were administered to patients in 1985-87, a side effect of the buggy software powering the device. A number of patients received up to 100 times the intended dose, and at least three of them died as a direct result of the radiation overdose.

Figure 2. Real case example of software bugs [1]



LOST IN SPACE

One of the subcontractors NASA used when building its Mars climate orbiter had used English units instead of the intended metric system, which caused the orbiter's thrusters to work incorrectly. Due to this bug, the orbiter crashed almost immediately when it arrived at Mars in 1999. The cost of the project was $327 million, not to mention the lost time (it took almost a year for the orbiter to reach Mars).

Figure 3. Real case example of software bugs [1]

3.2 Software Development Models

Software development models are a set of processes selected for the development of a project. The chosen model depends on the goal, team and time limitations of the project and QA Methodologies depend on the chosen software development model.

There are several models of Software development. Some of them which will be explained later are: waterfall model, V model, spiral model, and agile model. Between those models, the one that will be explained in detail will be the agile models, concretely the Scrum methodology, which is the model currently used in Eurecat.

3.2.1 Waterfall Model

The waterfall development model [2] is a sequential process, in which progress is seen as flowing gradually downwards through the phases of requirement analysis, design, implementation, testing, deployment and maintenance. As can be seen on the figure 4, it looks like a waterfall, and that is the origin of the model name.

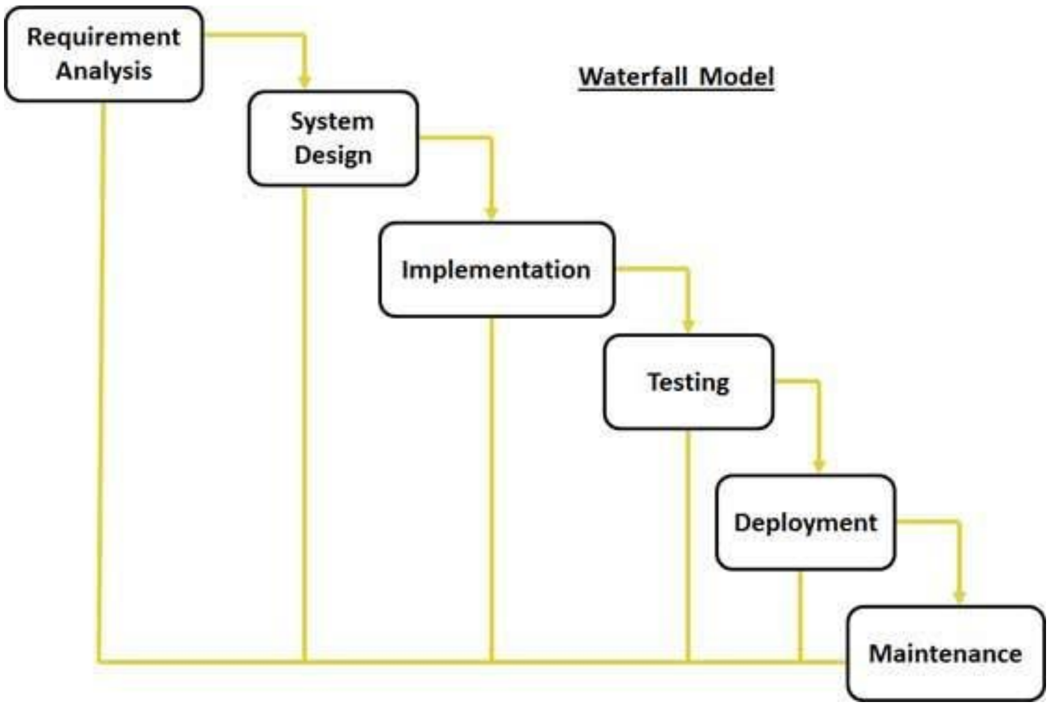


Figure 4. Waterfall Software Development Model [2]

As said above, this model is based on a systemic and sequential process, so this means that the next phase of development can be started only if the previous one is finished. The sequential phases of the waterfall model consists on:

1. Requirements and analysis: In this phase it is tried to gather all requirements needed to do the project and documented while analysis of the software is done.
2. System design: It consists of studying the requirements document in order to prepare the design, specifying hardware needed and defining system architecture.
3. Implementation: With the system design, the project is developed in small portions of the projects or small programs which function is to perform simple tasks of the project, which will be integrated on the next phase.
4. Integration and Testing: Here all small programs developed in the previous phase are tested and then integrated on the system. After all integration is done, tests are performed on the complete system in order to detect failures.
5. Deployment: Once all testing is done, and passed, the product is deployed on the customer environment or released into the market.
6. Maintenance: Some issues appear on the client environment, so to fix those issues a process of maintenance is done, which will result in a patch or new versions release to improve the current product.

3.2.2 V Model

V model [3] is a model that can be considered an extension of the waterfall model. V model, instead of moving downwards in a sequential way as the waterfall, the process steps are going upwards after the coding phase, as can be seen below on figure 5.

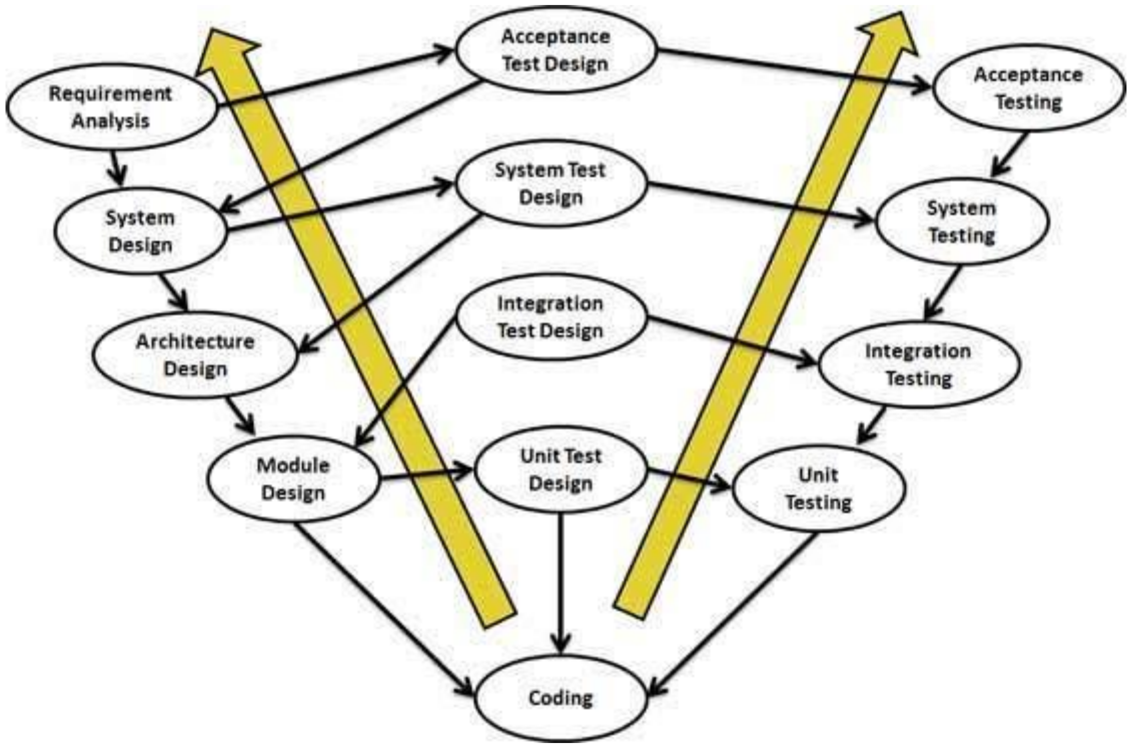


Figure 5. V Software Development Model [3]

This model can be divided into three phases:

Verification Phase:

- Requirement and analysis: In this phase, as in the waterfall model, it is tried to gather all requirements needed to do the project and documented while analysis of the software is done.
- System design: It consists on study the requirements document in order to prepare the design, specifying hardware needed and defining system architecture.
- Module design: in this step the internal design for all the system modules is specified.

Coding phase

- In this phase the programming language that will be used for the project is decided and developers starts to code following the coding guidelines and standards. The code is reviewed several times and optimized for best performance before final release.

Validation phase:

- This phase performs different types of testing to ensure the correct functionality of the program such as unit testing, integration testing, system testing, and acceptance testing, which will be explained on the Types of testing section of this document.

3.2.3 Spiral model

The spiral model [4] combines iterative development with some aspects of the waterfall model. It consists of four phases distributed on cycles or iterations that are also called spirals. In each cycle or iteration the software goes through those phases, performing different activities, as can be seen on the figure 6.

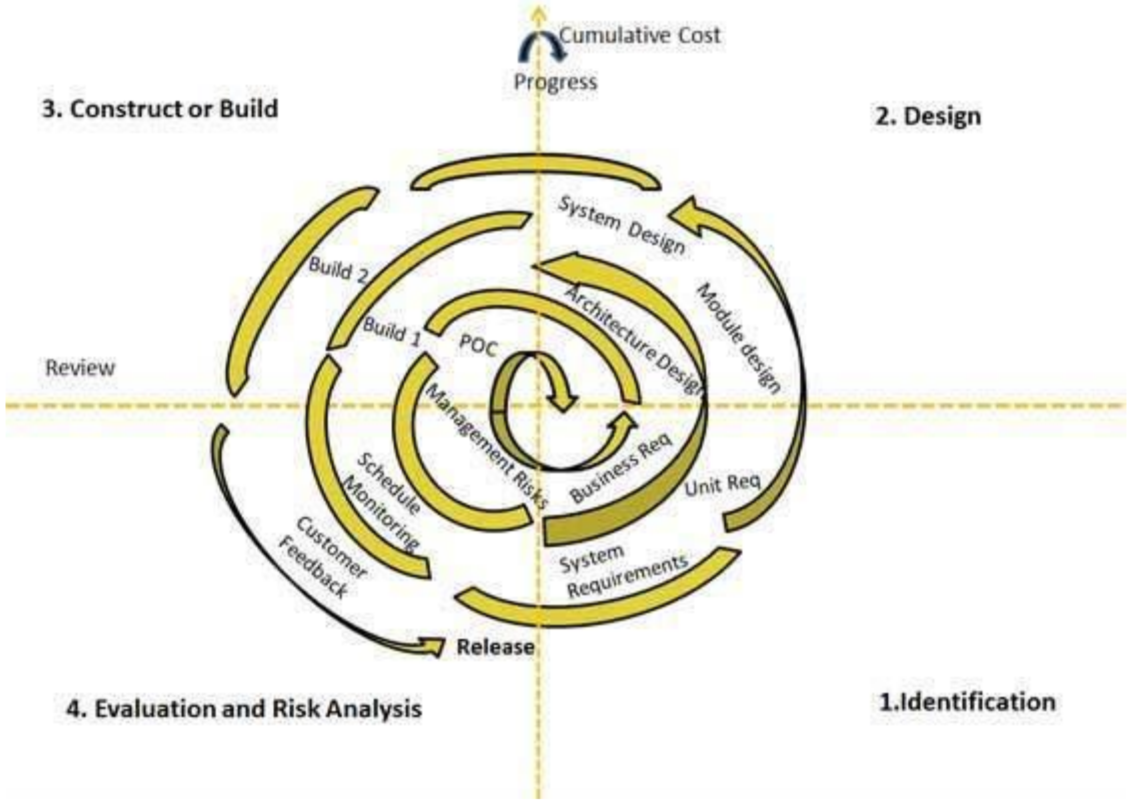


Figure 6. Spiral Software Development Model [4]

The four phases of this model are:

1. Identification: This is the initial phase of the spiral in which it is tried to gather all the requirements needed for the project. On the next cycles or spirals, this phase will update or add new requirements needed for the system as the project evolves.
2. Design: At the beginning of this phase the architectural and logical design is defined. Also, it is defined provisional physical design of the product in early cycles and final design on the end.
3. Construct or build: In this phase developer starts developing functions of the system as they are defined. At the first cycle a Proof of Concept (POC) is developed.
4. Evaluation and Risk analysis: This phase performs a risk analysis, which includes identifying, estimating, and monitoring technical and management risks. After testing the release, at the end of every iteration, the customer evaluates the software and delivers a feedback report.

3.2.4 Agile model

Agile model [5] is based on iterative and incremental models. The main focus of this model is on customer satisfaction. The idea of this model is to deliver at short time functional software product that will be growing incrementally at every iteration. This way, the customer can take a look at the current software and check that requirements were understood correctly or perform changes.

In an agile project, all tasks are divided into small builds of functional code that will be delivered on the established iteration time to the customer.

Two of the most known examples of agile models are Extreme programming and Scrum that will be explained on the following pages.

3.2.4.1 Extreme programming

Extreme programming [5.1] is one of the best-known agile development models, and as an agile model, it has iterative and incremental development, and performs constant releases in short period of time or cycles. In each release the software is adapting to new features or requirements of customers.

The extreme programming model has some fundamentals features:

- Continuous unit tests, including regression tests.
- Programming in pairs: this way the code can be reviewed and discussed meanwhile it is being written.
- Fixing all errors before adding new features to the project.
- Code simplicity: on extreme programming it is better getting first whatever to work, making it the simplest way and then if it is necessary to refactor it.

3.2.4.2 Scrum

Scrum [5.2] is an agile model, which gives team members the freedom of self-organizing their work by everyday communication and cooperation. As an agile methodology it is expected to react instantly and efficiently to changes.

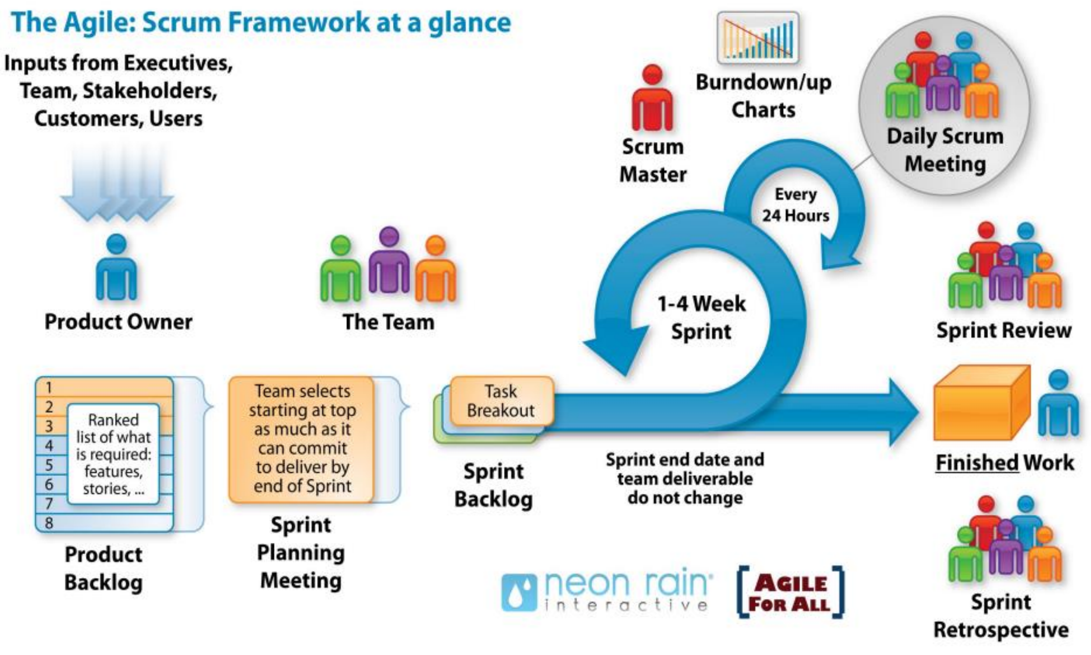

The Agile: Scrum Framework at a glance
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective
neon rain interactive
Agile For All

Figure 7. Scrum Idea [5.3]

In the previous figure, a diagram representing the Scrum idea is shown. In the following subsections its components will be explained, which are divided in three sections: Artifacts, Roles, and Events.

Artifacts

In Scrum all the documents generated in product development, which are necessary to ensure that everyone in the team has the same understanding of the product, are called artifacts. The following lines explain the documents that are always present in the Scrum process.

Scrum relies on transparency, so every task (or subtask) has to have a “Definition of Done”, which means that when any task is defined as “Done”, everyone must understand exactly what it means “Done”. So, this means that the developer performing the task would know exactly what is asked to mark the task as done.

Product Backlog

The Product Backlog is a list of everything that is needed on the final product ordered by priority. The responsible to create and update this list is the Product Owner, who will be explained later. This artefact, as long as the product is alive, is never complete, because it evolves with the product and the environment. After the product is released, the Product Backlog is still alive as the environment of the product may change, leading to incompatibilities or new features can be added to next versions.

There is only one Product Backlog on the project as it is the roadmap and everyone has to follow the same. To summarize, the Product Backlog is a view of everything that need to be done by the Team in order of priority.

Product Backlog Item

Product Backlog Items, also called User Stories, are the tasks defined by the Product Owner (explained below) which explains the main functionalities of the product under construction. These User Stories can be divided in subtasks if their complexity is high.

Sprint Backlog

The Sprint Backlog is a subset of items from the Product Backlog that contains the set of tasks that should have been done by the end of that Sprint. The chosen Product Backlog items cannot be changed during the Sprint. A Product Backlog item can be divided into smaller tasks that will have an estimated time to complete their development in Sprint time.

Every task on the Sprint Backlog is assigned to a member of the team and an estimate time to develop the task. The team member would have to implement all of their tasks at the end of the Sprint in order of priority, although the task may have an incorrect estimation time or the developer may encounter some problem that makes the task more difficult. In these cases, the task may remain unfinished for that Sprint (and it would not be integrated) and will be in the next Sprint.

Increment

Every time a Sprint is finished, a piece of the product is created, that it is integrated with previous pieces of the product. The set of these pieces of product is called Increment. So the Increment is the functional product created from Sprint and growing in the next Sprints as more functionalities are integrated.

This Artifact is still alive when the product is released, as improvements or updates that may be done would be also an Increment of the product.

Roles

Product Owner

The Product Owner is the person responsible for creating and managing the Product Backlog. This task is to create and manage the Product Backlog items, also called “User Stories”, making it as clear as possible so that it can be understood without errors. It is also responsible for ordering the “User Stories” by priority, so the important functionalities are performed first.

Product Manager

The Product Manager is the person who defines the roadmap of the product, the project cost and have to deal with sales and marketing teams, resource management teams, etc.

Scrum Master

The Scrum Master is not the manager of the team. So its task is not to tell people what or how to do something, neither to assign tasks. The goal of the Scrum Master is to help the Development Team and Product Owner to accomplish their goals. Their responsibilities are to ensure that the rest of the team understands how Scrum works and promulgates its use.

Development Team

Development Team are a set of professionals that work with the Product Backlog Item or User Stories to turn them into Increments at the end of the Sprint. The team organizes and manages their work, so nobody tells them how to do their work, as long as their work meets the requirements of the task.

Events

Sprint

Sprints are iterations of a fixed duration, which oscillate between one to four weeks. Each iteration, the team will have to deliver a small functional part of the software to the product owner. This way, the product owner can see the progress of the work and can check the status of the software continually, so if any specification are not interpreted correctly o any change is needed it is detected at an early stage and is easier to solve.

Sprint Planning

Each Sprint the work to be done is selected at the beginning in the Sprint Planning and it is selected with the collaboration of all the Scrum Team. Sprint Planning is time-boxed to a maximum of 2 hours per week of Sprint duration.

Daily Scrum

An activity that the Development Team has to perform is Daily Scrum. It consists of a brief meeting (of a maximum of 15 minutes) of the team members to synchronize their work and to plan the work of the following day. These meetings cover what has been done from the previous Daily Scrum up to what will be done for the next Daily Scrum. Usually those meetings are hold the same place and time to make it easier.

Sprint Review

At the end of the Sprint an informal meeting is done. This meeting is used to inspect the Increment of that Sprint and cooperate on the next things that could be done to improve value. During the meeting are discussed the tasks that has been finished and those that have not, the problems that have been encountered and how it has been solved.

Sprint Retrospective

The Sprint Retrospective is a chance for the Team to revise their work and build a plan to improve it in the next Sprint. It is performed after the Sprint Review and before the next Sprint Planning.

3.3 QA Methodologies

There are a lot of different types of testing [6] [7], some of them are explained in the next pages. All of these tests can be divided mainly in two groups: White Box Testing and Black Box Testing as is shown on the diagram 1. Every group has its own types of tests depending on whether the code can be accessed or not.

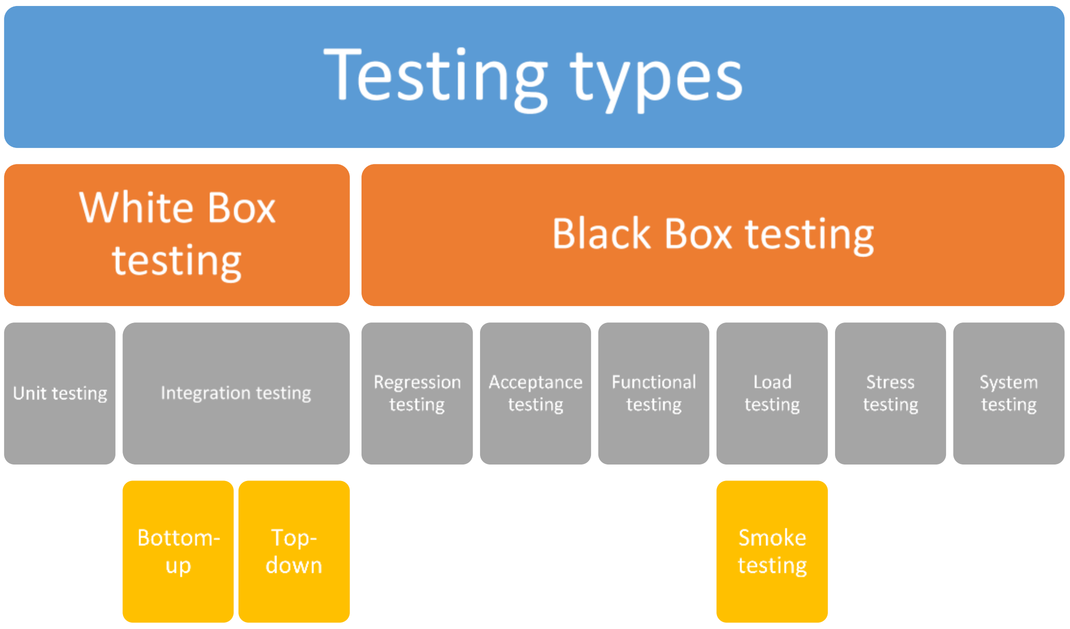


Diagram 1. Classification of testing types

3.3.1 White Box Testing

White Box Testing, also called Glass Testing or Open-box Testing, is referred to tests in which the code is known and visible, so it is possible to take a look to the code, examine the internal logic and find defects without executing it. The following figure shows a simple White Box Testing diagram.

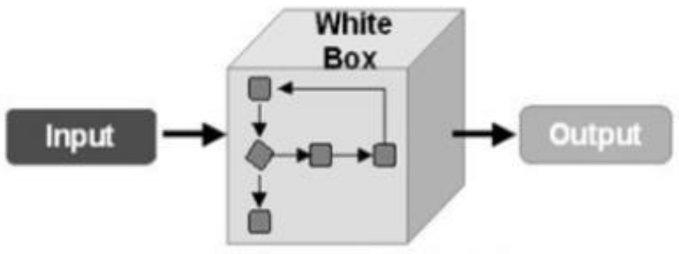


Figure 8. White Box Testing diagram [8]

With this type of testing, the tester has a complete knowledge of the code, so it is possible to make a full coverage test of the code, although look into every part of the code can be sometimes very difficult. Also, it is likely to find some lines of code that are never used or useless, so that lines can be removed, making the code easier to read.

For this type of testing a maintenance is required, so as the code evolves, testing tools has to adapt to new code.

3.3.1.1 Unit Testing

The Unit Tests are coded by the respective developers, who also will test its functionality before handing it to the QA Team, which will rerun the test with different test data.

The goal of Unit testing is to isolate small portions of the program in order to verify that no defects are found in the internal logic of the software under test as early as possible.

3.3.1.2 Integration Testing

The purpose of integration testing is to test functionality, reliability and performance requirements in a major portion of code at the same time, sometimes build from different unit tests.

These tests can be done in two ways:

Bottom-up

This testing is performed by combining different Unit tests starting from one and adding progressively more tests, ending with a big combination of test called modules or builds.

Top-down

Instead, this way, the test is started by testing a module/build and testing its subtests ending progressively by ending up with a unit test.

3.3.2 Black Box Testing

Black Box Testing, unlike White Box Testing, the code is unknown to the tester, so the way to test this code is to use the system as a normal user and provide different inputs that will lead to different outputs, which will be checked to verify that in all cases is the expected result. The following figure shows a simple Black Box Testing diagram

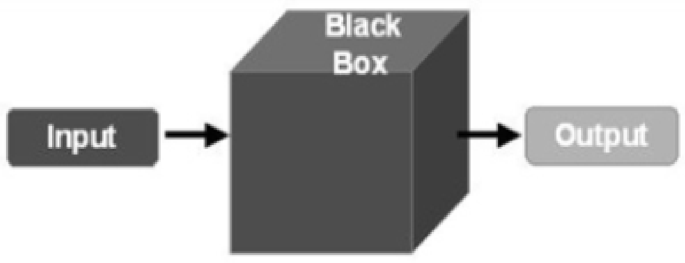


Figure 9. Black Box Testing diagram [8]

This type of testing has a limited coverage of the test, as tester can only perform a selected number of scenarios that are actually implemented. The fact that for these tests the code is unknown, a skilled tester on programming is not needed.

3.3.2.1 Regression Testing

As new functionalities or changes are applied to the software product, it is possible that some previous code stops working. This test is performed to avoid this and verifies that software previously developed and tested still performs correctly even after it was changed or interfaced with other software.

3.3.2.2 Acceptance Testing

This is undoubtedly the most important type of testing, since it is the final step in which it is determined by the QA Team that all customer requirements, specifications and contract are met or not.

A document of acceptance test cases is created, defining an acceptance criteria, from requirement and functionalities of the product. Also, after the execution of the test, a report of the execution is done, where all results are described and a conclusion of the product status.

To perform this test, the acceptance test cases are executed on a finished version of the software, which is required so every part of the product can be tested to check that the results acquired and the expected ones are the same. Those test cases execution can be performed either manually or using an acceptance test script and then comparing the obtained results.

3.3.2.3 Functional Testing

Functional testing usually describes what the system is supposed to do based on the specifications and requirements. This test is performed on a finished integrated system to evaluate if its requirements are fulfilled before release.

Although it is similar to the acceptance testing differs from it, since it is a product verification activity, whereas the acceptance testing is a validation activity that is usually performed with the cooperation of the customer.

There are other black box testing that are also functional testing such as: Smoke testing, Sanity testing, Regression testing, Usability testing.

3.3.2.4 Load Testing

Load testing is used to determine the behaviour of a software on normal and high peak load conditions in terms of software accessing and manipulating large input data. It is used to identify the maximum capacity of the software and how it works at peak time.

3.3.2.5 Stress Testing

Stress testing is similar to Load testing, but referring to software behaviour on normal and high peak under strange conditions such as lack of resources, turning server or database down.

3.3.2.6 System Testing

This test takes all integrated components that have passed integration testing and also the software system itself to detect any inconsistencies. It is tested in an environment as similar to production environment as possible and is performed very rigorously.

3.3.2.7 Smoke Testing

Smoke testing is a preliminary test that reveal simple failures, severe enough to reject a software release and not perform more tests until this test is passed.

It covers the basic functionalities of the application so it has to be executed and passed in each and every build in order to continue with the other types of testing.

3.3.3 Techniques

There are some techniques developed by Atlassian [9] that tries to make aware that the whole team is responsible for the quality of the project. Those techniques are designed both to find and prevent bugs from appearing. At last, the team will improve their programming skills as they will learn from their errors and will focus on bug prevention.

3.3.3.1 Blitz test

In a blitz test, people from the team (Developers, Technical writers, Product managers and also people not involved in the project) spend about an hour doing concentrated testing of specific features of the product, usually at the end of the Sprint. With this test, different developers of the team (including the ones that have developed the software under testing) would spend that time testing the targeted part of the software in order to discover some improvements and finds bugs.

3.3.3.2 DoT

To make everyone aware of the fact that quality is everybody's responsibility, there is a technique in which a developer is assigned to do testing. Putting a "Developer on Testing" or "DoT" helps developers to improve their skills and it also prevents the QA engineer to be the bottleneck of the team. So, here a different developer checks the work of his partner, after a meeting with the QA tester to ensure what is needed to test.

3.3.3.3 QA Kickoffs

This technique is performed at the beginning of the Sprint. It consists on QA and developers brainstorming together to think of all the risks that may have that task before they start implementing it. With the ideas that came out of this brainstorming, they choose how to implement their task, always following the requirements and preventing the problems from arising.

Although, developers have to be able to come up with potential problems, performing this techniques with a QA helps to encompass all fields.

3.4 Tools

In order to implement the QA process, a lot of useful tools have been developed by third parties. As the number of available tools is huge, only a small set of them is considered and explained in this document. The tools that are going to be explained, can be divided in three groups: Issue Trackers, Version Control Systems and Testing Tools.

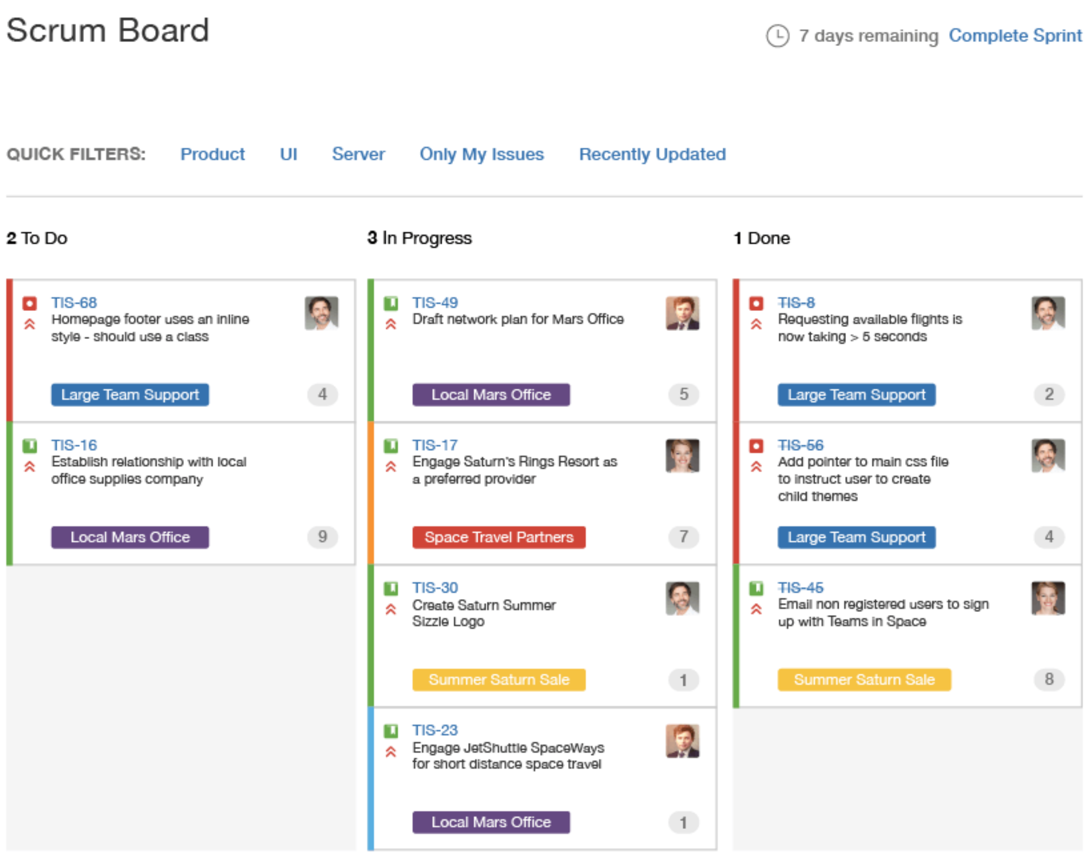
3.4.1 Issue Trackers

Issue trackers are programs that manage and maintain different types of issues. The users can report an issue as tasks, bugs, improvements, etc. and track its progression towards the resolution. Issue reports, usually, have additional useful information such as the person responsible to solve the issue, date of creation, priority, etc. that can be hid if it is considered unnecessary.

3.4.1.1 Jira

Jira is an issue tracking software developed by Atlassian [10]. It offers support for bug tracking, issue tracking and project management functions.

Jira allow users to manage agile projects easily. It allows to create scrum boards, with the state of the issues, its priority and the person assignee to the task, as seen on the figure 10.


Scrum Board

7 days remaining Complete Sprint

QUICK FILTERS: Product UI Server Only My Issues Recently Updated

2 To Do

- TIS-68 Homepage footer uses an inline style - should use a class — Large Team Support — 4
- TIS-16 Establish relationship with local office supplies company — Local Mars Office — 9

3 In Progress

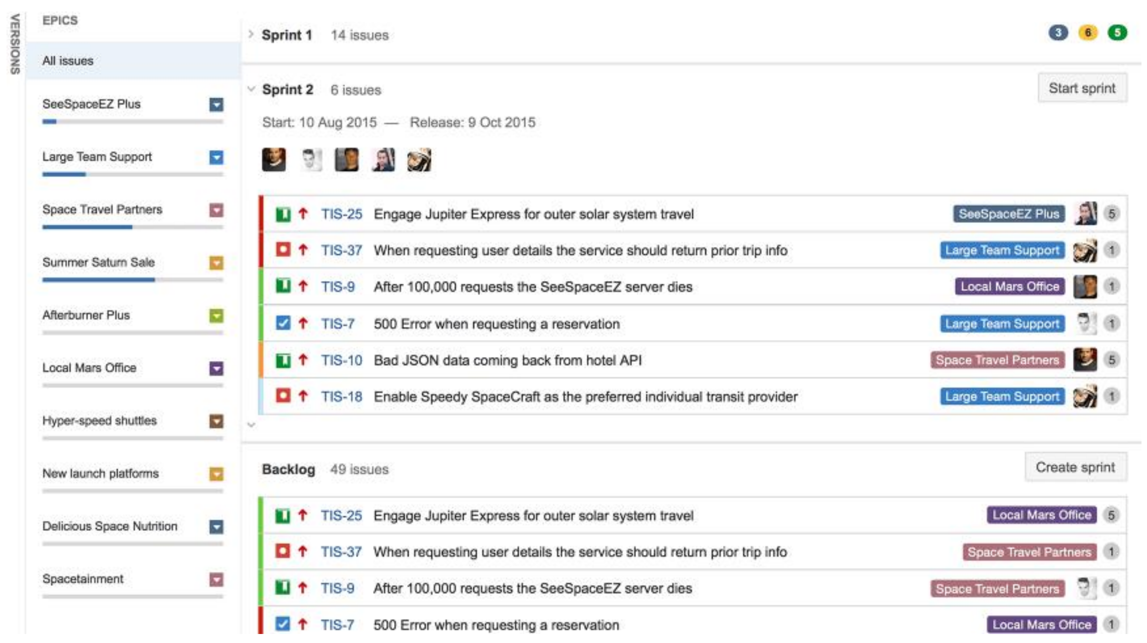
- TIS-49 Draft network plan for Mars Office — Local Mars Office — 5
- TIS-17 Engage Saturn's Rings Resort as a preferred provider — Space Travel Partners — 7
- TIS-30 Create Saturn Summer Sizzle Logo — Summer Saturn Sale — 1
- TIS-23 Engage JetShuttle SpaceWays for short distance space travel — Local Mars Office — 1

1 Done

- ~~TIS-8~~ Requesting available flights is now taking > 5 seconds — Large Team Support — 2
- ~~TIS-56~~ Add pointer to main css file to instruct user to create child themes — Large Team Support — 4
- ~~TIS-45~~ Email non registered users to sign up with Teams in Space — Summer Saturn Sale — 8

Figure 10. Scrum Board on Jira [11]

It also allows to write Scrum artifacts, such as product backlog or user stories, and planning sprints, see figure 11.


VERSIONS

EPICS

All issues

SeeSpaceEZ Plus

Large Team Support

Space Travel Partners

Summer Saturn Sale

Afterburner Plus

Local Mars Office

Hyper-speed shuttles

New launch platforms

Delicious Space Nutrition

Spacetainment

Sprint 1 14 issues 3 6 5

Sprint 2 6 issues Start sprint

Start: 10 Aug 2015 — Release: 9 Oct 2015

TIS-25	Engage Jupiter Express for outer solar system travel	SeeSpaceEZ Plus	5
TIS-37	When requesting user details the service should return prior trip info	Large Team Support	1
TIS-9	After 100,000 requests the SeeSpaceEZ server dies	Local Mars Office	1
TIS-7	500 Error when requesting a reservation	Large Team Support	1
TIS-10	Bad JSON data coming back from hotel API	Space Travel Partners	5
TIS-18	Enable Speedy SpaceCraft as the preferred individual transit provider	Large Team Support	1

Backlog 49 issues Create sprint

TIS-25	Engage Jupiter Express for outer solar system travel	Local Mars Office	5
TIS-37	When requesting user details the service should return prior trip info	Space Travel Partners	1
TIS-9	After 100,000 requests the SeeSpaceEZ server dies	Space Travel Partners	1
TIS-7	500 Error when requesting a reservation	Local Mars Office	1

Figure 11. Jira product Backlog [10]

So, Jira software has integrated multiple functionalities related to agile methodologies, specifically on the Scrum model. Even so, Jira does not have all functionalities that would be profitable to develop all tasks of the agile model, but fortunately has many plugins available to compensate, although some of them are for payment.

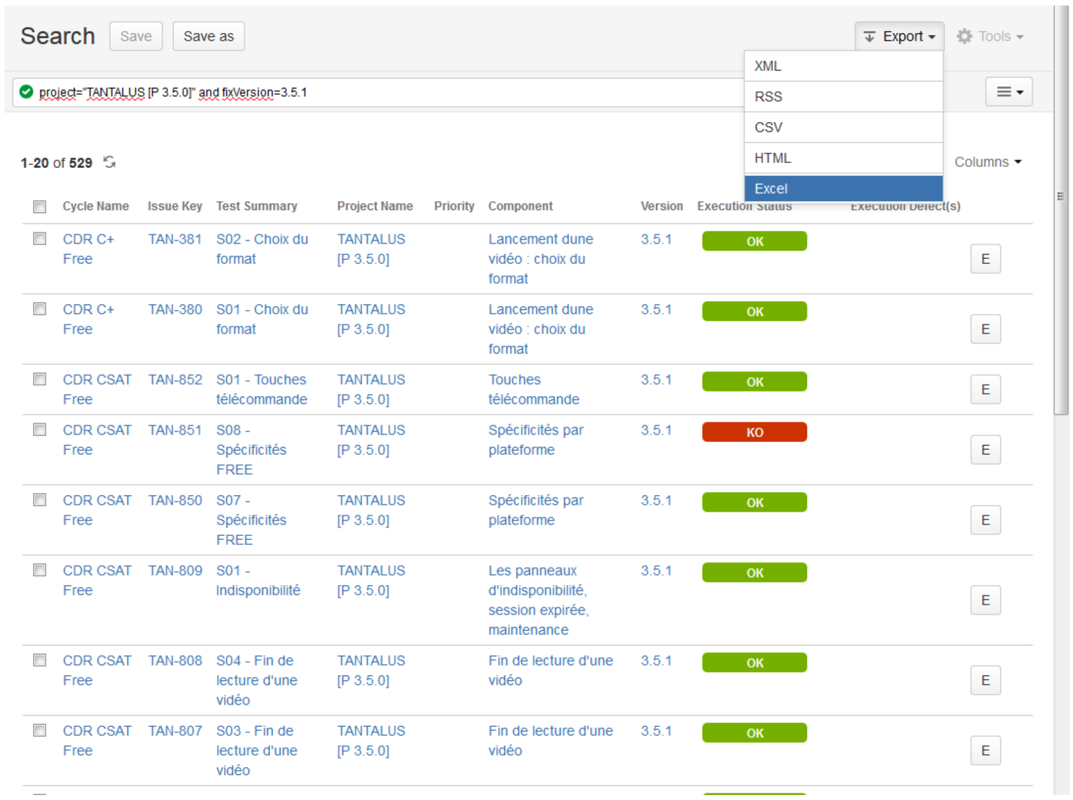
A pair of these plugins, which seem quite profitable, are explained below. These plugins are Confluence and Zephyr.

Confluence

Confluence is software from Atlassian [12], which integrates completely with Jira and allows users to create meeting notes, product requirements and knowledge base articles on the web so everyone on the team can see and contribute.

Zephyr

Zephyr is an Atlassian software [13] that integrates with Jira. Zephyr is a real-time test management that allows to create test cases and test cycles on the Jira platform. Test cycles are usually executed on every sprint or before a release. It includes a test case and test cycle finder, which can be seen on the figure 12, with filters to choose only the one that has defects or errors in order to add them to another test cycle to check is these defects are fixed.



project="TANTALUS [P 3.5.0]" and fixVersion=3.5.1

1-20 of 529

Cycle Name	Issue Key	Test Summary	Project Name	Priority	Component	Version	Execution Status	Execution Defect(s)	
CDR C+ Free	TAN-381	S02 - Choix du format	TANTALUS [P 3.5.0]		Lancement dune vidéo : choix du format	3.5.1	OK		E
CDR C+ Free	TAN-380	S01 - Choix du format	TANTALUS [P 3.5.0]		Lancement dune vidéo : choix du format	3.5.1	OK		E
CDR CSAT Free	TAN-852	S01 - Touches télécommande	TANTALUS [P 3.5.0]		Touches télécommande	3.5.1	OK		E
CDR CSAT Free	TAN-851	S08 - Spécificités FREE	TANTALUS [P 3.5.0]		Spécificités par plateforme	3.5.1	KO		E
CDR CSAT Free	TAN-850	S07 - Spécificités FREE	TANTALUS [P 3.5.0]		Spécificités par plateforme	3.5.1	OK		E
CDR CSAT Free	TAN-809	S01 - Indisponibilité	TANTALUS [P 3.5.0]		Les panneaux d'indisponibilité, session expirée, maintenance	3.5.1	OK		E
CDR CSAT Free	TAN-808	S04 - Fin de lecture d'une vidéo	TANTALUS [P 3.5.0]		Fin de lecture d'une vidéo	3.5.1	OK		E
CDR CSAT Free	TAN-807	S03 - Fin de lecture d'une vidéo	TANTALUS [P 3.5.0]		Fin de lecture d'une vidéo	3.5.1	OK		E

Figure 12. Searcher of Test on Zephyr for Jira [14]

Zephyr also provides tools to export the test execution data to various formats as is seen on the previous figure, and tools to generate graphics and statistics of tests execution, as will be seen on the following figure.

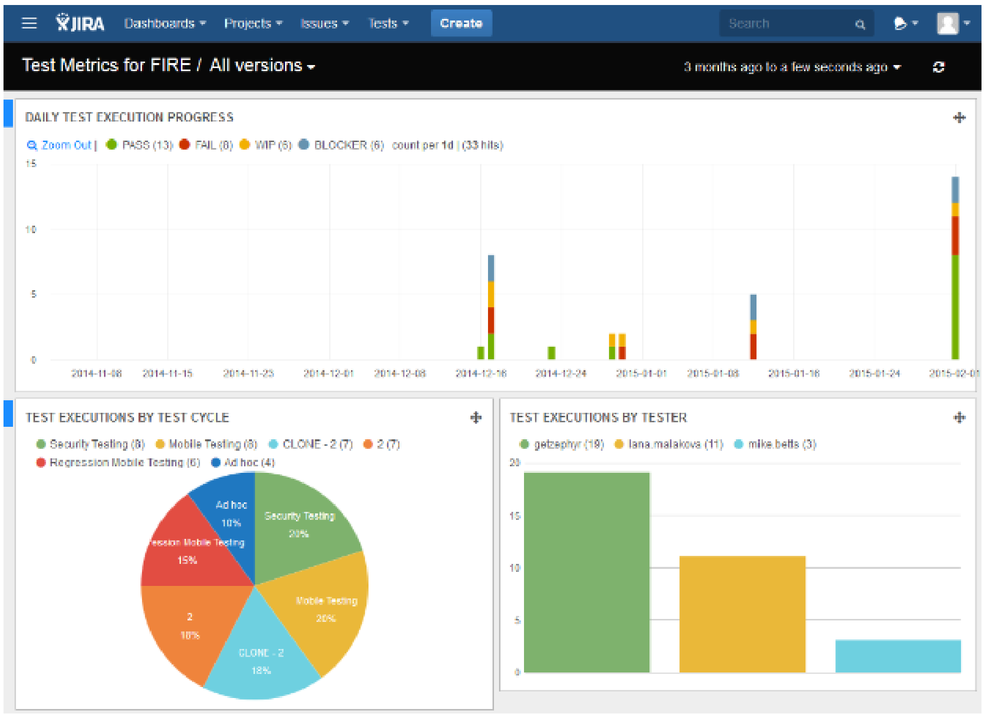

JIRA
Dashboards
Projects
Issues
Tests
Create
Search
Test Metrics for FIRE / All versions
3 months ago to a few seconds ago
DAILY TEST EXECUTION PROGRESS
Zoom Out | PASS (13) FAIL (8) WIP (5) BLOCKER (6) count per 1d | (33 hits)
2014-11-08 2014-11-15 2014-11-23 2014-12-01 2014-12-08 2014-12-16 2014-12-24 2015-01-01 2015-01-08 2015-01-16 2015-01-24 2015-02-01
TEST EXECUTIONS BY TEST CYCLE
Security Testing (8) Mobile Testing (8) CLONE - 2 (7) 2 (7)
Regression Mobile Testing (6) Ad hoc (4)
Ad hoc 10%
Security Testing 20%
Mobile Testing 20%
CLONE - 2 18%
2 18%
Regression Mobile Testing 15%
TEST EXECUTIONS BY TESTER
getzephyr (18) lana.malakova (11) mike.betts (3)

Figure 13. Graphics and Statistics of Test executions [15]

3.4.1.2 Bugzilla

Bugzilla [16], as Jira, is an issue tracker system developed by Mozilla Foundation. The main difference between Jira and Bugzilla is that Bugzilla is free, but the main functionalities used for bug tracking are also implementing.

As an issue tracker Bugzilla does:

- Track bugs and code changes
- Communicate with teammates
- Submit and review patches
- Manage QA

Figure 14. Bugzilla main page [17]

3.4.2 Version Control Systems

Version Control Systems, or VCS, are programs that manage file changes. They store an historic of changes with information of the committed changes, so it is easy to return to a previous version of a file if it is necessary.

3.4.2.1 Git

Git [18] is a free and open source distributed version control system. It is designed to handle everything related to projects, whether large or small, with speed and efficiency.

Git is designed to work in projects with a few people involved. A Git repository uses branches to distribute work in progress and every repository has a master branch whose purpose is to store the stable code version or release candidate. The uses of creating branches are not restrained at all, so branches can be created from any other branches and merged.

In Eurecat, a workflow of branches has been defined as follows. From master a develop branch is created and from this one other branches will be created for developing concrete functions and then merged again to develop. After the required functions are integrated on the develop branch it is created a Release branch, where the software will be tested to check that fulfill the requirements and then merged to Master. Also, if an urgent error is found on Master, a Hotfix branch may be created in which the error will be solved and then merged to Master again. A branch structure of this workflow can be seen in figure 15, and detailed information of the Eurecat git workflow is attached in the Annex section.

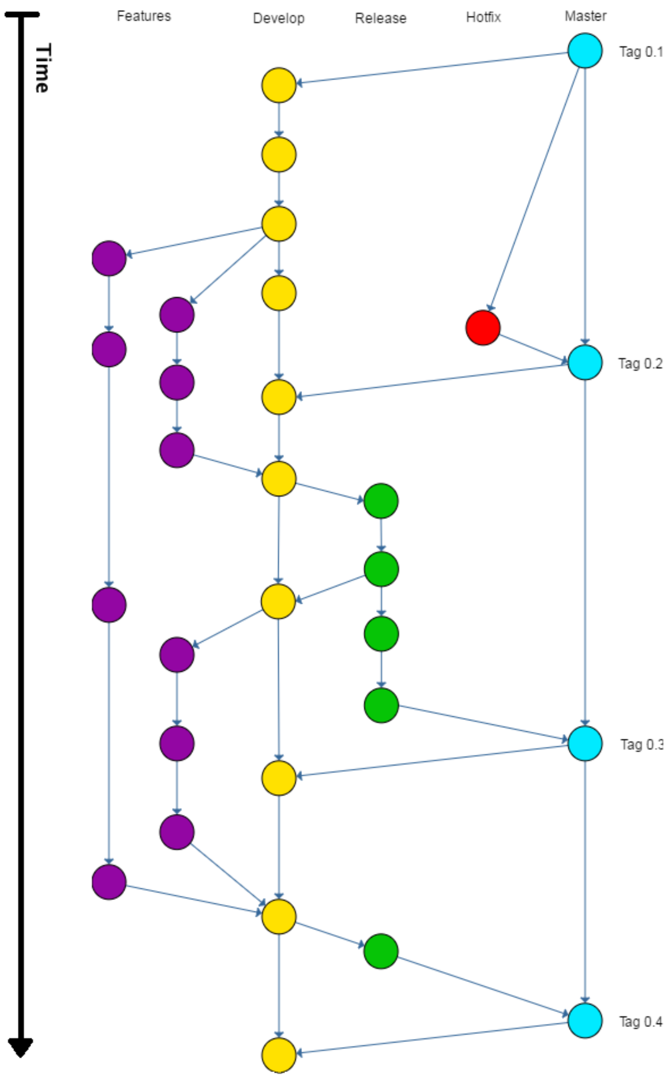


Figure 15. Structure of Git branches used in Eurecat

Git makes easy the creation, merging and deletion of these branches. Performing these actions only takes a few seconds. Branches can be created locally or distributed, normally the correct use of git is the developer creating a local branch to perform their task and then merge it to a distributed branch.

Most Git operations are formed locally, so changes can be pushed to the remote server only when it is necessary, this gives Git a speed advantage in relation with other centralized systems that have to continually communicate with the server.

Git has been built with speed and performance from the beginning. It is written in C, decreasing the overhead of runtimes related with high-level languages.

Git is distributed, which means that instead of performing a checkout a clone of the complete repository is done, and allows the user to have multiple Backups. So every user has a full backup of the main server. Each of these local copies, once the work is done, had to be pushed up in order to update the main server.

Git ensures the cryptographic integrity of every bit in a project. A checksum of everything that is committed is done and is recovered by its checksum when checked back out. This way it is not possible to change any file or any other data in a Git repository without changing the IDs of everything after it.

Contrasting the other systems, Git has something that is called the "staging area" or "index". This is an intermediate area where commits can be formatted and reviewed before completing the commit. In the figure 16, it is shown the commands required to stage the data from the working directory, which has to be a git folder, and then the command to commit only the staged files.

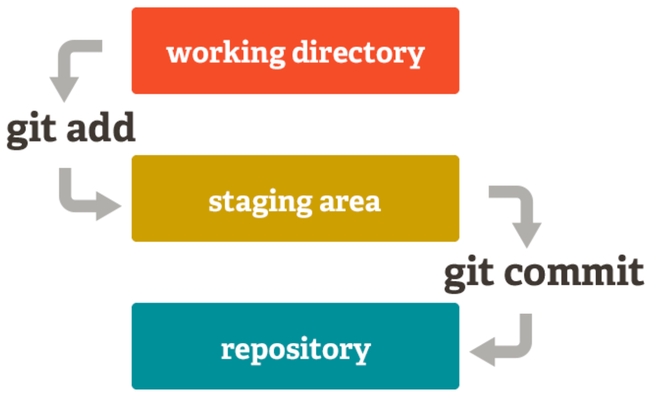


Figure 16. Git Staging area commands [19]

3.4.2.2 SVN

SVN (Apache Subversion) [20] is an open source version control tool, based in a repository which works very similar as a file system.

SVN uses the revision concept to store the changes produced in the repository. Between two revisions only the set of modifications (delta) is saved, optimizing the use of the disk space. SVN allows the user to create, copy and delete files with the same flexibility as if it were on their local hard drive.

SVN can access the repository using the network, allowing to be used by several people on different computers. The ability for multiple people to modify and manage the same set of data on their computers encourages collaboration. It is possible to progress faster without all the modifications going through a single pope. As the project is under version control, there is no reason to fear that quality will be affected, if an incorrect change is made, it can simply be undone.

3.4.3 Testing Tools

There are a lot of useful testing tools for every programming language, so this document explains only a few of them. Some of the testing tools use Mock objects. Mock objects are used in object-oriented programming and are fake objects that copy the behaviour of real objects. They are used when the real object is impossible to include into a unit test in order to perform tests as if the object was operating correctly.

These tools are very useful as, if properly programmed, they help to detect errors or misbehaviour of the program as its purpose to perform executions by simulating the behaviour of the project in the environment in which it will be released in the real world.

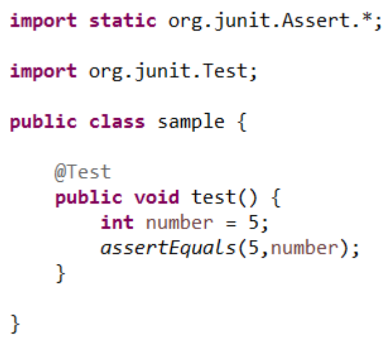
It is very important that tests are correctly developed, as an error evaluating outcomes of the executions may result on important bugs or problems in the product. So after executing the tests, if any error is detected in the development of the test case it is mandatory to be resolved.

Although those tools can be classified by the type of testing it is mainly designed, some of them can be used for more than one type of testing. For this reason it has been decided to explain the different tools on this section. Also Zephyr for Jira, explained above, is not in this section as it really helps to manage the tests, but the results of the execution are manually defined by the user, instead, the following described tools automatically return the result of the test depending on the execution.

3.4.3.1 JUnit

JUnit [21] is an Open Source Unit-testing framework for the Java Programming Language. It lets developers to build repeatable tests to measure progress and detect unexpected behaviour or defects. JUnit use assertions to determine if a test has passed or not, which implies that once finished executing a test, no subjective human interpretations of results are necessary.

On the next figure, a basic example of a JUnit Test Case, which assertion will return always true, can be seen.



```java
import static org.junit.Assert.*;

import org.junit.Test;

public class sample {

    @Test
    public void test() {
        int number = 5;
        assertEquals(5,number);
    }

}
```

Figure 17. Example of JUnit Test Case

3.4.3.2 NUnit

Nunit [22] is an Open Source Unit-testing framework for .Net software. It was taken from JUnit (see below) at the beginning, but the current production release has been completely rewritten and supports a wide range of .NET platforms.

3.4.3.3 pytest

Pytest [23] is a Python testing framework used with small tests performing plain assertions to check its correct functionality. So, as JUnit, no subjective human interpretations of the results are required.

3.4.3.4 DBUnit

DBUnit [24] is a JUnit extension used for testing databases. Between tests executions this framework puts the database into a known state in order to avoid problems executing multiple tests. With DBUnit it is possible to import and export the database data from and to XML datasets.

3.4.3.5 HttpUnit

This framework is usually used in combination with JUnit. HttpUnit [25] is used for testing web applications or simply testing a web-site. It is written in Java and emulates the browser behaviour, testing pages returned as text, XML DOM or containers of forms, tables and links.

With HttpUnit download is included ServletUnit, which allows to test servlets without a servlet container. The main class on the HttpUnit is the WebConversation, which acts as a browser. The following figure shows how to use it to do a request to a web page.

```
public void getGooglePage() throws Exception{
     WebConversation conv = new WebConversation();
     WebRequest request = new GetMethodWebRequest("http://www.google.com");
     WebResponse response = conv.getResponse(request);
}
```

Figure 18. Example of request with HttpUnit

3.4.3.6 SoapUI

SoapUI [26] is an open source web service testing platform for SOA (Service-Oriented Architectures) and REST APIs. It offers support for functional testing, Web Service Description Language coverage, message assertion testing and test refactoring. SoapUI has also a professional version which includes more features such as e-mail support, forum support, maintenance, Groovy code templates, among others features.

3.4.3.7 Selenium

Selenium [27] is a suite of tools used for browsers automation. Its main function is for automating web applications for testing purposes, but it can be used to automate any action performed in the browser. Selenium has two main parts:

- Selenium WebDriver: It is the successor of Selenium Remote Control and is used to create robust, automated regression tests and scale and distribute scripts across many environments.
- Selenium IDE: It is used for creating quick bug reproduction scripts recording and playback a set of interactions within the browser. As shown in the Figure 19, it has an intuitive environment, which is displayed in a different window and the actions performed in the browser will be recorded into Test Cases.

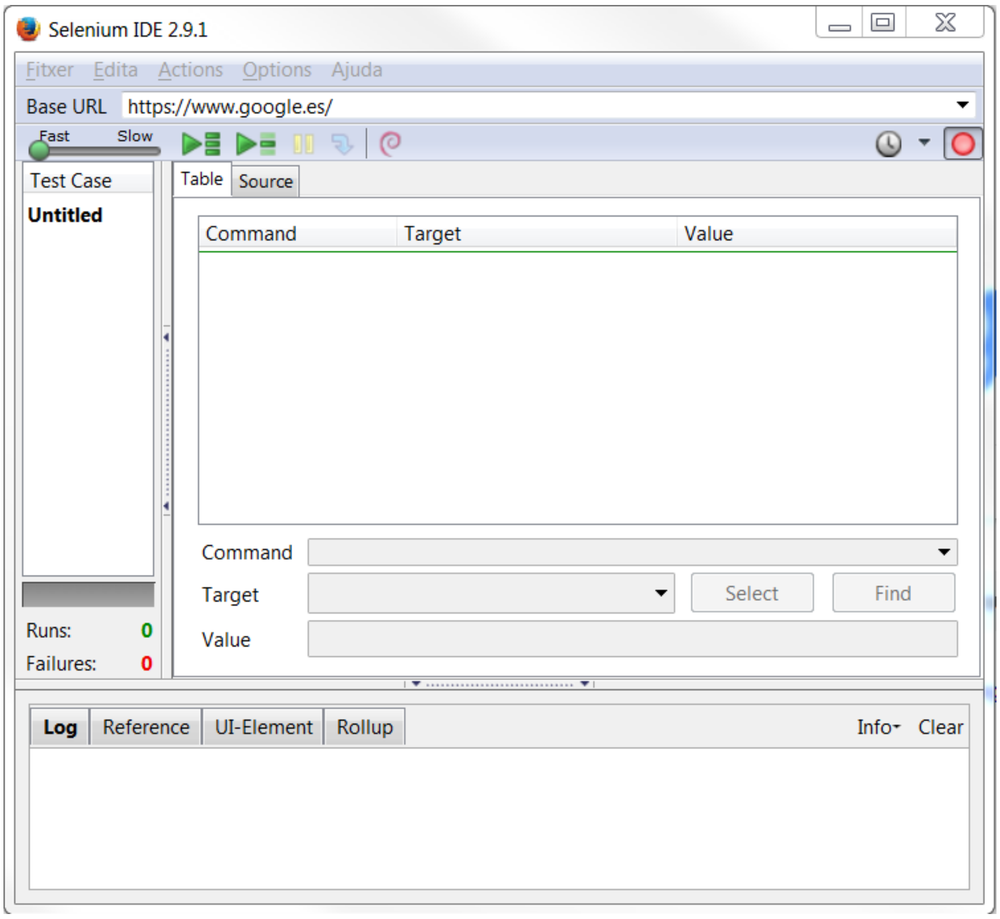


Figure 19. Selenium IDE

3.4.3.8 Watir

Watir [28] is an open source Ruby library for automating tests. It is based on selenium so it also interacts with a browser to record and playback the actions performed and its main feature is for automating web applications in Ruby.

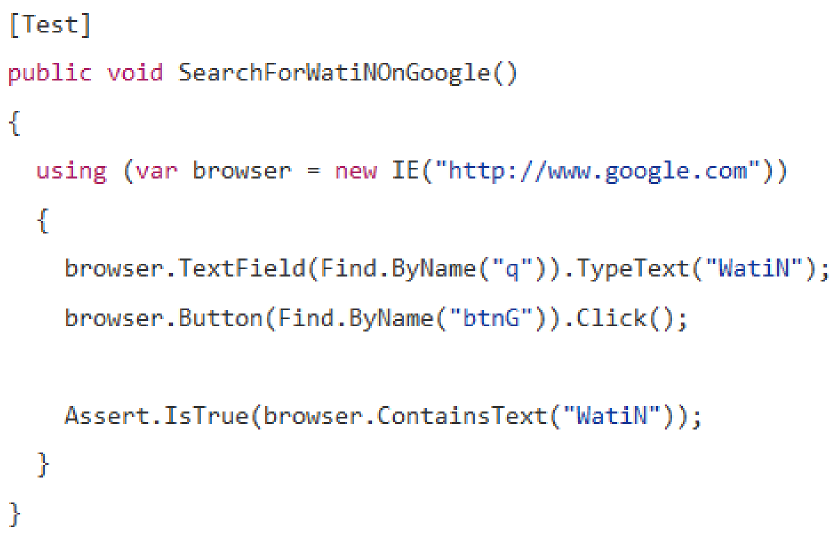
```ruby
browser = Watir::Browser.new :chrome

browser.goto 'google.com'
browser.text_field(title: 'Search').set 'Hello World!'
browser.button(type: 'submit').click

puts browser.title
# => 'Hello World! - Google Search'
browser.quit
```

Figure 20. Basic script example in Watir [26]

On the previous figure can be seen a basic script example of a search on google using Watir.

3.4.3.9 Watin

Watin [29], inspired by Watir, performed its own library for automating tests, but this time in Net languages. Its main feature is the automation of web applications in Net language. A basic example of a search in google using the Watin script can be seen below, in figure 21.



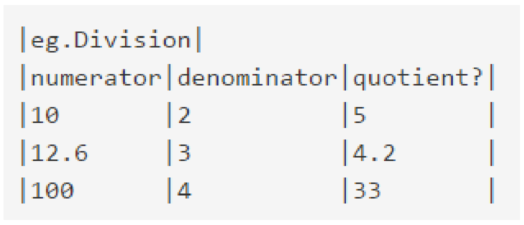
```
[Test]
public void SearchForWatiNOnGoogle()
{
  using (var browser = new IE("http://www.google.com"))
  {
    browser.TextField(Find.ByName("q")).TypeText("WatiN");
    browser.Button(Find.ByName("btnG")).Click();

    Assert.IsTrue(browser.ContainsText("WatiN"));
  }
}
```

Figure 21. Example of web test automation: searching on Google [29]

3.4.3.10 FitNesse

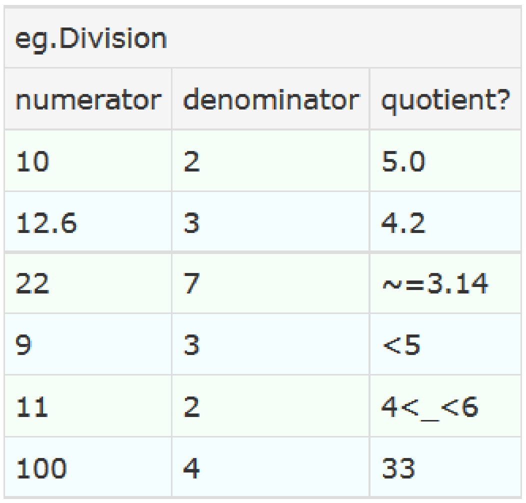
FitNesse [30] is an open source testing tool, based on a wiki web server, which means that everyone can contribute to improve it. It is an easy to use tool that allows non-technical users to specify and run acceptance tests for software systems. In FitNesse to create a test case is needed to create a wiki page, because the wiki pages are the tests themselves.

On FitNesse, the tests are built as tables of input and expected output data, called Decision Tables. Those Decision Tables are created with a wiki page code, with vertical bars, which delimit table cells, a basic example of Decision Table creation and visualization can be seen below, on figures 22 and 23.



```
|eg.Division|
|numerator|denominator|quotient?|
|10       |2          |5        |
|12.6     |3          |4.2      |
|100      |4          |33       |
```

Figure 22. Wiki code for create a decision table [31]



eg.Division		
numerator	denominator	quotient?
10	2	5.0
12.6	3	4.2
22	7	~=3.14
9	3	<5
11	2	4<_<6
100	4	33

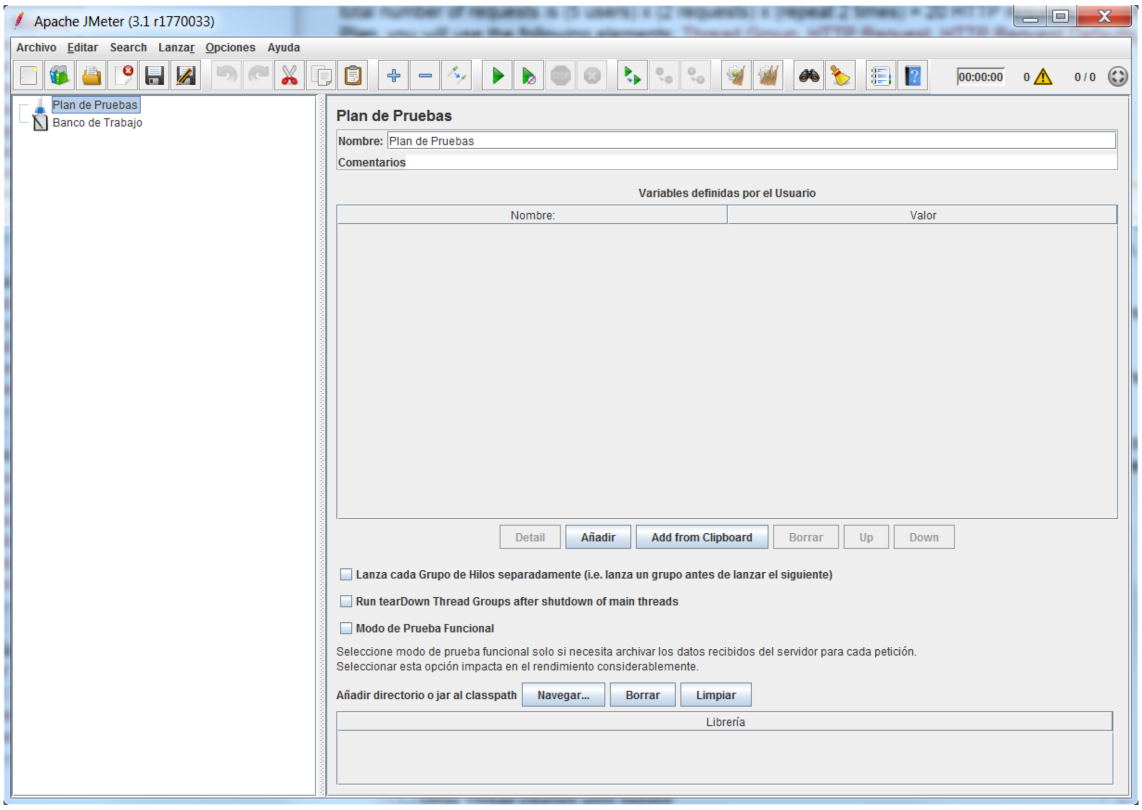
Figure 23. Example of FitNesse Decision Table [31]

In a Decision Table, each row represents a full scenario of inputs and outputs. Concretely in the previous example of figures 22 and 23, the numerator and denominator columns are the inputs and the question mark in quotient indicates to FitNesse that this column is for the expected outputs. In figure 22, in the last row, the expected output for the division is incorrect, this error has been made on purpose since it is possible to introduce this type of errors unintentionally while the test case is developed, so after executing this test, the result obtained (25) is different from the expected result and this test case will fail.

3.4.3.11 JMeter

JMeter [32] is an open source software built completely in Java by The Apache software foundation. It is designed to perform load testing on services, focusing on web applications. This framework allows multi-threading to run tests concurrently.

JMeter also has two modes, a GUI mode that should only be used to create test scripts, which can be seen below in figure 24, and a NON GUI mode that has to be used for load testing.



Nombre: Plan de Pruebas
Comentarios

☐ Lanza cada Grupo de Hilos separadamente (i.e. lanza un grupo antes de lanzar el siguiente)
☐ Run tearDown Thread Groups after shutdown of main threads
☐ Modo de Prueba Funcional

Seleccione modo de prueba funcional solo si necesita archivar los datos recibidos del servidor para cada petición.
Seleccionar esta opción impacta en el rendimiento considerablemente.

Figure 24. JMeter GUI window

3.4.3.12 Postman

Postman [33], as JMeter, is designed to perform load testing on different services and testing web applications. Moreover, it is used not only for testing, but also to API develop.

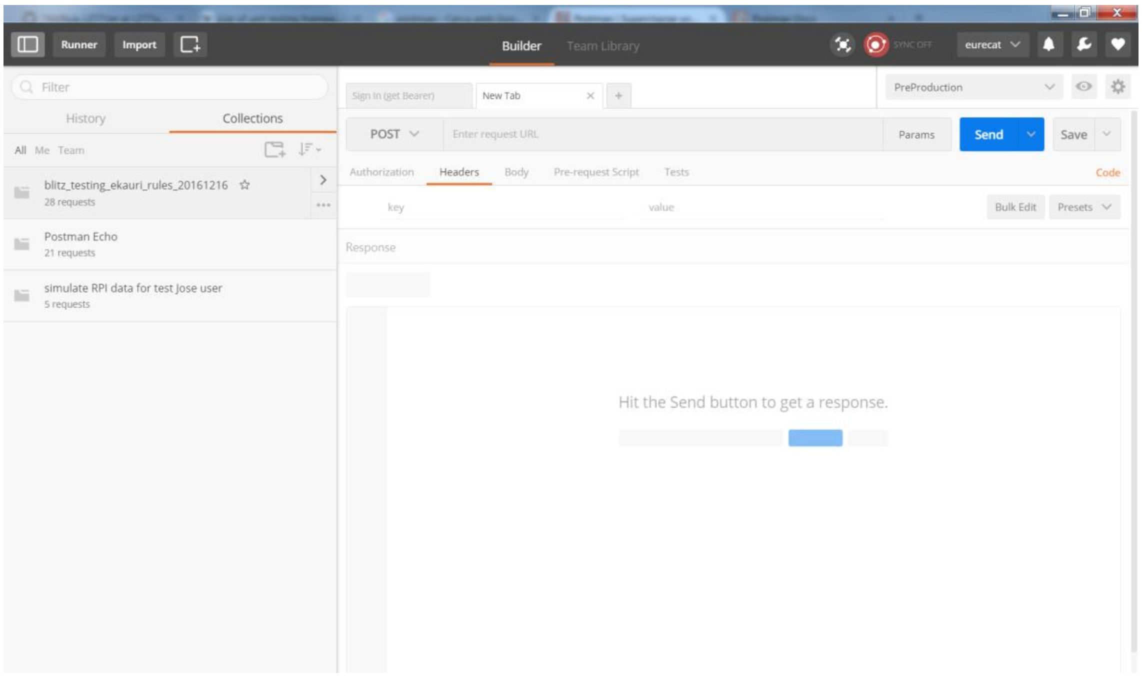
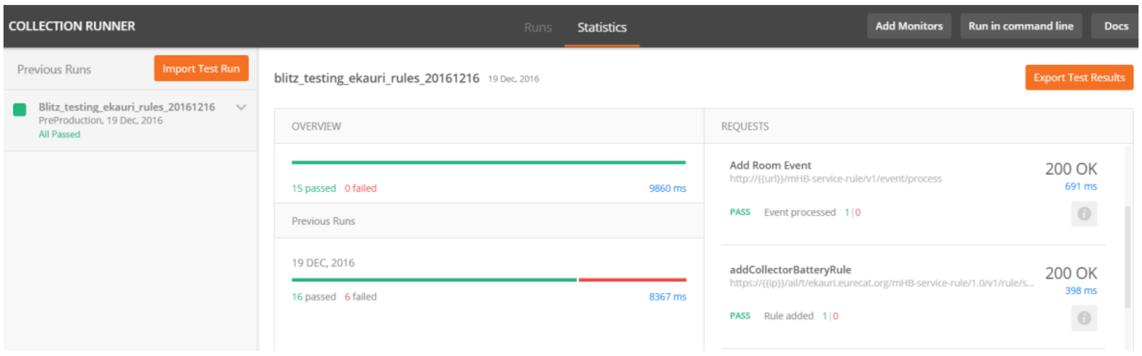


Figure 25. Postman GUI

Postman provides a GUI, which can be seen on the figure above, where different methods can be created to test remotes services. With that it can be implemented different methods such as POST, GET, PATCH, DELETE, define to which URL it will executed and the different parameters that can be used or needed, such as headers, the body of the request and also includes Pre-request Script that will be executed just before the request and Tests scripts, that is what concern to this document, that will be executed after the request. With this it is possible to concatenate different requests, setting environment and global variables.

After designing a set of requests with tests scripts, postman has a test Runner, that will executed all selected requests in order and give back a report of all the test executions, as it can be seen on the following figure.



Blitz_testing_ekauri_rules_20161216
PreProduction, 19 Dec, 2016
All Passed

blitz_testing_ekauri_rules_20161216 19 Dec, 2016

15 passed 0 failed 9860 ms

Previous Runs

19 DEC, 2016
16 passed 6 failed 8367 ms

Add Room Event
http://{{url}}/mHB-service-rule/v1/event/process
200 OK
691 ms
PASS Event processed 1 | 0

addCollectorBatteryRule
https://{{ip}}/ail/t/ekauri.eurecat.org/mHB-service-rule/1.0/v1/rule/s...
200 OK
398 ms
PASS Rule added 1 | 0

Figure 26. Postman Runner. Execution overview.

On the figure 26, on the request side, the URL has an environment parameter {{url}} that is changed before the execution. This is used to switch between environments without having to duplicate or change all the requests.

3.4.3.13 Httperf

Httperf [34] is a load testing tool which is used to measure the performance on web servers. It can generate several HTTP workloads and measure server performance easily. Httperf is a tools used to measure server performance generating a HTTP workload on the server that simulates real world behaviour.

Httperf also allows to build micro and macro level benchmarks. Three remarkable characteristics of this tool are support for the HTTP/1.1 and SSL protocols, the ability to generate and sustain server overload and its extensibility to new workload generators and performance measures.

4 QA Application

As seen on the QA State of the Art section in the previous pages, there are a wide variety of tests types and tools. So, with the information gathered it will be tried to implement the ones that best suit Eurecat in order to obtain better software quality, decreasing the number of bugs and do work more efficiently, which implies that software released will result in decreasing costs for the center at the end.

4.1 Center Description

In this section is described briefly the Eurecat center and the team, products, environments and software involved in this project.

Eurecat is the main technologic center of Catalonia, as a result of the integration of various companies such as Ascamm, Barcelona Media, Barcelona Digital, Cetemmsa, and CTM. Eurecat gathers the experience of over 450 professionals and give service to more than a thousand of companies.

4.1.1 Team

To check the viability of the QA process in Eurecat, it has only applied in the eKauri and eKenku projects, and if the result is positive it will be implemented later in other projects. So the team that will be described will be only the members of the eKauri/eKenku projects.

The team to develop these projects consists of seven people with roles in the scrum model, which is being used to develop both projects:

1. The Product Manager, who is in charge of finding new customers, creates and prioritizes User Stories.
2. The Product Owner, who is responsible for distributing and assigning the tasks helping the Product Manager and is now also playing the role of developer carrying out tasks related to Android development since the Android developer of the project left the center.
3. The Development Team, made up of four developers, but one of them left Eurecat in September, so, as stated above, the Product Owner is temporarily replacing their role. The three remaining developers are each specialised in:
    a. The backend.
    b. The frontend and videoconference.
    c. The mining service.
4. The QA Manager, who is in charge of the QA process.

4.1.2 Products

Among all Eurecat products, the two of them concerning the QA process will be explained. The products are eKauri and eKenku and both are from the eHealth section.

4.1.2.1 eKauri

eKauri is an ambient assistive living platform (AAL), which integrates different types of environmental sensors into a low-cost micro-architecture that is capable of transmitting data remotely using a wireless connection.

The set of environmental sensors includes temperature, humidity, luminosity, presence and gas alert. All of these sensors are cheap, small, non-intrusive and wireless z-wave, and continuously send the retrieved data.

The idea of eKauri is to help the elderly and support active aging, with two main goals:

1. Monitor people that requires assistance at their home.
2. Trigger alarms and emergency situations to the carers, and reporting them about user's activities.

The eKauri architecture is mainly composed by four components:

1. Home: Where a set of sensors would be installed.
2. Middleware: It is divided into independent web services that provide scalability, Interoperability and secure interconnection among all the components.
3. Intelligent monitoring system: It is continuously analysing and mining the data retrieved by the sensors.
4. Healthcare center: Using the web application platform, receives notifications, summaries, statistics of the remote monitored users.

The following figure shows an example of the eKauri installation. There is a housing model with the different zones where the environment sensors will be installed.

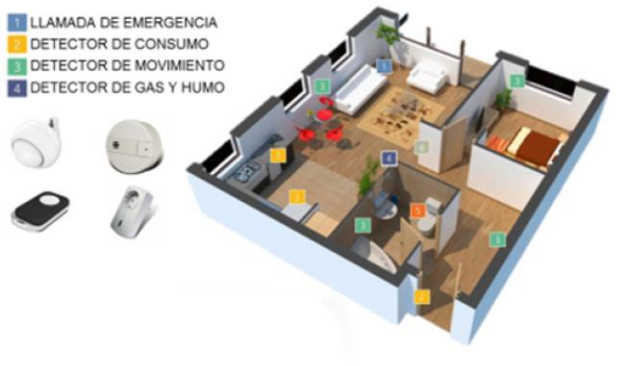


Figure 27. eKauri installation on a house model

4.1.2.2 eKenku

eKenku is a friendly mobile application designed to provide advanced remote assistance to support active aging. The eKenku set is composed by a tablet, which is configured to use a kiosk mode that prevents the user from performing actions outside the application, so users will only interact with the eKenku application and the measures devices: scale, tensiometer, pulse oximeter and glucometer. All these devices communicate with the eKenku application using a Bluetooth connection.

The eKenku idea is to remotely monitor the health status of chronic patients to trigger preventive, predictive and self-management interventions. In the latest version, eKenku includes a videoconference feature, where caregivers can call patient if necessary.

As eKauri, the eKenku architecture is composed by four main components, which functions are very similar to eKauri:

1. Home: A tablet with eKenku app installed, in kiosk mode, and the measurement devices that let the user to take their own physiological measures.
2. Middleware: It is divided into independent web services that provide scalability, Interoperability and secure interconnection among all the components.
3. Intelligent monitoring system: It is continuously analysing and mining the data retrieved by the devices.
4. Healthcare center: Using the web application platform, receive notifications, summaries and statistics from remote monitored users.

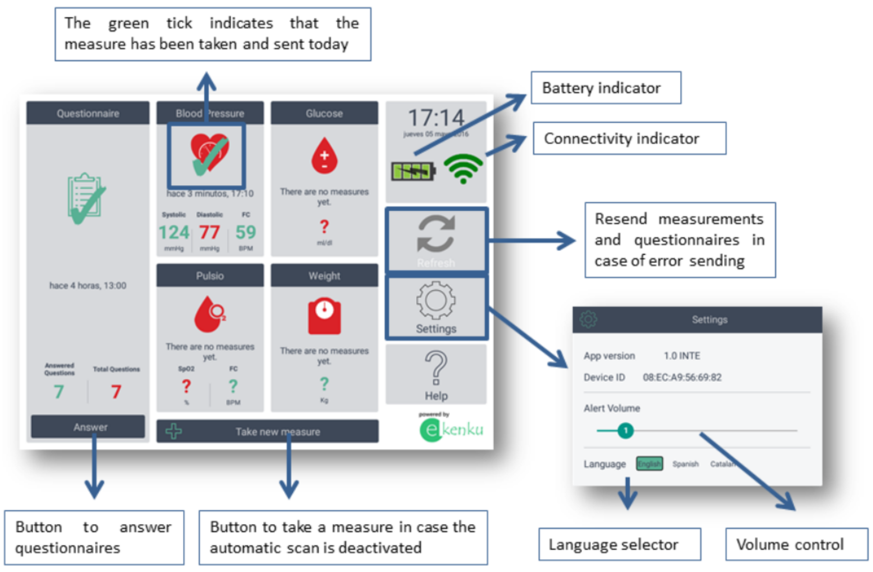

The green tick indicates that the measure has been taken and sent today
Battery indicator
Connectivity indicator
Resend measurements and questionnaires in case of error sending
Button to answer questionnaires
Button to take a measure in case the automatic scan is deactivated
Language selector
Volume control
Questionnaire
Blood Pressure
Glucose
17:14
Pulsio
Weight
Refresh
Settings
Help
Answer
Take new measure
eKenku
App version 1.0 INTE
Device ID 08:EC:A9:56:69:82
Alert Volume
Language English Spanish Catalan

Figure 28. eKenku help activity of the mobile application

In the previous figure, it is seen the help of eKenku with information about the use of the main activity. As stated above the application is easy to use, since the patient would only have to answer questionnaires if they are assigned and taking measures. To take measurements if automatic scan is disabled, it is necessary to click on the “Take new measure” button and take measurements with the external devices or if automatic scanning is enabled, the patient should only perform the measurements with the devices. In both cases, once a measurement is taken, the patient must accept it in the application.

4.1.3 Environment

The environment used in Eurecat for those projects consists of three servers: Integration, Preproduction and Production. The Integration and Preproduction are internal servers only accessible from the local network of the center or using a VPN (Virtual Private Network).

The Integration server is used to develop the new functionalities and integrate them into the system, and then check that they work correctly. After that, those features are transferred to the Preproduction server.

The Preproduction server was initially used to check the software with all the new integration features with an environment similar to the real world before transferring it to the Production server. Although, after starting the QA process, this server is used to store a stable version of the code where all QA tests and techniques are executed since it behaves similarly to the Production server.

The Production server stores the software that is released and delivered to the customers. This server is accessible through the network, since the tablets have to be able to communicate with it.

4.1.4 Software

In order to develop eKauri and eKenku, it has been decided to use different software for some parts. Different Git repositories have been used to store the code of the application and different services, using the defined Git workflow that is attached in the annex. Jira, which integrates with Scrum development model, has been used as an issue tracker, defining Sprints of two weeks.

The dashboard has been programmed with AngularJS and CoffeeScript. AngularJS is designed to extend the static view of the HTML to a dynamic view and CoffeeScript is a little programming language that compiles into javascript files.

Each one of these two projects has access to two servers, one with a WSO2 system installed and the second one with the database and the mining service. Within these servers the different parts of the software are defined.

In the WSO2 server, it is installed:

- API Manager (AM): It is a traffic manager that manages and scales the API traffic. It also supports API design and publishing as well as API management. The APIs installed here are developed with Java.
- Application Server (AS): It allows to host, deploy and manage various applications. There are installed services, which have been programmed with Java.
- Identity Server (IS): It connects and manages multiples identities through applications, APIs, the cloud, etc.
- Enterprise Service Bus (ESB): It mediates and transforms messages across systems.

In the database and mining server, it is installed:

- The database, which is written in the MySQL language.
- The mining service, which receives and redirects requests to the different services, and is programed in python.

In the future, the mining service will be transferred to the WSO2 server and reprogrammed in Java, as the current mining on python has given a lot of problems. Also, on the database server it will be installed Elastic search and XMPP.

4.2 Methodologies

In the theoretical section of this document, a study of the QA process has been carried out to know what types of tests exist, which ones are the most suitable to implement in the selected projects and which tools to use. So, in this section, the tests and tools chosen are briefly explained:

4.2.1 Acceptance tests

In order to know the initial state of the projects, acceptance tests were performed manually, using the applications as a normal user, checking all possible functions and verifying the result corresponding with the expected output of each feature. To perform these tests, Jira and Zephyr have been used to define different test cases and execute a test cycle to be able to track the execution result if necessary.

4.2.2 Unit and Integration tests

To test the correct procedure of the code functions, we have chosen to use JUnit to define unit tests, since the most modules are programmed in Java and gives us knowledge of errors at a logical level. As explained in previous sections, various unit tests meet for integration testing. So with these unit tests we can run both unit and integration testing.

4.2.3 Integration and load tests

To test the services, Postman software has been chosen to perform integration and load testing on the Web platform. For this, a series of requests have been prepared with scripts before and after the petition itself. Once these requests are defined, the Collection Runner can be run to execute load tests, executing all the requests the desired number of times.

4.2.4 Blitz technique

In the blitz test, some developers came to test the chosen parts of the product. This test can be performed using some tools or manually by having the attendees interact with the product. In this case, this test has been done using both the postman tool and interacting with the product dashboard.

4.3 QA Application in real projects

As it is said in the previous section, once it is known what tests and tools are going to be used for each test, it can already be performed. In this section, the methodologies and tools used to implement the QA process are described.

4.3.1 Initial work

Before starting this QA project at the center, first a Quality Control process was performed. In this process, applications were tested roughly to find bugs as a customer, checking from visual mistakes to functional misbehaviour.

To do this, at the beginning, tests for different functionalities and features were defined on Zephyr using an established naming to make it easier to reference. The naming structure was the following:

TCXyyy – Name of the test

Where **"TC"** refers to Test Case, **"X"** refers to the first letter in a subgroup of the application (i.e. Settings, Register, Battery, etc.), **"yyy"** refers to the number of the test in that subgroup starting from 001 and finally a brief name of what the test consists of. Therefore, TCR001-Register Patient, for example, will mean the first Test Case of the registration subgroup, which consists of registering a patient.

Then, at the end of every sprint, a test cycle is performed executing the concerning tests to ensure new functionalities, features and maybe corrections in that sprint were performed correctly and executing a complete test cycle before a release is done, so the product arrives to the client with the minimum numbers of bugs possible (ideally zero).

With Zephyr, which is fully integrated into Jira, the executions of the test cycles can be easily tracked, as well as tasks of improvements or bugs can be created directly. When a new task is created, it can be assigned directly to the person responsible for fixing it or, in the case of Eurecat, to the Product Owner who, after talking about priority, timing, etc., will assign the task to the developer and the sprint in which it will be developed/resolved

The priority applied on a bug is defined in Eurecat as follows:

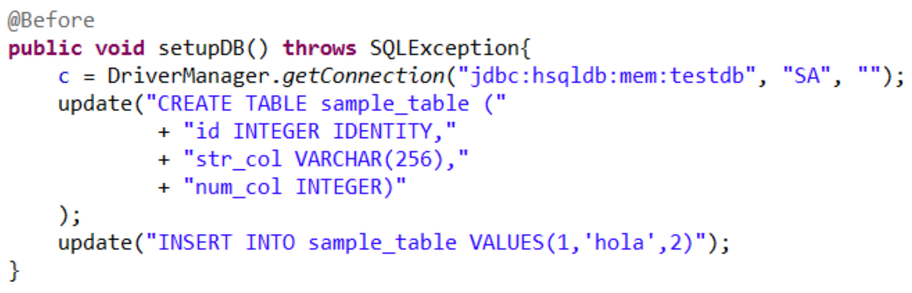
1. Blocker priority: Bugs that for some reason block the testing or make unable to check the functionality.
2. Major priority: Refers to serious errors that cause the application to crash, loss of data and big functionality errors.
3. Minor priority: Refers to small simple errors, which most certainly the client will not notice, but has to be fixed.
4. Trivial: Aesthetic errors that do not affect the functionality, translation errors.

After the execution of a couple of test cycles, on the Sprint meeting, a report about the status of the project is done, with information about the number of bugs found, the priority of them and comparison of error evolution between previous versions. An example of a report is attached in the annex.

4.3.2 JUnit

To start with test automation, JUnit testing of different services were performed in order to ensure its functionality at unitary level and looking for errors in the code logic. In some cases those tests needed the creation of an in-memory database, which is created on disk in a known state to execute the tests and destroyed after finishing the execution.

To create an in-memory database, JDBC plugin and JUnit annotations have been used to execute the setup database method before any test. A JUnit example of database creation method can be seen on the next figure.



```
@Before
public void setupDB() throws SQLException{
    c = DriverManager.getConnection("jdbc:hsqldb:mem:testdb", "SA", "");
    update("CREATE TABLE sample_table ("
            + "id INTEGER IDENTITY,"
            + "str_col VARCHAR(256),"
            + "num_col INTEGER)"
    );
    update("INSERT INTO sample_table VALUES(1,'hola',2)");
}
```

Figure 29. Example of database creation method with JUnit

To implement the JUnit test cases, as the project is configured as a gradle project, it is necessary to add the necessary dependencies to get JUnit to work. On the build gradle, it is configured to execute the JUnit tests before compiling, and it will only compile if all of the test were successful. This way, if any errors appear, regardless of their severity, the developers are required to solve the problem before deployment.

4.3.3 Blitz

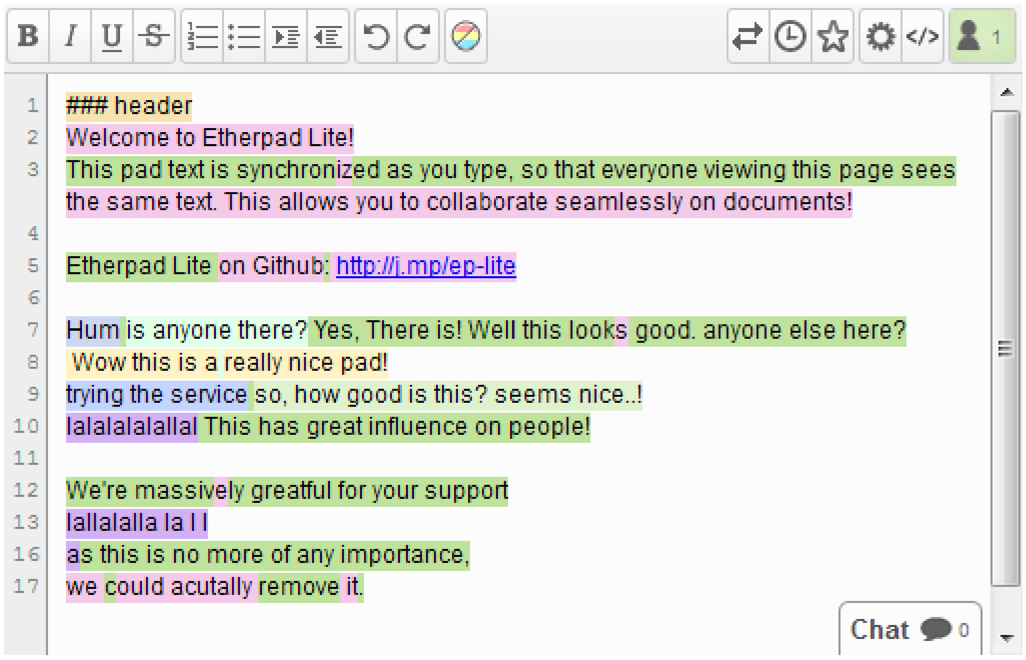
To perform the blitz testing some preparations were done. The preparation consists of a Blitz document that will describe:

1. The part of the software to be tested and the necessary equipment/software.
2. Assistants of the testing meeting.
3. Procedure of the basic scenario.
4. How to give feedback about bug or improvements found.

A Blitz template for future Blitz meeting is attached at the annex section.

While constructing this document, problems arose about how to give feedback correctly, as Jira was being used, but more than one people could find the same bug and maybe report them at the same (or at similar) time, resulting in duplicate bugs on Jira, and finally on an extra bug management task. Since this is something that should be avoided, a real-time editor has been sought to allow people involved in the blitz testing to collaborate to report the bugs they encounter.

After searching for different software, Etherpad was found, which can be installed on a private server or create a local server, allowing users to write documents in real-time, with each user's lines highlighted on different colours, as can be seen on figure 30.



```
### header
Welcome to Etherpad Lite!
This pad text is synchronized as you type, so that everyone viewing this page sees
the same text. This allows you to collaborate seamlessly on documents!

Etherpad Lite on Github: http://j.mp/ep-lite

Hum is anyone there? Yes, There is! Well this looks good. anyone else here?
 Wow this is a really nice pad!
trying the service so, how good is this? seems nice..!
lalalalalallal This has great influence on people!

We're massively greatful for your support
lallalalla la l l
as this is no more of any importance,
we could acutally remove it.
```

Figure 30. Example of Etherpad document with different users [35]

After deciding with the Product Owner and Product Manager about the part of the program that will be tested first on the blitz testing, preparations were started.

The part to be tested would be the eKauri rules, so postman will be used to make direct requests to the service. For that it has been necessary to create a series of demo users of the platform and postman requests with different pre-request scripts in order to obtain the authentication codes and environment parameters to make it the easier for the blitz testers. Also Etherpad would be installed to an internal server of the center so the data recorded will be kept on the local net of the center.

First Blitz Meeting

On the 1st of December the first blitz testing meeting was done. At first, the URL direction to the Etherpad server, the necessary postman files and the blitz template was distributed to the testers. The first problem found was that Etherpad on the server was not accessible due to a problem with installation, so at the end it was not used for this session. After that, the test continued using google docs as a real-time editor as replacement.

The session continued and the testers executed different postmans to sign in, add a rule, add actions to a rule, cause the rule to throw an alert, and also test in the web dashboard trying to find any bugs, but another problem arose during the meeting. There was a problem with the communication with the server due to a mismatch on the database version in which all tables were not updated to the correct version.

Despite all these errors on the session:

1. Some bugs were found on the application and then bug tasks were created on Jira to correct them for the next sprint.
2. Improvements and fixes for the next blitz testing were found:
    a. Solve the problem of the Etherpad on the server, installing it as a service.

   b. Create a bug template for Etherpad, so every bug report will have the same structure.
3. Improvements in methodology to know the exact database version applied on every server (Integration, Preproduction and Production).

Second Blitz Meeting

On the following blitz testing, the Etherpad software was working correctly and installed as a service, a study to solve problems with database versions was performed and was already fixed and a bug template for Etherpad to report all bugs in the same structure. To create the default bug template, the default-pad-text plugin for Etherpad has been installed, which allows the user to configure the settings file so that all the pads created with a concrete name, in this case 'blitz_', will use that template and for other names it will use a default template.

This time, three bugs were found with the blitz testing meeting and both of those bugs were major failures of the program that were prevented and avoided at the production server. So, in that test, it was demonstrated that these meetings are useful, since those bugs had not been found before.

4.3.4 Etherpad

Etherpad is an online real-time open source text editor that can be executed on an internal server, so data and written information is stored on a private protected server.

Etherpad allows the user to modify different settings to get it working as desired in each case, for example the type of database to store the data can be configured or if users need an account or just the URL to edit a pad.

By default, Etherpad highlight the text written on a defined colour, so with a simple sight it can be seen who wrote what, this can be seen above on the figure 30. This can be useful to know who found which bug and in case of problem to understand something or reproduce it, contact with the person who found it.

As Etherpad is free and open source, there are several plugins to improve the usability of it. For example, in the following figure, the main page of Etherpad is displayed with the list_pads plugin, which adds the functionality to search for the pads created by the letter it starts.

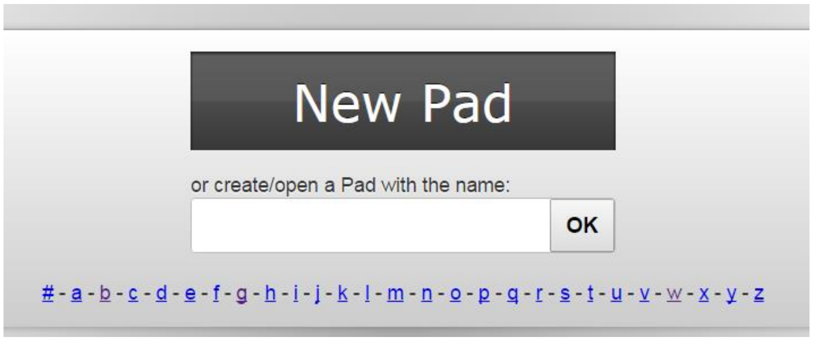

New Pad
or create/open a Pad with the name:
OK
# - a - b - c - d - e - f - g - h - i - j - k - l - m - n - o - p - q - r - s - t - u - v - w - x - y - z

Figure 31. Etherpad main screen with list_pads plugin enabled

In the Annex section is a complete guide to installing Etherpad as a service on a server with Ubuntu 14.04.5 LTS.

4.3.5 Environment versioning

A problem that arose at the beginning is to know which version are installed in which environment. The solution found to know the versioning of each environment was to apply API version services and on the web a version.json page, which provides the following useful information about the version installed:

- branch: The Git branch where is the code
- version: The version installed
- buildDate: The date of the build
- commitId: The unique Id of the commit
- commitDate: The date of the commit

In addition, on the tablet, information about the version installed inside the application has been added. The basic settings window shows the version field with the following information:

<version>-<additionalCommits>-<abbreviated_commitId> <environment> <buildDate>

For example, the latest version installed from the Integration environment on data the twentieth of January of this year, has the following version code:

v1.3.0-1-ge1434 INTE 20170120

4.3.6 Database version methodology

As seen before on the blitz testing, a problem with database versioning appeared. To solve that problem a methodology and ways to check the versions are studied.

Finally, a methodology is defined for the control of database versions, with some key elements:

- Version stored on the database itself: The database version will be stored in a dedicated table of each schema to ensure that the version installed is correct.
- Verify database version on start-up: The application communicating with database will verify the version on starting, if some version differs from expected, the application should display the corresponding error message.
- Upgrade scripts name convention: The scripts will be saved with the following name convention depending on the type of upgrade. If it is a full upgrade the naming will be:

  <schema_name>_full_<target_version>.sql

  And if it is a partial upgrade:

  <schema_name>_upgrade_<current_version>_<target_version>_<number_of_upgrade>.sql

  An example of the folders containing upgrades script would be:

```
v1.1.0 (folder)

      full_v1.1.0 (folder)

            mHB-service-rule_full_v1.1.0.sql

      upgrade_v1.0.0_v1.1.0 (folder)

            mHB-service-rule_upgrade_v1.0.0_v1.1.0_001.sql

            mHB-service-rule_upgrade_v1.0.0_v1.1.0_002.sql

            mHB-service-rule_upgrade_v1.0.0_v1.1.0_003.sql
```

The following figure shows the definition of the table for the version control of the database.

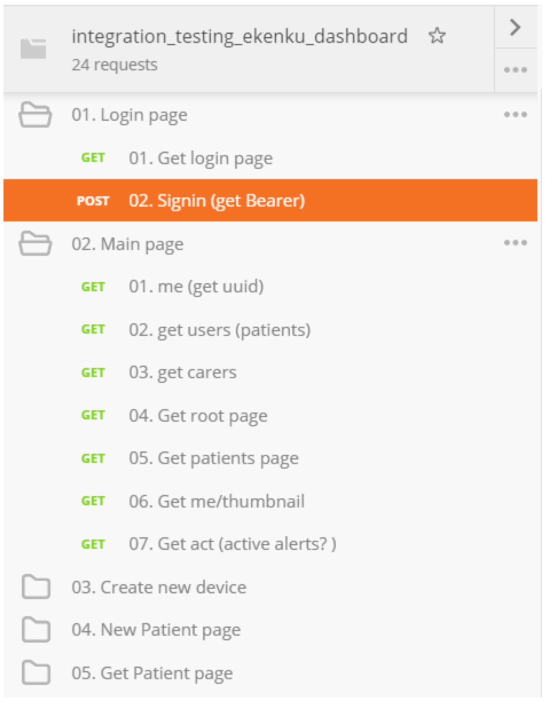
Column name	Column type	
Version	Nvarchar(50)	Not null
UpdatedBy	Nvarchar(50)	Not null
UpdatedOn	DateTime	Not null
Reason	Nvarchar(1000)	Not null

Figure 32. Table definition of database version [36]

4.3.7 Integration and Load Testing

To test the services of the eKenku project, the Postman software, as explained in the previous section, is used to perform integration and load testing on the Web platform. For that a set of postman files has been created to make requests to the server.

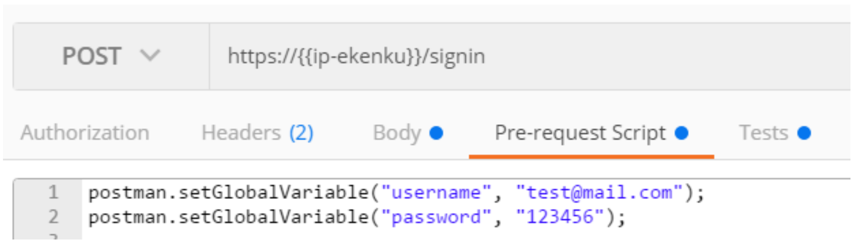
On the Postman Runner, it is possible to choose a concrete request or a complete collection to execute, but the order postman choses to execute every request is determined by the name of the request, so to prepare a correct and automated testing it is necessary to name the request in alphabetical order. To do this, the names of the requests will be preceded by a number, as can be seen on the next figure.


integration_testing_ekenku_dashboard
24 requests

- 01. Login page
  - GET 01. Get login page
  - POST 02. Signin (get Bearer)
- 02. Main page
  - GET 01. me (get uuid)
  - GET 02. get users (patients)
  - GET 03. get carers
  - GET 04. Get root page
  - GET 05. Get patients page
  - GET 06. Get me/thumbnail
  - GET 07. Get act (active alerts?)
- 03. Create new device
- 04. New Patient page
- 05. Get Patient page

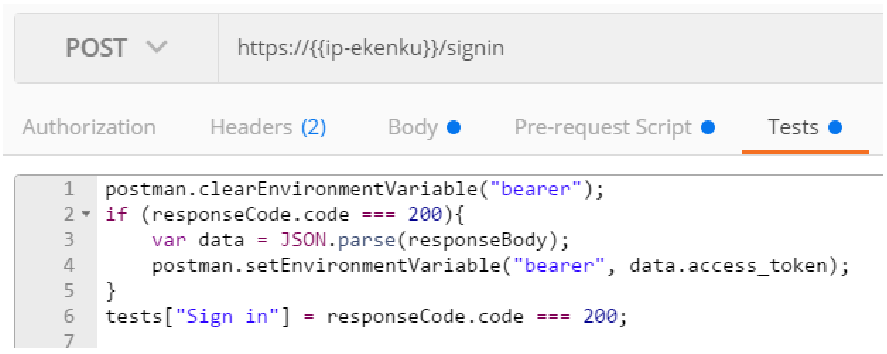
Figure 33. Postman requests ordered in folders by name.

With the help of the pre-request scripts and the tests defined in the different requests, multiple environmental and global variables were defined in order to perform correctly the following tests correctly.



```
postman.setGlobalVariable("username", "test@mail.com");
postman.setGlobalVariable("password", "123456");
```

Figure 34. Pre request script, defining global variables for login

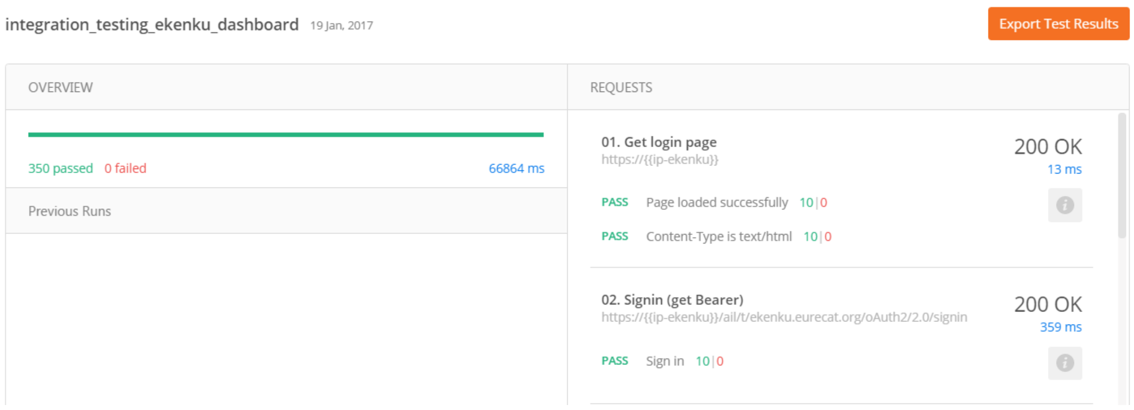


```
postman.clearEnvironmentVariable("bearer");
if (responseCode.code === 200){
    var data = JSON.parse(responseBody);
    postman.setEnvironmentVariable("bearer", data.access_token);
}
tests["Sign in"] = responseCode.code === 200;
```

Figure 35. Post request script, saving data needed for authentication

As seen on figure 34, global variables are defined, that will be changed on the body request before sending it to the server and on figure 35, the bearer environment variable is obtained from the response body, as it is needed on the following requests as authentication signature and a test is defined in order to check response code is correct, that will mean the request have been performed successfully.

After developing all postmans requests, the postman collection runner is executed on the testing environment (Preproduction), with the chosen number of executions to perform at the same time the load test. The following figure shows the results of the integration and load test to the eKenku services with 10 iterations, as an example. On the left side is a summary of the passed and failed tests, the execution time and if any, the previous runs. On the right side there is a list of the requests that have been made with information on the tests of each request.


integration_testing_ekenku_dashboard 19 Jan, 2017

Export Test Results

OVERVIEW

350 passed 0 failed 66864 ms

Previous Runs

REQUESTS

01. Get login page
https://{{ip-ekenku}}
200 OK
13 ms
PASS Page loaded successfully 10|0
PASS Content-Type is text/html 10|0

02. Signin (get Bearer)
https://{{ip-ekenku}}/ail/t/ekenku.eurecat.org/oAuth2/2.0/signin
200 OK
359 ms
PASS Sign in 10|0

Figure 36.Statistics of the load and integration testing in eKenku

As shown in the figure, the whole test was completed with a total time of 66864 milliseconds and with all tests passed. Which, as integration and load testing, means that all requests to services have been performed properly and the integration is correct and robust.

Although we do not have a concrete method to measure the benefits of applying these tests automatically to apply them manually, since each person performing the integration test may have a different rhythm, we can ensure that all requests are executed and the results are analyzed impartially.

In addition, in case of very long integration tests, the person in charge of executing it, can carry out other tasks. So we can guarantee that the application of these methods will be reflected in a reduction of time and cost.

4.4 Recommendations

While performing the application in the real projects, the errors that arose, made us realise that the Preproduction server, would have to be accessible, in the same way as the production server. Since some tests, such as sending measures without the local network of the center or with 4G, require that the network is open.

Instead of the preproduction server, in which the team can perform changes, it would be better to have an exclusive QA server with the same environment as the production server, so the tests executed would have a 100% validity.

5 Analysis of results

In this section will be described the testing environment used to develop this QA process and discussed the results obtained at different stages of the work.

5.1 Analysis of workplan

This section explains how the project was done in terms of time. For this, the actual realization times are contrasted with the estimates in the Gantt diagram. Briefly explaining the problems that have arisen and in some cases caused the time to differ considerably.

The section of methodologies of QA, took more time than in the planning, since to better understand the main types of tests, I opted to take the course of Software testing of Udacity, which has taken more time than expected.

Also, the place where the most has deferred planning, has been in the application, since at the time of testing several errors have been emerging. For example, problems with versioning of the different modules or installing Etherpad as a service on a Ubuntu server

5.2 Testing environment

In the next figure, a complete set of eKenku is shown. It consists of a tablet with the eKenku app and measurement devices, which may vary depending on the patient needs. The medical devices consist of a scale, tensiometer, pulse oximeter and glucometer. The other part of the environment is the dashboard. Also to perform the tests in the computer is verified the logs from the application and the server logs, as well as to perform the tests described in the previous sections.

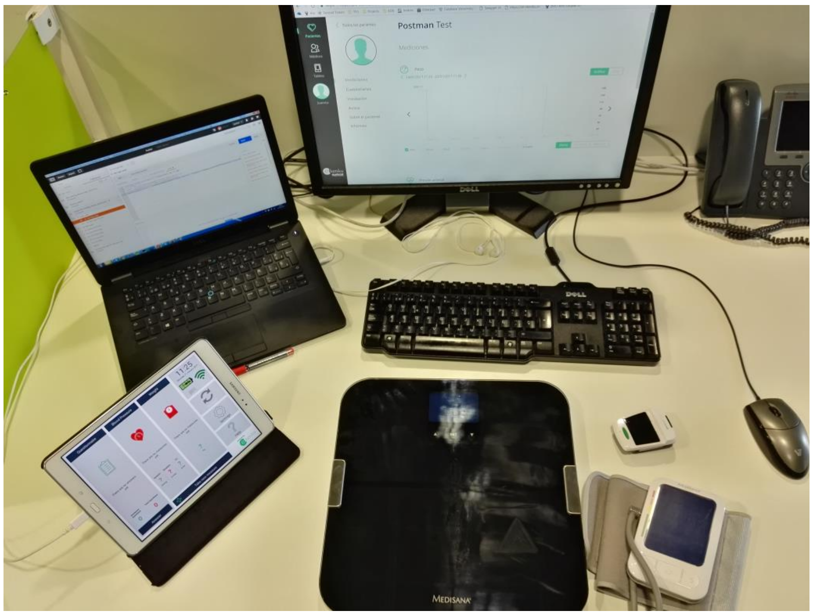


Figure 37. eKenku environment

5.3 Discussion of the results

To verify the results of the implementation of the QA process, some of the bugs that have been found using different types of tests are described.

5.3.1 Android Bugs

At first on the Android application a complete acceptance test were performed to know the state of the application. This acceptance test was executed with Zephyr for Jira, where the test cases to be tested have been defined. In the following subsections, the first testing cycle executed will be described and then the major bugs found during the implementation of the app.

5.3.1.1 First Android testing cycle

In this first testing, as said before and some of the functionalities were still under development, a complete acceptance test related to finished functionalities and about 86 tests were executed. Of these tests, three of them were not executed due to relation with market (install, uninstall, and update), four of them were blocked as errors found prevent prerequisites of the test to be accomplished.

The rest of the test was able to execute properly, but in thirteen cases the output were not the expected, so bugs were opened in Jira. Six of these bugs were of major priority and affected the correct functionality of the application. Also, on some of the passed test improvements were suggested.

5.3.1.2 Android major bugs

During the different test cycles executed on the Android application, more than 30 major bugs were found, that causes the application to work incorrectly. Some of them even made the application to crash and, as the application is designed to elderly people and patient that may not know exactly how to use an android device, that is something that cannot be allowed to happen.

5.3.2 API/Web bugs

The same way as in the Android application, an acceptance test was performed on the web platform. This test was carried out manually using the test cases defined on Zephyr.

5.3.2.1 Dashboard/API bugs

The bugs found during the dashboard acceptance tests, have been mostly bugs of visual type, but were not affecting the functionality of the platform, so these bugs were classified as minor priority. Other of the platform bugs, were defined as major bugs, because they affect the correct functionality.

Some examples of major bugs will be:

- A Security hole, in which after login with the browser developers tools, the user and password could be recovered.

- When receiving a mail from the application, the characters with accents were displayed strangely.
- Generated reports were not including the last day data.
- Changing data on the logged user profile threw an Internal Server Error.

5.3.3 Blitz testing bugs

In this part, are explained the bugs found on the blitz testing, which until now has only been done on the eKauri services. As these blitz meetings are already explained on a previous section, describing mainly the meeting itself, so here it will be described the session with focus on the bugs.

5.3.3.1 First Blitz Meeting

In this first blitz meeting, although errors arose to perform the test, bugs in the eKauri services were found. Some of the events were not performed correctly and were not stored, for example the enter room event, and on other events the room was not stored correctly and the web displayed __room__ instead of the room's name. Also, when throwing an alert about a patient, the mails were not sent to the doctor. And also a bug caused because the database was not updated to the last version.

5.3.3.2 Second Blitz Meeting

In this second meeting, three bugs were found and all of them were major bugs. The bugs were:

1. Number of visits in room for time range was not correct.
2. There was a displacement of UTC zone, so depending on the hour some alerts were thrown incorrectly.
3. Wrong visualization on mail received from alert in which the alert button was not displayed correctly on Outlook desktop program.

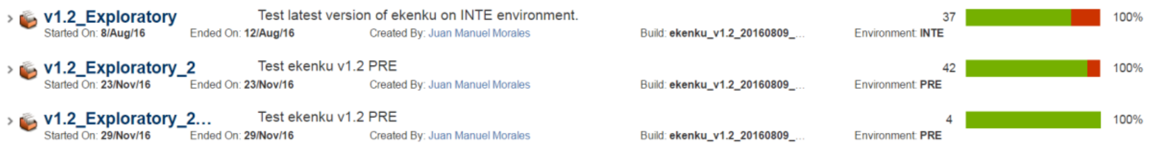
The application tested is designed to support elderly people and chronic patients, these errors cannot be allowed as the alerts may be wrong or not arrive when it is necessary and when talking about elderly people or chronic patients it may be dangerous.

5.3.4 Summary of test cycles

During the development of this project, several tests cycles have been performed with Zephyr. Zephyr also provides a test summary in which the total of tests cases defined for the project can be seen and the total of executions (one test case may be executed more than once).

Also, in this summary page is displayed the number of tests by label so, with the test cases created on this project, it is possible to know that the number of tests defined for the APP is 118 and for the backend is 160. Contrasting the total number of tests executions, which is over 800, 129 of these tests executions failed.

Despite that, searching on every version to get the test cycles would require some time, so it is possible to search tests executions. In the figure 38, an example of test cycles executed on a concrete version is shown, where it can be seen that the number of passed tests are increased on the next iteration, which only executes the tests relating to the bugs found.



v1.2_Exploratory	Test latest version of ekenku on INTE environment.				37	🟩🟥	100%
Started On: 8/Aug/16	Ended On: 12/Aug/16	Created By: Juan Manuel Morales	Build: ekenku_v1.2_20160809_...	Environment: INTE			
v1.2_Exploratory_2	Test ekenku v1.2 PRE				42	🟩🟥	100%
Started On: 23/Nov/16	Ended On: 23/Nov/16	Created By: Juan Manuel Morales	Build: ekenku_v1.2_20160809_...	Environment: PRE			
v1.2_Exploratory_2...	Test ekenku v1.2 PRE				4	🟩	100%
Started On: 29/Nov/16	Ended On: 29/Nov/16	Created By: Juan Manuel Morales	Build: ekenku_v1.2_20160809_...	Environment: PRE			

Figure 38. Iterations of test cycles of a concrete version

6 Conclusions

The conclusion of this project is divided in two sections, the conclusion itself and the remaining work to do in the future.

6.1 Conclusion

We have defined different methodologies in the two products involved in this project: eKauri and eKenku. The methodologies we used are based on the different types of tests and QA techniques that have been described in previous sections.

As for the different types of tests we have performed unit tests, integration tests, acceptance tests, functional tests and load tests. On the other hand, we have defined the bases for the blitz technique, in which the eKauri services were tested with a group of developers.

In many cases, thanks to these tests we have detected new errors, some of them major, since they crash the application and others minor, for example, translation failures.

Although we do not have a specific way to measure how much money or time has been saved in applying this procedure, we have determined that costs and total development time have been reduced, as some errors that were initially unnoticed were found thanks to the methodologies and it is easier to fix them, since they were found during development time and not in production by the final users.

Besides, finding any bugs before taking the product to the market, prevents the customer from finding the errors and decreasing his satisfaction with the product, thus making the repercussions are minimal.

Additionally, the implementation of the QA process at Eurecat has helped me to understand all the importance of a QA engineer and to learn the procedures and techniques that would help me in the future.

To conclude, the goal of this thesis to implement the QA process in Eurecat has been achieved, since most of the tests and methodologies have been applied. Although there are still some tests and techniques to perform in these projects.

6.2 Future work

As said in the previous section, there is still pendant work to do related to the QA process. The Developer on Test (DoT) technique has not been put in practice yet, so in the future it will be tried to perform it making a developer to perform different types of tests. After some iterations, a report of the usefulness of this technique in the short and long term will be done.

Also, there are tests pendant to apply in both projects. For example, the stress testing that, as said in the QA Methodologies section, is very similar to load testing, so performing this test is supposed to be easy. Using the existent Load testing prepared on postman, it is only necessary to execute the runner after turning down server or database. Maybe it can be necessary to update the postman tests to manage an error response.

Although all pendant tests would be performed, as the project evolves with new functions, changes in the environment, or maintenance, the tests will evolve to check the new changes and will be executed again. Therefore, the QA process is never finished during the lifetime of a product.

7 Bibliography

[1] pingdom. 10 historical software bugs with extreme consequences. [Consultation: 26 of January of 2017]
Available: http://royal.pingdom.com/2009/03/19/10-historical-software-bugs-with-extreme-consequences/

[2] Tutorialspoint simplyeasylearning. SDLC – Waterfall Model. [Consultation: 4 of October of 2016].
Available: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

[3] Tutorialspoint simplyeasylearning. SDLC – V-Model. [Consultation: 4 of October of 2016].
Available: https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm

[4] Tutorialspoint simplyeasylearning. SDLC – Spiral Model. [Consultation: 6 of October of 2016].
Available: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm

[5] Tutorialspoint simplyeasylearning. SDLC – Agile Model. [Consultation: 6 of October of 2016].
Available: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm

[5.1] Extreme Programming. Extreme Programming Rules. [Consultation: 7 of October of 2016].
Available: http://www.extremeprogramming.org/rules.html

[5.2] SCRUM GUIDES. Scrum guide [Consultation: 10 of October of 2016]
Available: http://www.scrumguides.org/scrum-guide.html

[5.3] AGILE FOR ALL Making Agile a Reality. Introduction to Agile. [Consultation: 10 of October of 2016]
Available: http://agileforall.com/resources/introduction-to-agile/

[6] UDACITY. Software Testing. [Consultation: 20 of October of 2016]
Available: https://www.udacity.com/course/software-testing--cs258

[7] Tutorialspoint simplyeasylearning. Software Testing – Levels [Consultation: 21 of October of 2016]
Available:
https://www.tutorialspoint.com/software_testing/software_testing_levels.htm

[8] Linkedin. Introduction to White box testing. [Consultation: 26 of January of 2017]
Available: https://www.linkedin.com/pulse/introduction-white-box-testing-part-1-aliaa-monier-ismaail

[9] Atlassian. From Quality Assurance to Quality Assistance. [Consultation: 8 of October of 2016]
Available: https://www.atlassian.com/inside-atlassian/quality-assurance-vs-quality-assistance

[10] Atlassian. JIRA Software. [Consultation: 9 of November of 2016]
Available: https://www.atlassian.com/software/jira

[11] JIRA Software. Agile Project Management. [Consultation: 20 of November of 2016]
Available: https://es.atlassian.com/software/jira/agile-project-management

[12] Atlassian. Confluence – Team Collaboration Software. [Consultation: 21 of November of 2016]
Available: https://www.atlassian.com/software/confluence

[13] Zephyr. Software Testing Tools & Test Management Software. [Consultation: 21 of November of 2016]
Available: https://www.getzephyr.com/

[14] Atlassian Answers. Problem of export with Zephyr for Jira. [Consultation: 21 of November of 2016]
Available: https://answers.atlassian.com/questions/329542/problem-of-export-with-zephyr-for-jira

[15] Atlassian Marketplace. Zephyr for JIRA - Test Management. [Consultation: 22 of November]
Available:
https://marketplace.atlassian.com/plugins/com.thed.zephyr.je/cloud/overview

[16] Bugzilla. Home :: Bugzilla. [Consultation: 23 of November of 2016]
Available: https://www.bugzilla.org/

[17] Wikimedia. BugzillaScreenshot. [Consultation: 23 of November of 2016]
Available:
https://upload.wikimedia.org/wikipedia/commons/thumb/0/0d/BugzillaScreenshot.png/350px-BugzillaScreenshot.png

[18] Git. Git. [Consultation: 8 of November of 2016]
Available: https://git-scm.com/

[19] Git. About. [Consultation: 8 of November of 2016]
Available: https://git-scm.com/about/staging-area

[20] SUBVERSION. Apache Subversion. [Consultation: 8 of November of 2016]
Available: https://subversion.apache.org/

[21] JUnit. JUnit – About. [Consultation: 24 of October of 2016]
Available: http://junit.org/junit4/

[22] Nunit. NUnit – Home. [Consultation: 26 of October of 2016]
Available: https://www.nunit.org/

[23] Pytest. Full pytest documentation [Consultation: 26 of October of 2016]
Available: http://docs.pytest.org/en/latest/

[24] dbUnit. DbUnit – About DbUnit. [Consultation: 28 of October of 2016]

Available: http://dbunit.sourceforge.net/

[25] HttpUnit. HttpUnit Home. [Consultation: 30 of October of 2016]
Available: http://httpunit.sourceforge.net/

[26] SoapUI. SoapUI | Functional Testing for SOAP and REST APIs. [Consultation: 1 of November of 2016]
Available: https://www.soapui.org/

[27] SeleniumHQ. Selenium – Web Browser Automation. [Consultation: 1 of November of 2016]
Available: http://www.seleniumhq.org/

[28] Watir powered by selenium. Watir is… - Watir Project. [Consultation: 3 of November of 2016]
Available: http://watir.github.io/

[29] WatiN. WatiN. [Consultation: 4 of November of 2016]
Available: http://watin.org/

[30] FitNesse. FrontPage. [Consultation: 7 of November of 2016]
Available: http://fitnesse.org/

[31] FitNesse.UserGuide.TwoMinuteExample. [Consultation: 7 of November of 2016]
Available: http://fitnesse.org/FitNesse.UserGuide.TwoMinuteExample

[32] THE APACHE SOFTWARE FOUNDATION. Apache JMeter. [Consultation: 10 of November of 2016]
Available: http://jmeter.apache.org/

[33] Postman. Postman | Supercharge your API workflow. [Consultation: 14 of November of 2016]
Available: https://www.getpostman.com/

[34] GitHub. The httperf HTTP load generator. [Consultation: 17 of November of 2016]
Available: https://github.com/httperf/httperf

[35] Etherpad. Etherpad. [Consultation: 28 of November of 2016]
Available: http://etherpad.org/

[36] InfoQ. Database Versioning and Delivery with Upgrade Scripts. [Consultation: 9 of January of 2017]
Available: https://www.infoq.com/articles/db-versioning-scripts

8 Annex

8.1 QA report

Weekly progress report

This is the actual state of eKenku APP on 2016/05/12

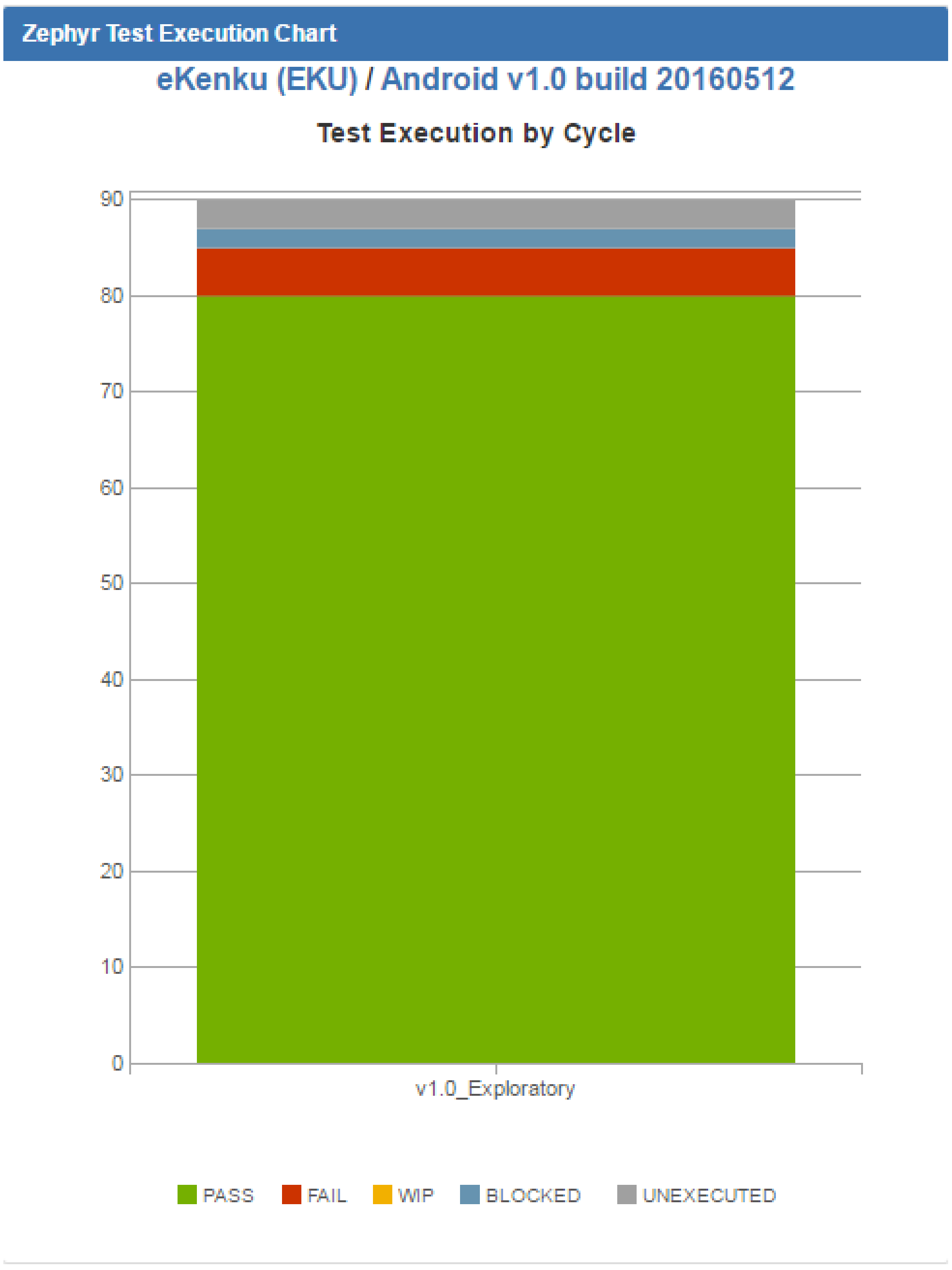
Cycle: v1.0_Exploratory

Version: Android v1.0

Build: ekenku_v1.0_20160512_1_PRE

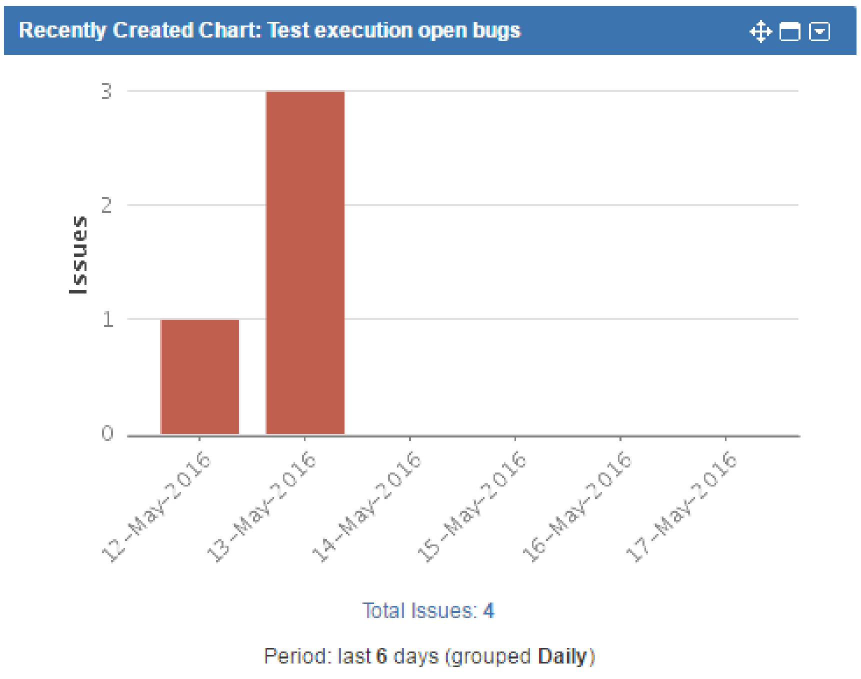
Environment: PRE

Tests cycle status



Here we see that mostly of the tests in this cycle were successfully passed (88.89%). A few tests in this cycle (3.33%) remain unexecuted due to this tests are related to market (install, uninstall, update). Blocked tests (2.22%) are related to the lack of glucometer during the test execution. Finally we have 5.56% of tests Failed.

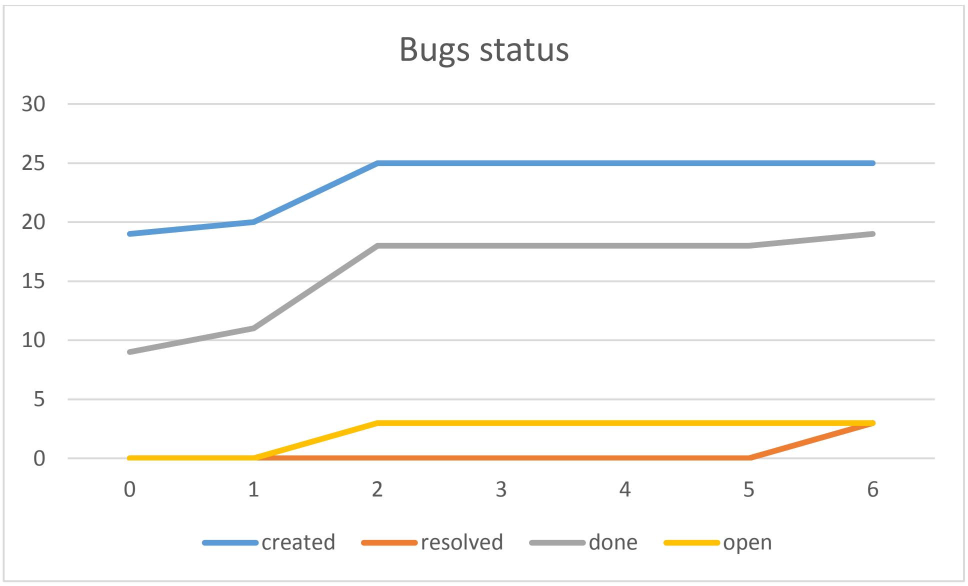
Recently created bugs chart



Here we can see the number of created bugs still not closed every day since the last test cycle.

During this cycle, we have checked that old bugs has been successfully resolved or if the error keeps on. On the previous chart we can't see 2 issues that hasn't been created during this cycle, but that were supposed to have been resolved and after check it, it wasn't resolved.

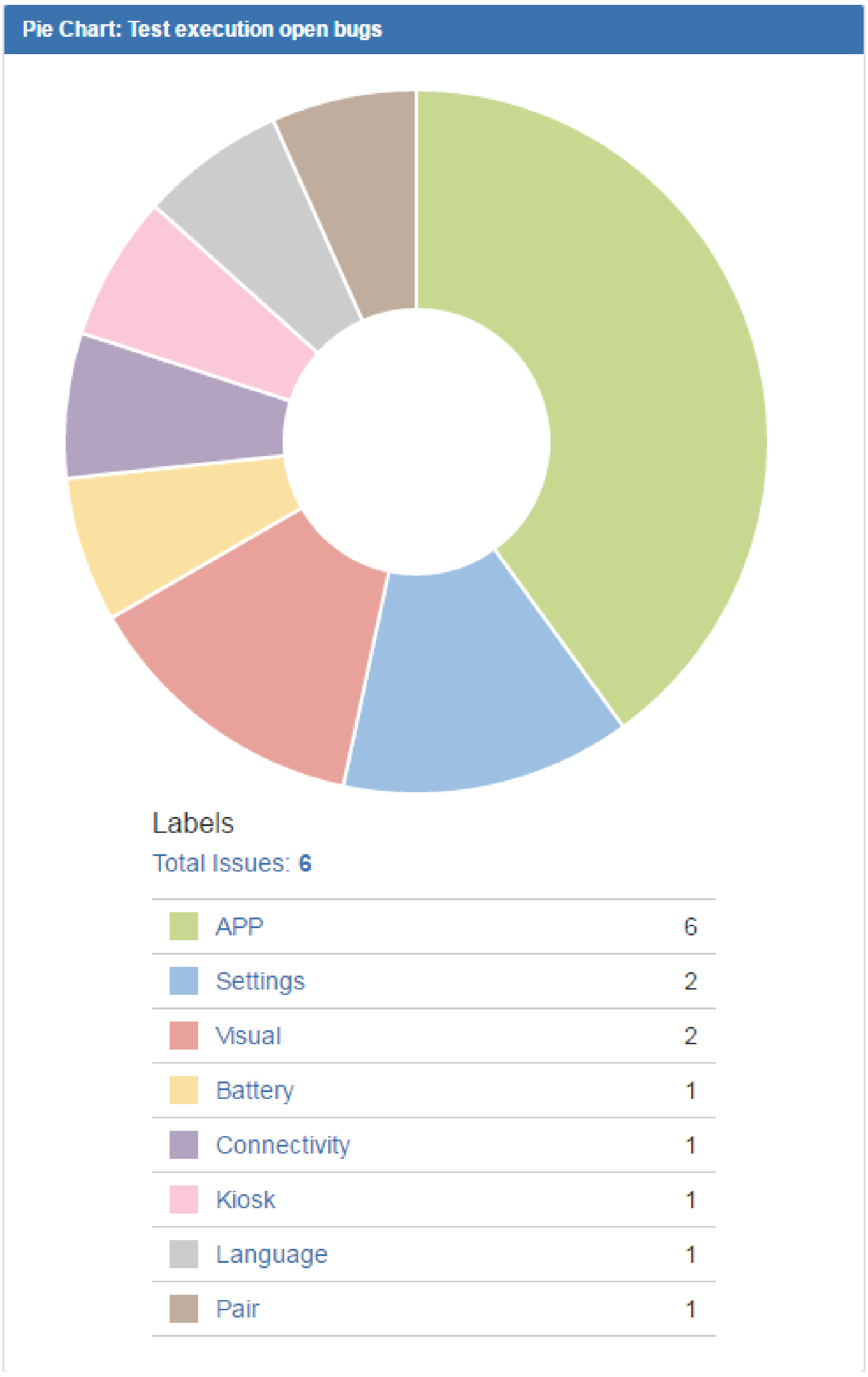
Bugs status



In this graph, we see the progression of created, resolved, done and open bugs by day.

The first day in the previous graph, we took the total created and closed issues until that day. Resolved bugs makes reference to bugs found in the current and previous versions of the APP and Open bugs references to the found bugs in this test cycle that are still not resolved.

Nº of bugs by type



Labels

Total Issues: **6**

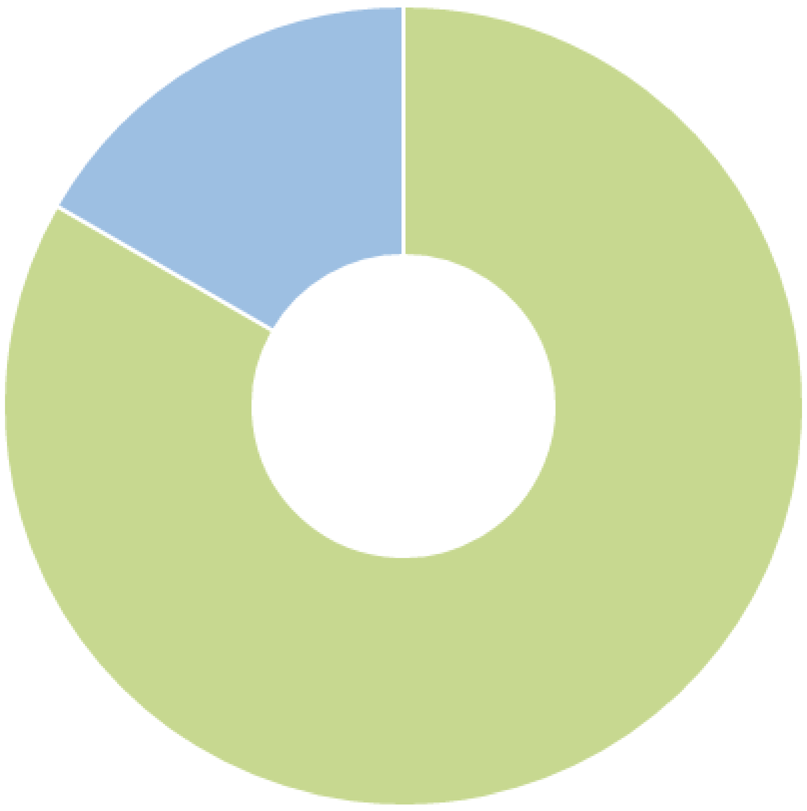
🟩	APP	6
🟦	Settings	2
🟥	Visual	2
🟨	Battery	1
🟪	Connectivity	1
🟥	Kiosk	1
⬜	Language	1
🟫	Pair	1

In this pie chart we can see the number and types of the current bugs found.

As we can see in the pie chart, we have opened a total of 5 issues, although in the legend may seem that we have more issues. This is because this chart is classified by labels to recognize which components are affected by the issues. Some issues were referred to more than one component and all issues refers to the APP.

In the following graph, we can see the priority of the found bugs:

Pie Chart: Test execution open bugs



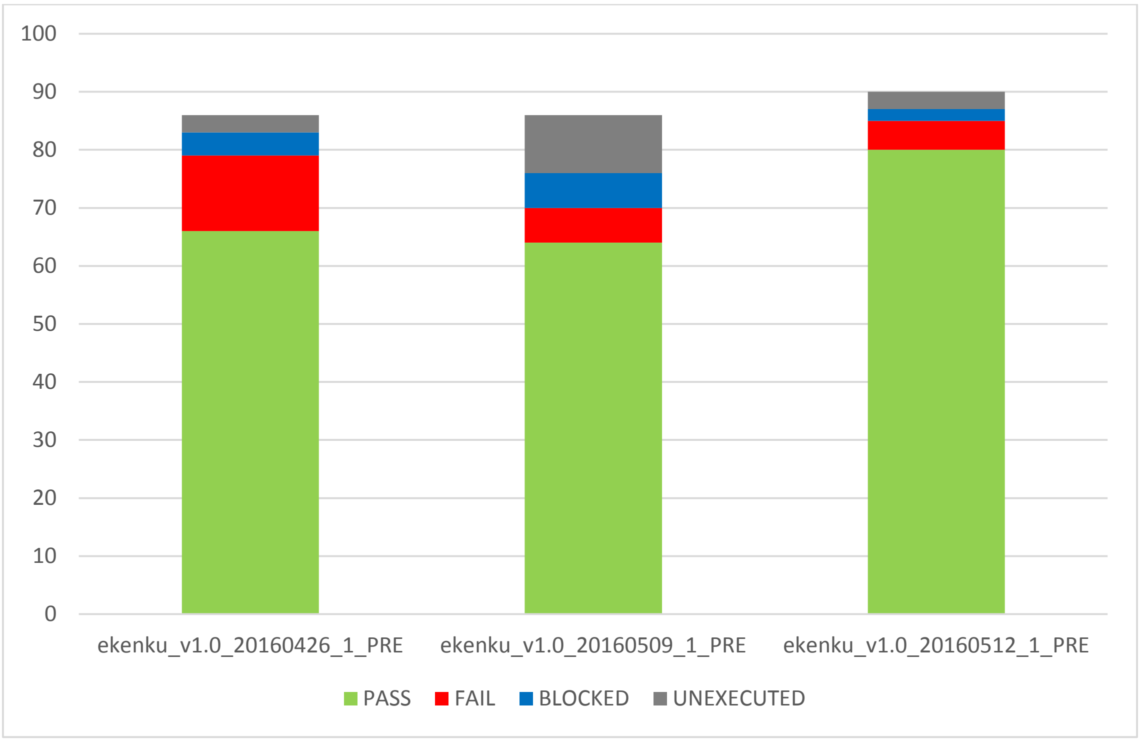
Priority

Total Issues: **6**

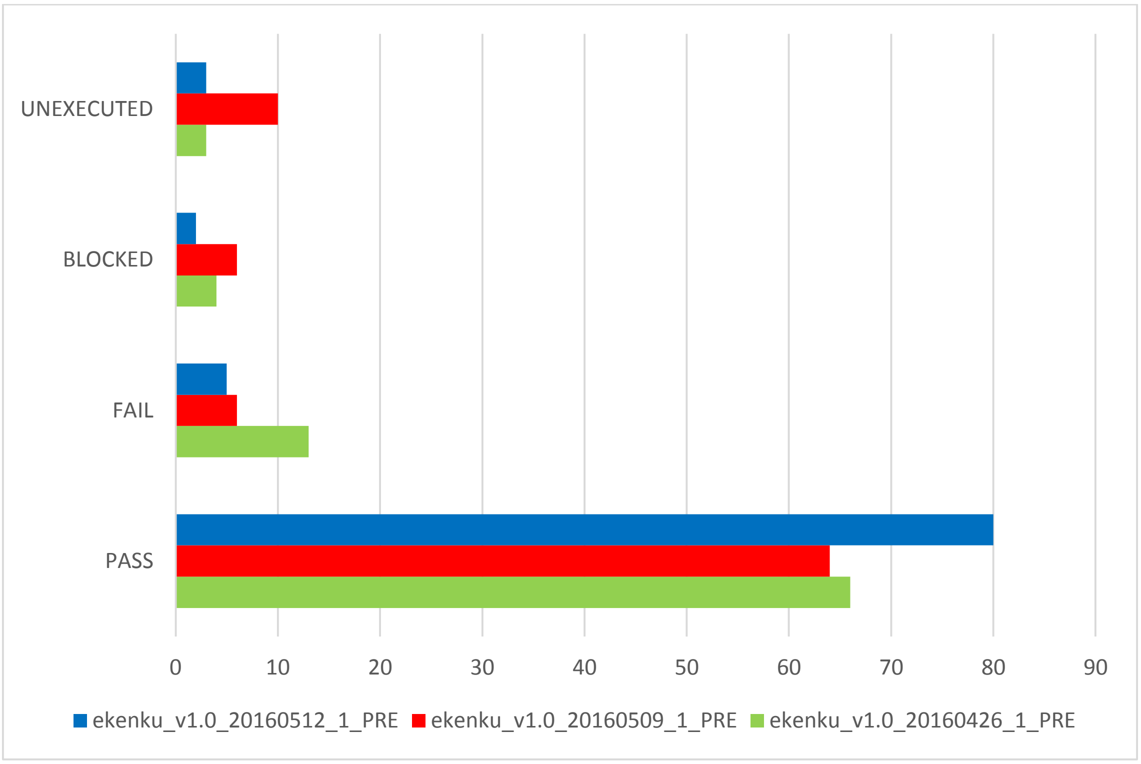
🟩	Minor	5
🟦	Major	1

As we see, only 1 of the found bugs is a Major priority, which means it caused some malfunction or serious problem during execution

Tests cycle progression



Here we see the progression status of the APP by test cycles. If we look the second version of the APP, there were more test unexecuted. This happened because there were changes in the backend that invalidates some tests.



This graph is structured by status of realized tests.

If we take a look in the previous graph the latest version of the APP (blue color), we can see that the number of passed test has increased meanwhile the blocked and failed tests has decreased regarding previous version.

So after taking a glance at the document, as almost any major bug has been found, after resolution of these bugs I would consider the APP ready to market.

8.2 Git workflow

APENDIX: EURECAT'S (GIT) REPOSITORY STRUCTURE MODEL

This model relies on three main branches:

- A **master** branch containing production-ready code.
- A **develop** branch that reflects the latest delivered development changes for the next release. Pull requests should be issued against this branch.

In addition to these two branches with an infinite lifetime, three types of supporting branches may also be used occasionally:

- **Feature or personal experimentation** branches with active code development. Usually related to one ticket/issue.
- **Hotfix** branches with emergency fixes to the production site/code
- **Release** branches that hold code ready for production on a staging environment.

Each of these branches have a specific purpose and are bound to strict rules as to which branches may be their originating branch and which branches must be their merge targets.

The master branch (Git's default branch)

The **master** branch should always reflect a production-ready state. In other words, this branch is used to "store" production releases so an end-user cloning this branch will have:

1. A stable code
2. The "raw and unpackaged" production version of the last table release.

The tag naming convention differs when it an android repository from other repositories. Common tag naming convention:

v<major>.<minor>.<patch> (i.e. v1.2.1)

Android tag naming convention:

<major>.<minor>.<patch> (i.e. 1.2.0)

The develop branch

The **develop** branch is the main development. This branch ***should not contain highly experimental code*** but should rather be considered as an "integration branch": when code is stable enough to be added to the next release, it may be added to this branch. Work happens on the develop branch and small changes are committed directly on this branch. Features branches will be merged in here. Develop is always ready for new functionality.

The supporting branches

As opposed to the **main** branches, the supporting branches are used (more or less) temporarily. Their goal is to aid parallel development between team members by:

- Providing feature freeze staging branches. (**release** branches)
- Easing tracking of features (**feature** branches)
- Assisting in quickly fixing live production problems. (**hotfix** branches)

The **release** branches are used to prepare a new production release as well as to keep a record of the different releases that took us to a certain production version. They allow last-minute fixes and verifications to make sure everything is OK before a “public” release.

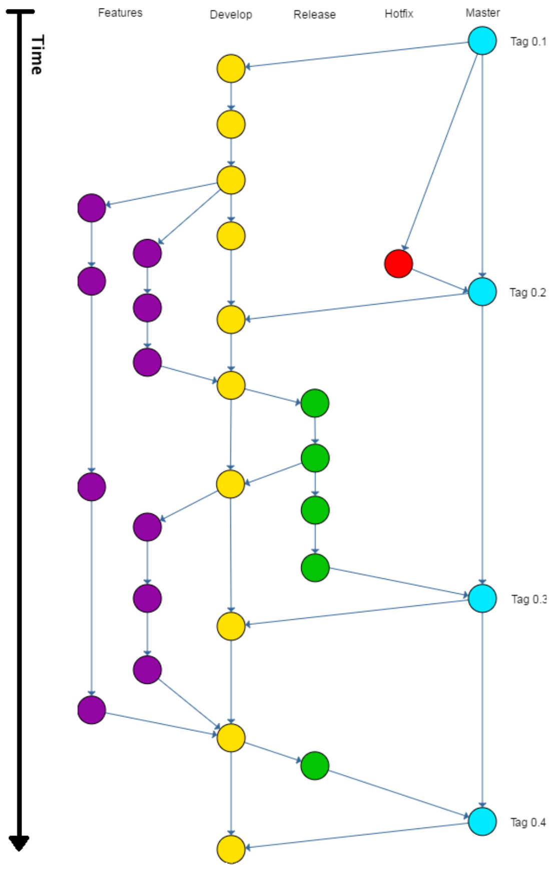
- It must branch from **develop** and must be merged into **master**, and then **master** merged into **develop**.
- Naming convention: **rc-<major>.<minor>.<patch>** (i.e. rc-1.2.0)

The **feature** branches are used to develop new features or fixes for the upcoming or a distant future release. A typical **feature** branch would exist as long as the feature, fix or experiment is in development and should be deleted when the code is merged into the **master** branch (or if the feature is discarded).

- Must branch from **develop** and must be merged to **develop** again.
- Naming convention: **feature-<JiraIssueNum>** (i.e. feature-EKU1002)

The **hotfix** branches are typically used when a critical bug is found in a production release. Everything related to the resolution of the bug should be handled in a **hotfix** branch and then propagated to **master**, and then from **master** to **develop**.

- Must branch from **master** and must be merged to **master**, and then **master** merged into **develop**.
- Naming convention: **hotfix_<date>** (i.e. hotfix_20170109)



Eurecat’s (Git) Branching Strategy

8.3 Blitz template example

Blitz testing

Project: ekauri

Environment: PRE

Platform/s: Rules

Required equipment: Laptops (everyone), internet connection, postman file

Required software: Postman

Preconditions: everyone will have two accounts: patient testing account and admin account

Data needed: URL, Accounts (user: Eurecat mails / password: 123456)

Procedure: First of all, configure postman variables for your user: On Sign In (get Bearer) postman, on Pre-request Script:

- Update line 2 with your Eurecat e-mail.
- If you have a different password, then update line 3.
- Update lines 18 and 19 with your correct patient/recipient.

Use the given postman's to check the correct functionality of the rules of ekauri in PRE environment. As this is a free test to find bugs/improvements feel free to change postman data to test all possible cases (i.e. change between positive/negative numbers, letters, inexistent patients or sensors).

Feedback: Etherpad will be used for feedback as is free and distributed, so everyone can report a bug/improvement and see others reports in real time (to avoid duplicate bugs). After session found bugs/improvements will be opened in Jira.

8.4 Etherpad installation guide

APENDIX: ETHERPAD INSTALLATION GUIDE

Installation of Etherpad on Ubuntu 14.04.5 LTS as a service

This tutorial helps you to deploy Etherpad on Ubuntu 14.04.5 LTS. Before starting this guide, ensure you have administrative rights. Additionally, you'll need node.js installed, ideally the latest stable version, it is recommended installing/compiling nodejs from source (avoiding apt).

1. Download the last version of Etherpad from http://etherpad.org/#download and extract it to a folder.
2. Configure the file settings.json if needed. This file it is prepared by default to be installed on a server using the server ip, with 9001 port and dirtyDB type of database.
3. After that, upload your Etherpad folder to the server. From the server, give writing permission to the Etherpad folder and define the user and group ownership of the Etherpad files.

```
chmod +x /path of Etherpad installation/
chown user:group -r /path of Etherpad installation/
```

4. Create an empty folder for Etherpad logs and give it writing permissions with the same commands as before, but changing the path to the log folder.
5. To create the service script, execute the following command:

```
sudo nano /etc/init.d/etherpad
```

And paste the following code and change the parameters between <> from the highlighted lines to the ones you chose on previous steps:

```
#!/bin/sh

DIR="<path of etherpad installation>"
PID=/var/run/etherpad.pid
LOG=<path of etherpad folder log>/etherpad.log
DESC=<description of Etherpad instance>
USER=<user>

set -e

. /lib/lsb/init-functions

case "$1" in
        start)
                echo "Starting $DESC..."
                exec nohup sudo -u $USER /opt/etherpad/bin/run.sh > $LOG 2>$1 < /dev/null &

                # Wait five seconds for the node.js server to start up.
                sleep 5

                # Store the PID in a file.
                ps ax | grep [s]erver.js | awk '{print $1}' > $PID
                echo "Done"
                ;;
        stop)
                # Terminate Etherpad.
                echo "Stopping $DESC... "
                kill `cat $PID`
                rm -f $PID
                echo "done"
                ;;
        restart)
                # Terminate Etherpad.
                echo "Stopping $DESC... "
                kill `cat $PID`
                rm -f $PID
                echo "Done"

                echo "Starting $DESC..."
                exec nohup sudo -u $USER /opt/etherpad/bin/run.sh > $LOG 2>$1 < /dev/null &

                # Wait five seconds for the node.js server to start up.
                sleep 5

                # Store the PID in a file.
                ps ax | grep [s]erver.js | awk '{print $1}' > $PID
                echo "Done"
                ;;
        status)
                ps ax | grep [n]ode
                ;;
        *)
                echo "USAGE: $0 {start|stop|restart|status}"
                exit 0
        esac
```

6. Now, you can start/stop/restart Etherpad with the following commands:

```
sudo service etherpad start
sudo service etherpad stop
sudo service etherpad restart
```

An example of the installation is described next:

1. Download the last version of Etherpad from http://etherpad.org/#download and extract it to a folder.
2. Upload the Etherpad to the /opt/ folder and give writing permissions to the Etherpad folder and define the user and group ownership of the Etherpad files (we used ekauri user and root group).

```
chmod +x /opt/etherpad/
chown ekauri:root -r /opt/etherpad/
```

3. Create an empty folder for Etherpad logs on /var/log/etherpad.

```
chmod +x /var/log/etherpad/
chown ekauri:root -r /var/log/etherpad/
```

4. create the service script, executing the following command:

```
sudo nano /etc/init.d/etherpad
```

The code for the example installation is the following:

```
#!/bin/sh

DIR="/opt/etherpad"
PID=/var/run/etherpad.pid
LOG=/var/log/etherpad/etherpad.log
DESC=Etherpad
USER=ekauri


set -e

. /lib/lsb/init-functions

case "$1" in
        start)
                echo "Starting $DESC..."
                exec nohup sudo -u $USER /opt/etherpad/bin/run.sh > $LOG 2>$1 < /dev/null &

                # Wait five seconds for the node.js server to start up.
                sleep 5

                # Store the PID in a file.
                ps ax | grep [s]erver.js | awk '{print $1}' > $PID
                echo "Done"
                ;;
        stop)
                # Terminate Etherpad.
                echo "Stopping $DESC... "
                kill `cat $PID`
                rm -f $PID
                echo "done"
                ;;
        restart)
                # Terminate Etherpad.
                echo "Stopping $DESC... "
                kill `cat $PID`
                rm -f $PID
                echo "Done"

                echo "Starting $DESC..."
                exec nohup sudo -u $USER /opt/etherpad/bin/run.sh > $LOG 2>$1 < /dev/null &

                # Wait five seconds for the node.js server to start up.
                sleep 5

                # Store the PID in a file.
                ps ax | grep [s]erver.js | awk '{print $1}' > $PID
                echo "Done"
                ;;
        status)
                ps ax | grep [n]ode
                ;;
        *)
                echo "USAGE: $0 {start|stop|restart|status}"
                exit 0
        esac
```

5. Now, you can start/stop/restart Etherpad with the following commands:

```
sudo service etherpad start

sudo service etherpad stop

sudo service etherpad restart
```