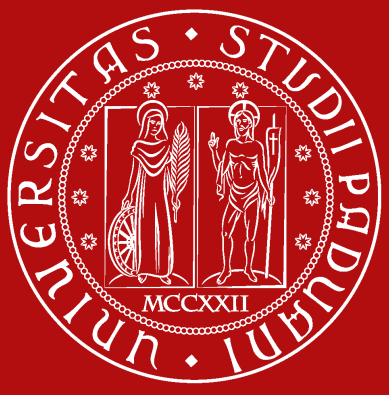

DIPARTIMENTO
MATEMATICA

Supporting tools for Agile software development:

Experience from a real use case

Bachelor's Degree thesis in Computer Science

Candidate: Ciprian Stefan Voinea
Student ID: 1143057

Supervisor: Dott. Armir Bujari

Accademic Year 2018 - 2019

Università degli Studi di Padova

Department of Mathematics "Tullio Levi Civita"

Bachelor Thesis in Computer Science

Supporting tools for Agile software development: experience from a real use case

Supervisor
Dott. Armir Bujari

Candidate
Ciprian Stefan Voinea
Student ID
1143057

Accademic Year 2018 - 2019

To my parents,
for the sacrifices they made for me

Abstract

This document describes the work done during the internship period by Athonet S.R.L. between the months of June and July of 2019.
Athonet presented its needs to have a suite of software that would allow them to have a more disciplined way of working.
The objective was to create a suitable environment using the tools Jira and Confluence, developed by Atlassian, ready to be transferred into production.
These software have been developed to work alongside: an Issue Tracking System to always keep track of the project's progress and a sharing platform that allows a better control over project related documents. With their features a company could also create a portal to communicate with clients and have documentation of their products public, in order for the customers to have access it anytime.
By introducing Jira and Confluence in its way of working, a company can adopt Agile methodologies or similar ones which would allow not only a better project management but a better handling of the resources as well.

Contents

Listing of figures

1 Introduction

One of the most important factors for the success of an Information Technology (IT) company is its "*way of working*": a set of rules that make systematic, disciplined and quantifiable the activities of the project. In time these could become *best practices*, i.e. a well known to work way of working that is reliable and has demonstrated to have succeeded.
To correctly apply these rules and follow the work of its employees, the company has to use software instruments, which, in the context of this thesis, are Jira and Confluence.

1.1 Premise

This document is a report of the two month curricular internship done between June and July 2019 at Athonet under the supervision of Dott. Fabio Giust. It contains a description of the work that done and an introduction to Agile and Scrum software developing methodologies.
To introduce and describe some of the arguments there will be a few comic strips of "*Dilbert*", a character invented by Scott Adams[1]. It satirically represents the problems that can be present in any small or big IT company.

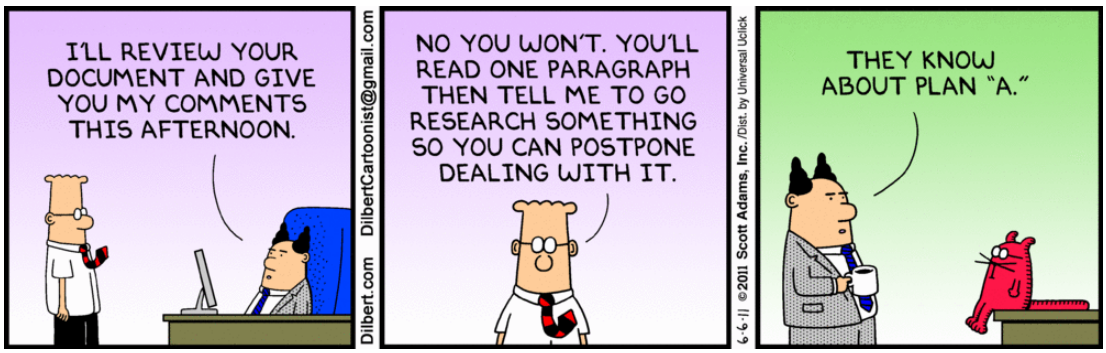

Figure 1.1: Dilbert, "*Plan A*"

The "*Glossary*" section at the end of the document describes technical words that would need a more specific introduction: these are marked as EXAMPLE$_G$.

1.2 THE COMPANY

Athonet[2] is a telecommunication company headquartered nearby Vicenza, Italy. It stem from the idea that BROADBAND NETWORKING$_G$ should be easily available to people in rural areas and to companies that work in mission critical services or special environments like shipping, mining companies or even hospitals (safety critical).
Officially it was founded in 2004, although a working prototype of their idea was already developed by the CEO (Chief Executive Officer) and CTO (Chief Technology Officer) that were working alongside in Ericsson[3] at that time.

Figure 1.2: Athonet's logo

LOW LATENCY$_G$ communication, reliability and security are at the core of what Athonet provides. Their main product is "*PriMo*", a device that allows to create a dedicated CORE NETWORK$_G$, or enterprise LTE$_G$.

Figure 1.3: Athonet's main product **"*PriMo*"**[4]

The philosophy behind PriMo is that "*the software is the network*", as Athonet was a pioneer in decoupling the software from the hardware, in order to have the solution running on different general purpose computing hardware, e.g., based on x86 processors.
In 2012, after a disastrous earthquake destroyed all the communication lines in Emilia Romagna, Athonet has reestablished connectivity, thus showing the how PriMo can be used on the field in emergency situations.
Athonet installed PriMo at the top of a school, allowing them to cover the affected area not only for the operators of Servizio Civile but for the citizens as well.
In December 2013, Athonet has been rewarded by Giorgio Napolitano, the then President of the Italian Republic, with a medal for merits for the sustainability in the digital sector[5].
Lately they have migrated some of their functions to the cloud: by using AWS they have achieved a hybrid product, BubbleCloud, a plug and play solution that allows to locally deploy the physical Edge Nodes while managing them from the AWS cloud.
This small business has much to offer, considering that some of its competitors are giants like Nokia and Ericsson.
As a proof of the continuous will to improve and for the solution it provides, Athonet has been awarded with four Global Mobile Awards at the GSMA Mobile World Congress, held in February 2019 in Barcelona[6].

1.3 The project

It consists in installing and configuring two main software tools, Jira and Confluence, alongside plugins to add more functions and enhance their potentiality. However, the most important part of this project is not installing the software, but adapting it to the needs of the company.
As said earlier Athonet is a growing business, and because of this it needs to give itself some internal rules and specifications to follow when working on a task, communicating with the client or even share internal information to employees. Not only for themselves but for clients they work with as well, since most big companies require that their partners have internal regulations, no matter the number of people in the company.
As will be explained in the following chapters, Jira is an Issue Tracking System (ITS), a software that allows to follow and record tasks (like resolving bugs or implementing features), called issues, that are related to a project, while Confluence is a software for sharing knowledge, that means exchanging internal documents, keeping a wiki, having documentation available and even for customers, etc.
The assignment was to demonstrate that these tools are what Athonet needs to be a stronger an coherent company, in which information is always shared and available, while maintaining a history of changes, by creating an environment that suited their needs and that can evolve alongside the company.
To do this I have learned the basics of Agile and Scrum methodologies, how a company operates internally and with their clients and how introducing a new tool may improve the way of working, even if it creates chaos at the beginning.

1.4 Document organization

The main part of this thesis is organized as follows:

- Chapter 1 or "*Introduction*": describes the overall content of this document
- Chapter 2 or "*The internship project*": describes in detail the objectives and planning of the internship project
- Chapter 3 Chapter 3 or "*Agile processes and methodologies*": an introduction to the Agile software development
- Chapter 4 or "*Jira and Confluence: the essentials*": describes the most valuable functionalities of Jira and Confluence

- Chapter 5 or "*Project implementation*": details how the project has been implemented by dividing it into time periods
- Chapter 6 or "*Conclusions*": contains the retrospective of the project, future developments and personal considerations

A goal without a plan is just a wish.

Antoine de Saint-Exupéry

2 The project

This chapter contains the results of the discussions and planning that have been made prior to the beginning of the internship. These contents are described in more detail in the "*Piano di Lavoro*" (work plan in English), a document that contains an estimation of resources for each task that composes the project. Chapter A contains two extracts from that document: a table with the hours spent per task and a Gantt chart. For both it's present the original estimation and a final version.
At the beginning of May there was a first meeting with the tutor to understand the needs of the company and draft a plan. Below follows a description about the project and how it was approached.

2.1 The company's needs

At the time of writing it has nearly 40 employees and it is expected to hire new people soon. Athonet uses multiple software tools for keeping track of projects, sharing information internally and with the clients, visualizing PRODUCT ROADMAPS$_G$, etc.
The most important tools they are currently relying on are:

- "*Redmine*"[7] as an Issue Tracking System
- "*SharePoint*"[8] as a collaborative platform
- "*Planner*"[9] for visualizing project roadmaps

- "*GitLab*"[IO] as a REPOSITORY$_G$

While some of these software packages are compatible, others were not developed to be used together, although there are plugins that allow to interconnect them. These don't allow much customization since they are programmed by third party developers and not always the compatibility is up to date with the latest releases.
Consequently there is a need to provide employees with a suite of stable tools that are easily interconnected and with a vast and well done documentation. Also, these tools must be easy to maintain and update, since they don't have to become obsolete and outdated.
Nobody likes legacy systems.

Figure 2.1: Dilbert on legacy system infrastructures

Internal changes may not be directly visible to the clients, but the effect of having a more organized company, where there is always track of the work done and in progress, ensures that when there is a request from the client it does not go unseen or unanswered.
Although, forcing employees to use a software instead of another one is not easy without creating chaos. This is why a good and available documentation is the key on helping employees transition from a tool to another.

2.2 Requirements and objectives

The final objective was to create a suitable and working environment that would be ready to transfer it into production and explain the users how to use it.
Requirements can be divided in three categories based on their relevance:

- Mandatory

- Desirable
- Optional

A detailed list of requirements that describe more accurately the ones included in the original planning document can be found in B.

2.3 Time division and planning

The time division of the project has been visualized using a Gantt chart, alongside a table that contains a realistic approximation of the hours spent per task.
The internship was planned for a duration of ten weeks and the project has been spread such that there would be enough time to understand the tools and use them to get acquainted and demo them to the other employees. This time period also includes the phases for getting feedback from users, fine tune the configuration, explain the tools to various members of the company and, in case of incidents, slack time. The complete Gantt chart can be found in A.3.
This temporal sequence has been draft only after understanding the requirements and getting an idea about how Atlassian's software works, also I wanted to perceive the final objective and think on how to achieve it. As can be seen in the Gantt chart, there are some tasks that overlap in time: for example the "*Study of the Atlassian suite*" overlaps with Configuration of the environment tools. This because the tasks may have something in common or one helps the achievement of the other. While studying the requirements for the Atlassian software I decided I would also configure the host machine.
I have coarse grained planned four main time periods: "*Learning*", "*Implementing*", "*Testing and fine tuning*", "*Feedback*". Each of these contain smaller tasks that involve studying the tools, installing and configuring them, etc.
As discussed with the tutor, meetings had to be considered so that the progress done could be explained to him and the other company figures that would eventually use the software.
As milestones$_G$, in agreement with the tutor, the deliverable that marks the progress of the work was a VM that represented the status of the system. By definition this must be versionable and with a well written changelog.

If a team couldn't be fed with two pizzas, it was too big.

Jeff Bezos

3

Agile Software Development

Before getting into the implementation and adaptation of Jira and Confluence, let's take a step back and understand the concept of Agile, a "*Software Development Life Cycle*" (or SDLC) model.

SDLCs did not emerge until the 1960s; they are considered to be the oldest formalization of FRAMEWORK$_G$. A SDLC refers to the ensemble of activities that compose a software project. It starts with the concepts of understanding the problem as well as the requirements and it ends with the retirement of the system (when there is no more maintenance) or with the cancellation of the project.

Small projects (generally for a single person) have a simpler life cycle: find the problem and write a program to solve it; once the problem has been solved, the program can be deleted and forgotten.

In larger projects, that require a team to be developed, there must be some explicit rules to set a higher quality for the software.

As activities are assigned to different people, it becomes critical that all participants share a common view of the execution of the project. A SDLC model is a framework providing the ordering and dependencies of life cycle activities, managing these can be a major impact for a successful project and its duration.

For example, a change in requirements during implementation may invalidate a substantial amount of work and delay the delivery of the system by several months. Different life cycle models prescribe different actions to handle such changes[11].

There are many SDLCs like "*Waterfall*", "*Prototyping*", "*Iterative*" and "*Incremental Development*", "*Spiral Development*", "*Rapid Application Development*", and "*Extreme Programming*" (XP).

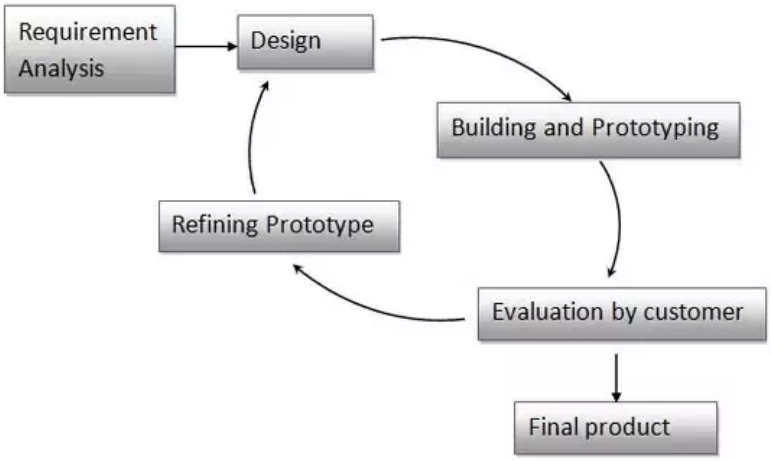

Figure 3.1: The "*Prototype*" model

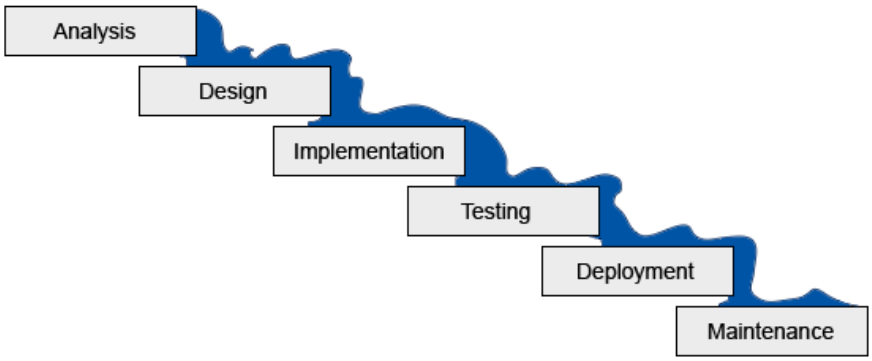

Figure 3.2: The "*Waterfall*" model

As the word "*model*" suggests, each company can apply its own SDLC designed ad hoc for their internal use.
This led to the creation of a standard that presents the guidelines for elements that are most frequently used in describing a process: title, purpose, outcomes, activities, task and information item. The "*ISO/IEC TR 24774:2010: Systems and software engineering – Life cycle management – Guidelines for process description*"[12].
The complexity and slowness in producing a concrete product in older SDLCs brought the need for a faster and more communicative model.

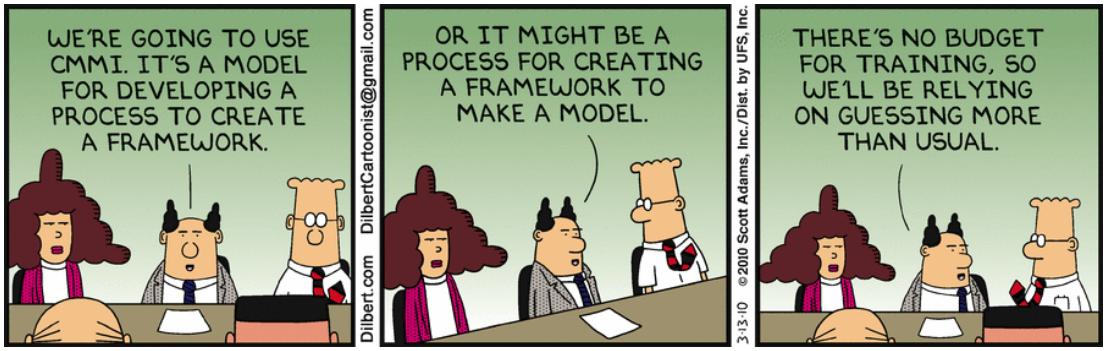

Figure 3.3: Dilbert on creating a framework

The Agile SDLC model is a combination of "*iterative*" and "*incremental*" process models with focus on process adaptability and customer satisfaction by rapidly and continuously deliver of working software product[13].
This chapter describes the most fundamental points of the Agile method, how it started and the adaptations that derived from it, like Scrum. At the end, it also explains how Athonet's adaptation of the Agile life cycle works.

3.1 The need for a new Software Life Cycle

The decade of 1990 represented a very important turning point for the digital industry. Computers were spreading everywhere and the software companies faced the so-called application development crisis.
The problem was that businesses moved too fast and within the space of three years, requirements, systems, and even entire businesses were likely to change. It meant that a lot of software ended up being incomplete or canceled halfway and the ones that made it through, even if they fulfilled the original objectives of the client, may not meet all the business needs[14].
SDLC models can be of two types:

- Iterative: enhances the evolving versions until the complete system is implemented and ready to be deployed
- Incremental: the product is designed, implemented and tested incrementally until it is finished

One of the most characteristic models used before Agile was the Sequential model (or Waterfall). The Waterfall model became very famous because it has many strong points as:

- Uses clear structure
- Determines the end goal early
- Transfers information well

On the other hand, it became obsolete when projects started to be more dynamic and complex. Some of its disadvantages in modern software projects are:

- Makes changes difficult
- Excludes the client and/or end user
- Delays testing until after completion

The excessive documentation, the forceful binding to the unchangeable decisions made early in the project and the little communication with the client brought to the need of a new model that prioritizes the product and the stakeholders over bureaucracy.
Those things frustrated people like Jon Kern, an aerospace engineer in the 1990s that with other figures from different industries "*were looking for something that was more timely and responsive*", as he noted. He was one of 17 software leaders that started meeting informally and talking about ways to develop software in a simpler way without the excess of documentation and other strict rules.
These talks led to the now famous "*Snowbird*" meeting (in Utah, February 2001), when the "*Agile Manifesto*" was written down and published.

3.2 The Agile manifesto

The Agile Manifesto is a brief document built on four foundational values and twelve supporting principles for Agile software development[15].
The four values written on the official website[16] are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The responsiveness of people and embracing the importance of changes are the fundamentals of Agile. Although documentation is secondary, it's important to note that Agile streamlines documentation and does not eliminate it.
These twelve principles emphasize things like "*early and continuous delivery of valuable software*" and "*continuous attention to technical excellence*", which are:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

All of them are important but, in my opinion, the ones that add the most value to the Agile line of thought and that differentiate it from other methods are the second, the fourth and the sixth: they represent the intent of placing the product and the customer above everything else, allowing the use of small informal meetings (even if the decisions should be recorded) and the easy change of requirements because the client is always involved, even as an end user

(tester).
Each Agile methodology applies the four values in different ways. However, all of them rely on these values to guide the development and delivery of high-quality, working software[17].

3.3 Agile's little big cousins

While Agile's manifesto contains values and principles, these are not prescriptive. In fact the manifesto does not outline specific processes, procedures or best practices. The goal was not to develop a rigid framework but rather create a mindset for software development. Agile is a blanket term that describes a set of software development principles.
There are many methodologies that derive from Agile's thinking, the most famous ones, according to the annual survey[18] from VersionOne's team are:

- Scrum
- Scrumban
- Kanban
- Extreme Programming (XP)

The essence of Scrum is being "*agile*" as in fast: having a small team that is highly flexible and adaptive.

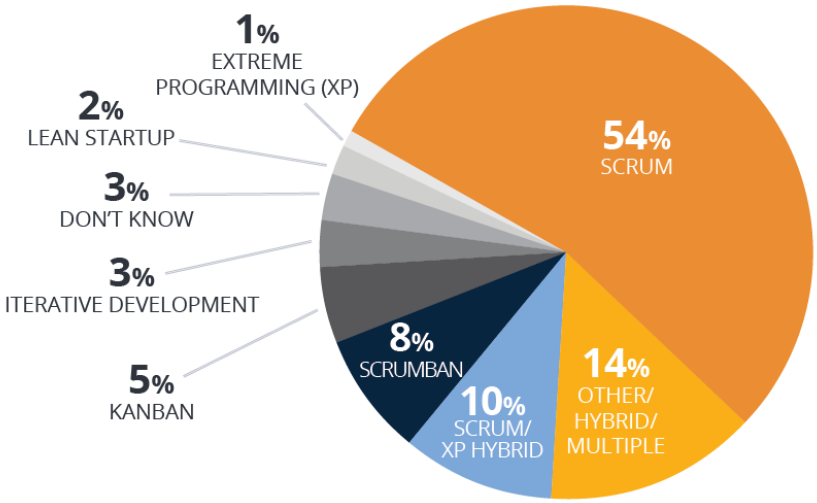

Figure 3.4: Percentage of the use of Agile derived methods according to VersionOne

This survey is quite interesting because it provides information from small and big companies that want to share their experience. A very important fact to be noted is that although technology companies are the ones that have participated the most to the survey, there are other industries that use Agile and are interested in sharing their experience.
In the following figure there are some percentages that indicate some areas of the companies that participated to VersionOne's survey.

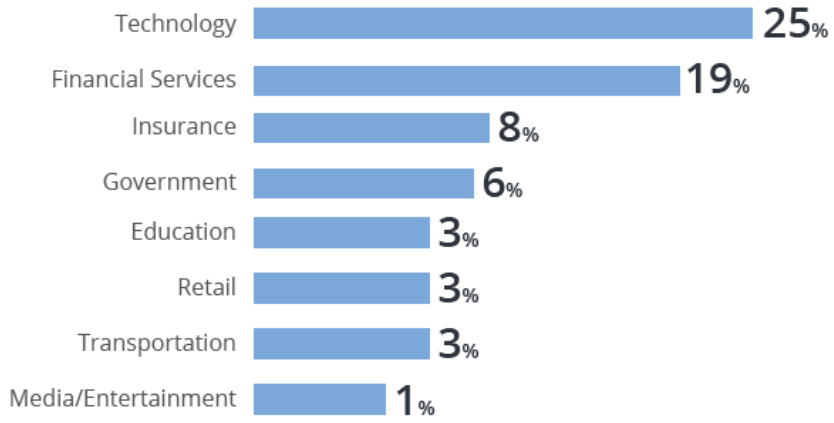

Figure 3.5: Areas of some of the companies that participated to VersionOne's survey

According to the previous figure[18], only 25% of the companies work in the technology business.
This survey states that for the year 2018 (which marked their 13th annual report) the reasons for adopting Agile were productivity, improving team morale, reducing product risk (with a lesser percentage than the previous year) and reducing project costs.
The measures of success mostly cited by the respondents were customer, or user, satisfaction and business value.
According to these companies, some of the most indicated benefits of adopting Agile are:

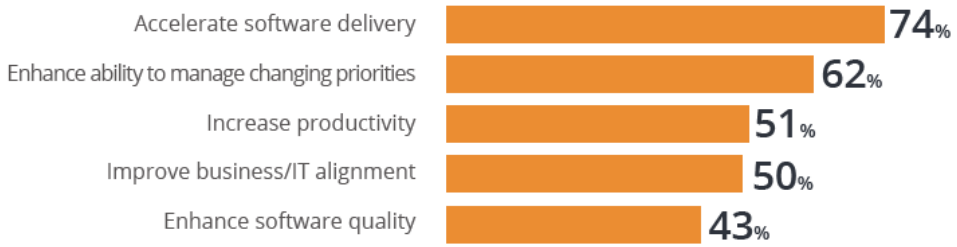

Figure 3.6: Some of the most voted benefits of adopting an Agile methodology

According to the previous figure[18], 74% of the companies the Agile methodology accelerates software delivery.
Also the questionnaire contained a question about what are the recommended Agile project management tools.

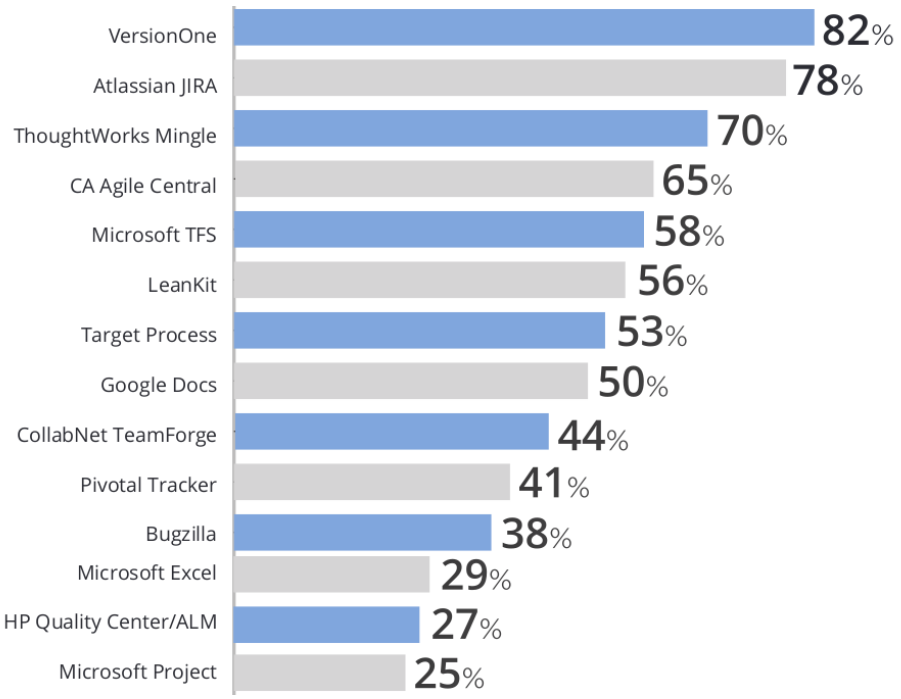

Figure 3.7: Some of the most recommended Agile management tools

As we can see Jira is the second most recommended one.

3.4 Agile in practice

Let's introduce the methodologies that two large companies, Spotify and Amazon, have derived from Agile's line of thought. These businesses succeeded using their own ad hoc methods, and now they have chosen to document and release them for others to use.

Spotify

Spotify has become the most popular music streaming application. It was launched in 2008 and now has more than 1600 employees. The success of this company is rooted in their Agile adaptation: the "*Spotify Tribe Engineering Model*".

This is composed by:

- Squads
- Tribes
- Chapter
- Guild
- Trio
- Alliance
- Chief Architect

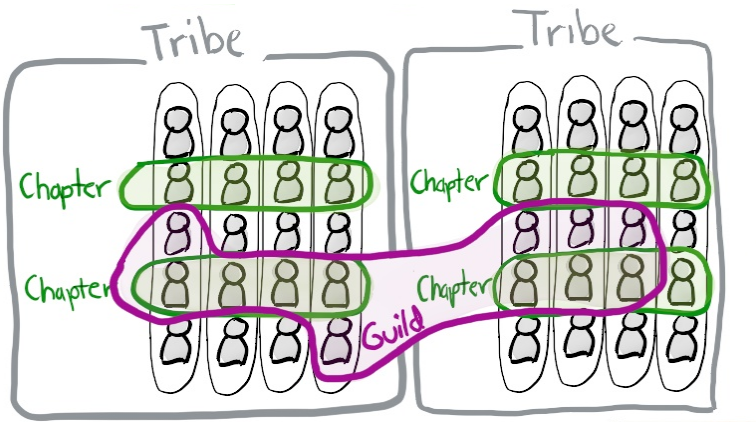

Figure 3.8: The "*Spotify Tribe Engineering Model*"

These groups and roles are very coherent because the bedrock of the Spotify Tribe model is autonomy and trust[19]. This methodology is introduced on Spotify's lab website as "*Spotify engineering culture*"[20].

Amazon

Amazon's organization can be seen unusual among other large firms. This company is characterized an ubiquitous "*customer-obsessed*" mindset. Everyone is expected to know about the data that is generated by customers so that Amazon can enhance the impact of what it offers.
The methodology adopted by Amazon is described, like Agile's manifesto, in 14 "*leadership principles*" on their jobs website[21].
One of the most famous quotes by Jeff Bezos, Amazon's CEO is "*if a team couldn't be fed with two pizzas, it was too big*".
As Bezos explained: "*The Two-Pizza Team is autonomous. Interaction with other teams is limited, and when it does occur, it is well documented, and interfaces are clearly defined. [...] One of the primary goals is to lower the communications overhead in organizations, including the number of meetings, coordination points, planning, testing, or releases. Teams that are more independent move faster*"[22].

3.5 The roles in Agile

A distinctive characteristic of the Agile methodology is its definition of roles: they are not fixed positions, any given person can take on one or more roles and switch over time, also any given role may have zero or more people in it at any given point in a project[23]. The ideal team is considered to be composed of five or six people.
These roles are:

- Team leader: team coach or project lead in other methods (Scrum-master e.g.), he is responsible for facilitating the team, obtaining resources for it, and protecting it from problems
- Product owner: an executive or key stakeholder, the Product Owner has a vision for the end product and a sense of how it will fit into the company's long-term goal
- Team member: developer or programmer, is responsible for the creation and delivery of a system
- Stakeholder: any other person that has direct or indirect interest in the project

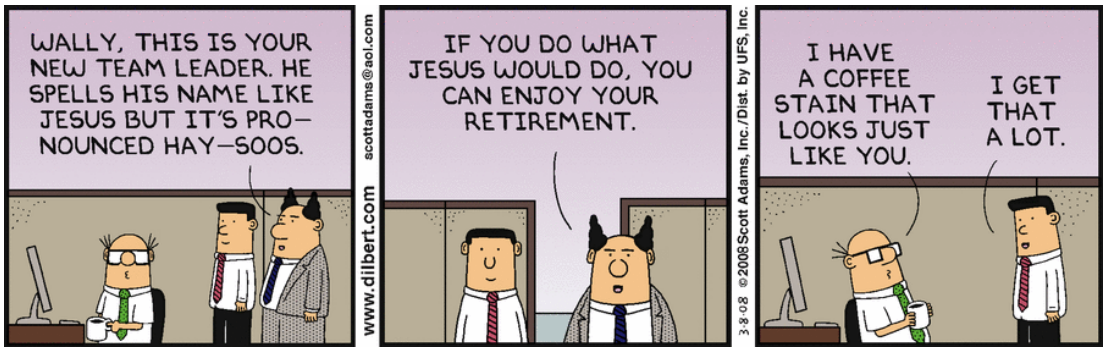

Figure 3.9: Dilbert on team leaders

3.6 Time cycles and metrics

Scrum teams coordinate development into time-boxed "*Sprints*". Outside the Sprint, teams organize and estimate the amount of work that can be concluded. The goal is to have all the forecasted work completed by the end of the sprint.
Metrics are used to track the completion of tasks, these are called "*burndown reports*" and are graphs where the X-axis represents time, and the Y-axis refers to the amount of work left to complete, measured in either story points or hours.

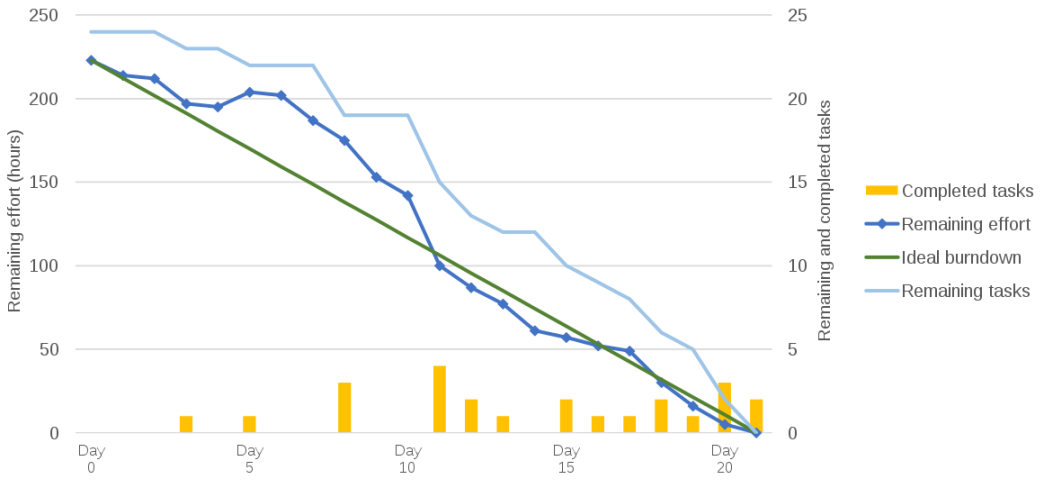

Figure 3.10: Example of "*burndown report*"

The Sprint is one of the most important structural bricks in Agile, the other principal ones, according to Atlassian's "*Agile Coach*"[24], are:

- Stories: short requirements or requests written from the perspective of an end user
- Epics: large bodies of work that can be broken down into a number of smaller tasks (called stories)
- Initiatives: collections of epics that drive toward a common goal
- Themes: large focus areas that span the organization

Figure 3.11: Structural bricks in Agile according to Atlassian's Agile Coach

3.7 Disadvantages of Agile Software Development

Despite the benefits offered by the Agile model, transition a company's way of working to it is not that easy and if done wrong it may make damage instead of good. According to the American entrepreneur Adam Fridman, here are the drawbacks[25] of Agile:

1. Less predictability: developers may not be able to quantify the full extend of the required effort
2. More time and commitment: a constant interaction, with many face-to-face conversations, is required
3. Greater demands on developers and clients: extensive user involvement that impacts the quality and success of the project
4. Lack of necessary documentation: new members that join the team may need more time to understand the project
5. Project easily falls off track: if a consumer's feedback or communications are not clear, a developer might focus on the wrong areas of development

The following image contains the percentages that, in VersionOne's previously presented survey on the "*State of Agile*"[18], represent some of the most cited "*Challenges Experienced Adopting & Scaling Agile*":

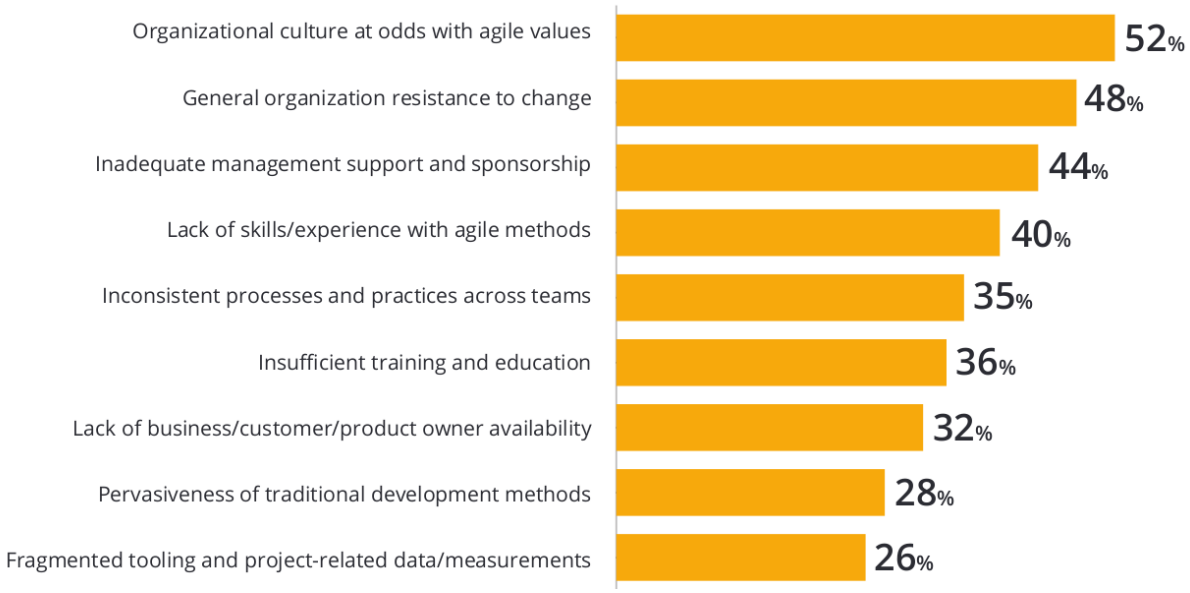

Figure 3.12: Some of the "*Challenges Experienced Adopting & Scaling Agile*"

3.8 What Agile variant Athonet uses

As many small companies, Athonet will not strictly use one Agile implementation. This is also due because of the nature of their product that is not released like simple software patches but it is presented in a more monolithic version. After a research for the available software development life-cycle tools on the market, they chose to try Atlassian's Jira in tandem with Confluence.

As said in Chapter 5, the managers were keen on the idea of Kanban because it allows employees to "*come in the morning and choose what they want to work on from the backlog*", as said by the development team manager, without giving them a two week period of time but letting them complete the tasks in time for the following release. But they also liked the idea of having a tool that could transition from a type of project to another in case they want to start applying stricter rules in the future.

Figure 3.13: Dilbert on Agile programming

In an Agile project the team takes care of the tasks and the project leader takes care of the team.

Jim Highsmith

4

Jira and Confluence: the essentials

Now that the main concepts of Agile and it's derivatives have been explained, let's understand how Jira and Confluence work. These are proprietary tools developed and maintained by the Australian company Atlassian.

Figure 4.1: The logos of *Atlassian*, *Jira*, *Confluence* and *Jira Service Desk*

Jira is an Issue Tracking System that was first released in 2002, it's name is a truncation of "*Gojira*", the Japanese word for "*Godzilla*". This is a reference to another ITS that was dominating the market at the time, "*Bugzilla*". At the time of writing, the main competitors of Jira are other software like Redmine, VersionOne, PivotalTracker, Workzone or integrated ITSs in repositories like GitHub's or GitLab's issue trackers[26].
Athonet's previous choice was Redmine because it is open source (this implies it doesn't need a paid license), it has a medium-large community of people that use it and maintain it behind

and the plugins allow the integration with other internally used tools like repositories or software for reporting customer requests.

Figure 4.2: *Redmine*'s logo

Confluence, on the other hand, has been designed as a collaboration platform for sharing knowledge like documents, product specifications, meeting notes and can be used as a wiki for internal use or for releasing information to the clients.
Atlassian released the first version of Confluence in 2004, saying its purpose was to create "*an application that was built to the requirements of an enterprise knowledge management system, without losing the essential, powerful simplicity of the wiki in the process*"[27].
Since the first releases of these products, Atlassian has developed and acquired new tools like "*Bamboo*", "*Clover*", "*Crowd*", "*Crucible*", and "*FishEye*", all orientated towards collaboration, content sharing, issue tracking, time scheduler, etc.
Both Jira and Confluence are written in Java.

4.1 What can these tools do

These tools are made to be flawlessly integrated with one another, not only because they are made by the same company but they are strictly correlated in their intents.
Integrating an Issue Tracker with a platform able to share documents and thoughts allows a more granular analysis of project requirements. This means that the company can store meeting notes and documents related to the project in Confluence, and when they are ready to move into the development phase, convert them to Issues inside Jira.
Plugins can extend by far the usage and integration with other tools and the Atlassian Marketplace [28] is full. Although there is a license that has to be paid, if used correctly this software can allow the company to save money that could otherwise be lost in badly managed resources like time, documentation and even people.
Let's understand them better.

4.1.1 Jira

Over the years Jira has become such an important software that Atlassian had to separate it in three specialized components, each with its own scope. Here is a table from Jira's official documentation [29] that contains all the major differences:

	JIRA Core	JIRA Software	JIRA Service Desk
Simple business projects	✓	✓	✓
Workflow editing	✓	✓	✓
Powerful issue search	✓	✓	✓
Dashboards	✓	✓	✓
Basic reporting	✓	✓	✓
Agile boards		✓	
Backlog planning		✓	
Agile reports		✓	
Development tool integration		✓	
Release hub		✓	
Queues			✓
SLAs			✓
Confluence knowledge base integration			✓
Detailed service metrics			✓

Figure 4.3: Differences between *Jira Core*, *Jira Software* and *Jira Service Desk*

The three editions of Jira have the same look and feel to provide a consistent project experience, but there are some differences that characterizes these standalone products [30].

Jira Core

The core version is a *business oriented* project management software for general team project collaborations and workflow approvals of not technical tasks that can eventually be related with developers' issues. It's targeted users are non technical teams like the business, legal, operations and marketing departments.

Jira Software

Its purpose is to plan and track the trend of the project by managing releases and creating reports from real time data in order to improve the teams' performance. Each project can have its own workflow and can be integrated with other Atlassian tools for better coverage.

Jira Service desk

This software's main purpose is to handle customer requests and provide them with a user friendly help center, that, in combination with Confluence, can create a rich navigable knowledge base. It can be integrated with Jira Software so IT and developer teams can collaborate on one platform.

Jira Portfolio Plugin

Portfolio [31] is an add-on for Jira Software that enables teams to build plans and track progress. Its main purpose is creating real time roadmaps that can include issues, at various levels by applying filters to see only the interesting ones.

4.1.2 Confluence

As described earlier, Confluence is a collaboration platform. It allows to create spaces of different categories that can be associated with Jira projects. There are many different space types (like knowledge base, documentation, software project, etc.) that contain templates used for creating documents.

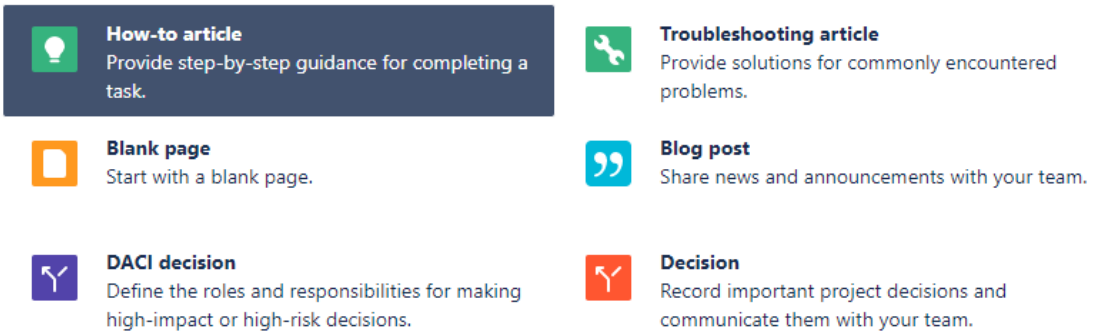

Figure 4.4: Some of the custom templates for documents that can be redacted in Confluence

Companies use Confluence to create public wikis for their customers, e.g. Akraino [32].

4.2 Key concepts for Jira

Jira has its own terminology that needs to be understood in order to completely master the software. The key terms that must be known, according to Jira's guidelines[33] are:

- Issues: a work item of any type or size that is tracked from creation to completion
- Projects: a collection of issues that are held in common by purpose or context
- Workflows: represent the sequential path an issues takes from creation to completion

Figure 4.5: Dilbert on issues

4.3 How Athonet uses these tools

As anticipated, Athonet has chosen Atlassian's Jira and Confluence because they need a single solution that is coherent and that could provide them an unique access for all the company figures to the projects and their related documentation. Plus, Jira was also used in the past by some employees at their former company, so there was more familiarity compared to other tools.

Although Confluence's potentials as a documentation writing software, it will be impossible to eliminate the use of Microsoft Office programs (Word, PowerPoint, Excel) to share documents, presentations and other content with clients. This is because of their simplicity and massive use for producing records in a fast and user friendly way.

These tools can become complex, it is necessary to understand the scenario they can be used in and make a plan for transitioning from the old software.

It is important for the company to adapt the tools for it and not change their entire way of working to accommodate to the new software.

4.3.1 Development

Compared to Redmine Jira has many more useful functionalities and on top of that it has a richer User Interface (UI).

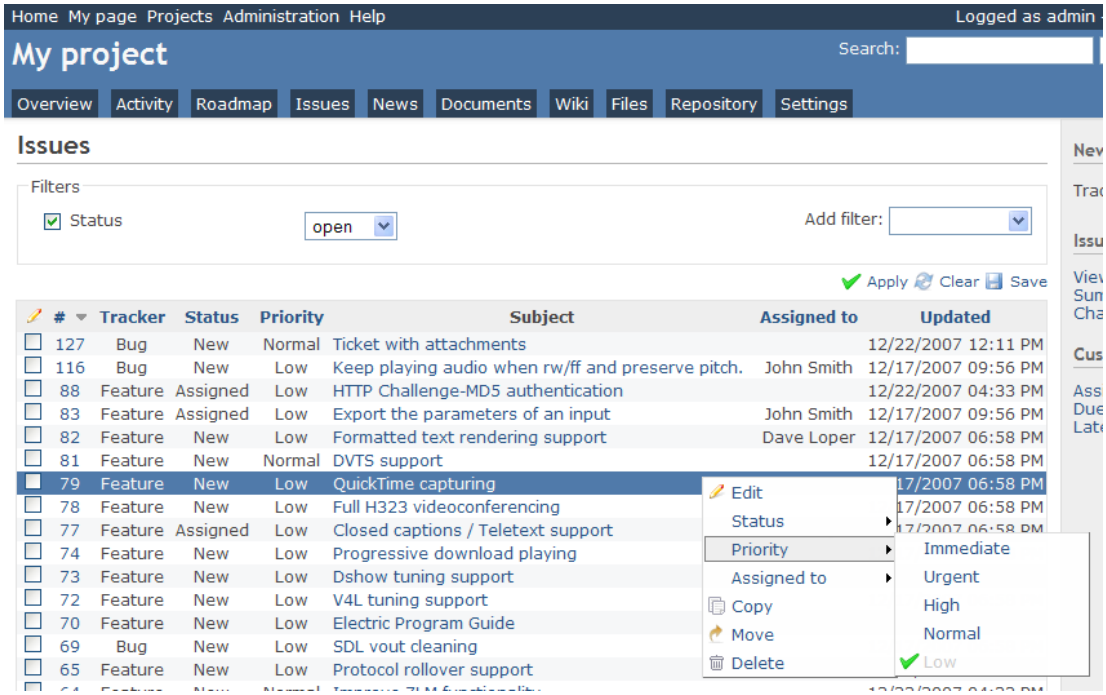

Figure 4.6: Example of Redmine's interface

The development team will be the one that will use it the most, and because of that it is important that it can integrate with GitLab. With the GitLab Listener plugin [34] Developers can interact with Jira's issues from the messages in the repository's operation.

4.3.2 Management

The most important feature the management department needed was the live roadmap. It's a key element for this kind of company since it allows to keep an eye the trend in development and manage the releases. Jira's Portfolio allow to apply filters that can display a different granularity on what is shown, this means that it can be used at more levels in the company's hierarchy.
With the kind of tools currently in use it's not possible to have such a smooth integration.

4.3.3 Client interaction

Although Service Desk would be the ideal software to allow the interaction with clients, Athonet has not yet thought of using it in such a way, but there is the possibility to do so in the future.

4.3.4 Internal documentations

Confluence substitutes the multiple wiki software that were previously installed, allowing an integration with Jira and offering a richer experience for the employees.

4.4 The Atlassian Community

Athonet has chosen wisely to buy the licenses Jira and Confluence over other tools and over creating their own internal solution, which would have meant reinventing the wheel. Compared to other tools, Atlassian has built a large community around its products, allowing people to ask and answer questions on a blog, request for new features, open tickets, view the dates and changelog of next releases and report bugs.

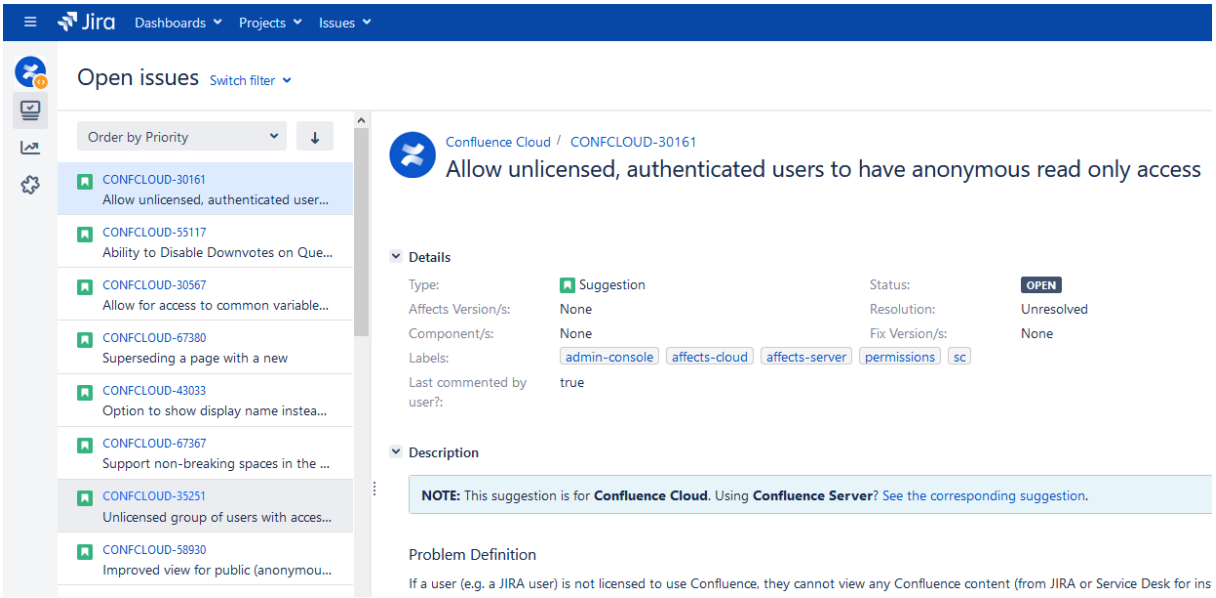

Figure 4.7: The issues related with Jira and Confluence are handled by a dedicated Jira Cloud instance

This allowed me to easily find not only the resolutions to some implementation problems that occurred during the installation, but also tips on how to better configure the software.

Scrum means "Waterfall but we don't have time for analisys"
Kanban means "Scrum, but we don't have time for Sprint planning"
Agile means "We have no process but we do use Jira extensively"

@mikeveerman on Twitter

5 Projet implementation

This chapter is the core of this document and describes the way that this project has been implemented according to the planning described in Chapter 2.
It is structured in four main sections that go over the main time periods presented in the "*Piano di Lavoro*" document.
Since the scope of the project was to create a demo environment to show the capabilities of Jira, Athonet has bought ten user licenses for Jira Software, ten for Service Desk and ten for Confluence.
As described in Atlassian's documentation[35], these tools have three installation options:

- Cloud: hosted on their infrastructure
- Server: download and install on a local network
- Data Center: download and install for a large infrastructure

Because of Athonet's policies about storing internal data in the cloud, the IT department opted for an on premise, Server, installation. This kind of installment requires a one time payment for the software while the subsequent fees will be for support.

5.1 Learning phase

This phase corresponds to the first two weeks of the internship. As described in the "*Piano di Lavoro*", the main task in this first period was to understand what the tools are and what they

can do. The first important thing to do was to acquire knowledge on Jira and Confluence by researching them on the Internet. The first one I have started researching was Jira, this brought me to the official documentation[36], which is very well organized.

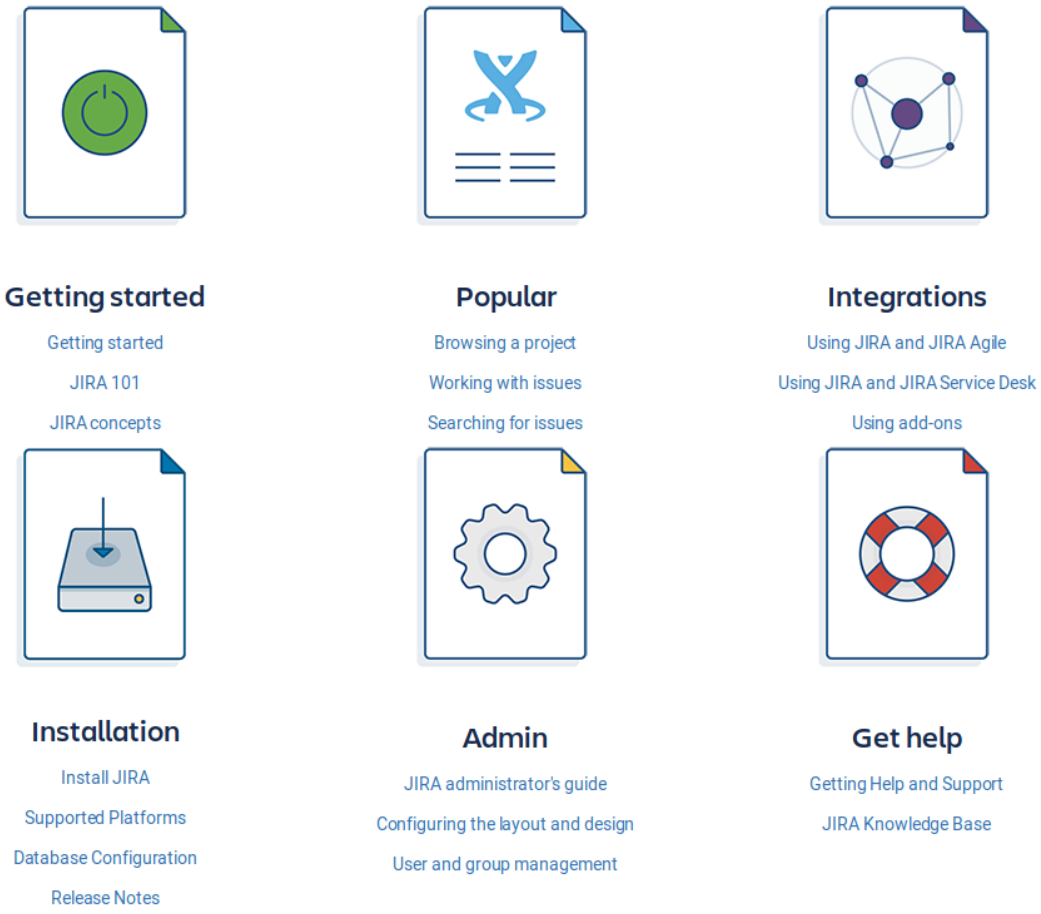

Figure 5.1: Screenshot of the sections from Jira's Documentation homepage from the Atlassian Support website

It is very easy to navigate because it's versioned for each release, major and minor, of the software, plus it contains links to related pages, including Confluence's documentation and Atlassian's blog. As said in 4.4 both Jira and Confluence have a bug reporting and issue tracking section in their documentation. If a web page is related to an issue, this is showed at the end of the page with its status and a link to its dedicated section.
The Confluence and Jira documentation are both written and hosted using a Confluence instance, showing how powerful can be this tool for handling a wiki for such a complex software that has a large userbase.

Confluence's documentation is structured like Jira's, easy to access and consult, other than being public and free to read, despite the software are not.
The second half of the first week was dedicated to configuring the environment. The IT department has chosen to create an instance of the Atlassian's software on a CentOS$_G$ Virtual Machine$_G$ (VM) with 512GB of storage and 32GB of RAM, connected to a special testing network domain for internship students. The Remmina$_G$ was used to remotely connect to the given VM.

Figure 5.2: Logo of the CentOS Linux distribution

5.2 Initial installation and configuration

Since the exhaustiveness of the documentation, the installation phase was anticipated so that there could be a more hands on approach. As for the previous phase, the installation of Jira was done by following the dedicated article on the official documentation[37].

Figure 5.3: Screenshot from the installation page of Jira in Atlassian's documentation

To store all the information regarding projects, users, etc., Jira needs a database. For the first installation, intended to be used in a testing manner, the embedded H2 DATABASE$_G$ was enough. It's important to note that the documentation says the H2 database is not suitable for production environments[38]. The first thing done after the installation was getting acquainted with the interface and understanding ho Jira's components interconnect with each other.
In order to do this it was necessary creating some mock projects and filling them with issues. Here is where the concept of Board come out in Jira.
Boards and workflows are very tied notions. Experimenting with workflows was one of the most important things to do, because these are fundamental in an Issue Tracking System's configuration and are strictly connected to Boards.

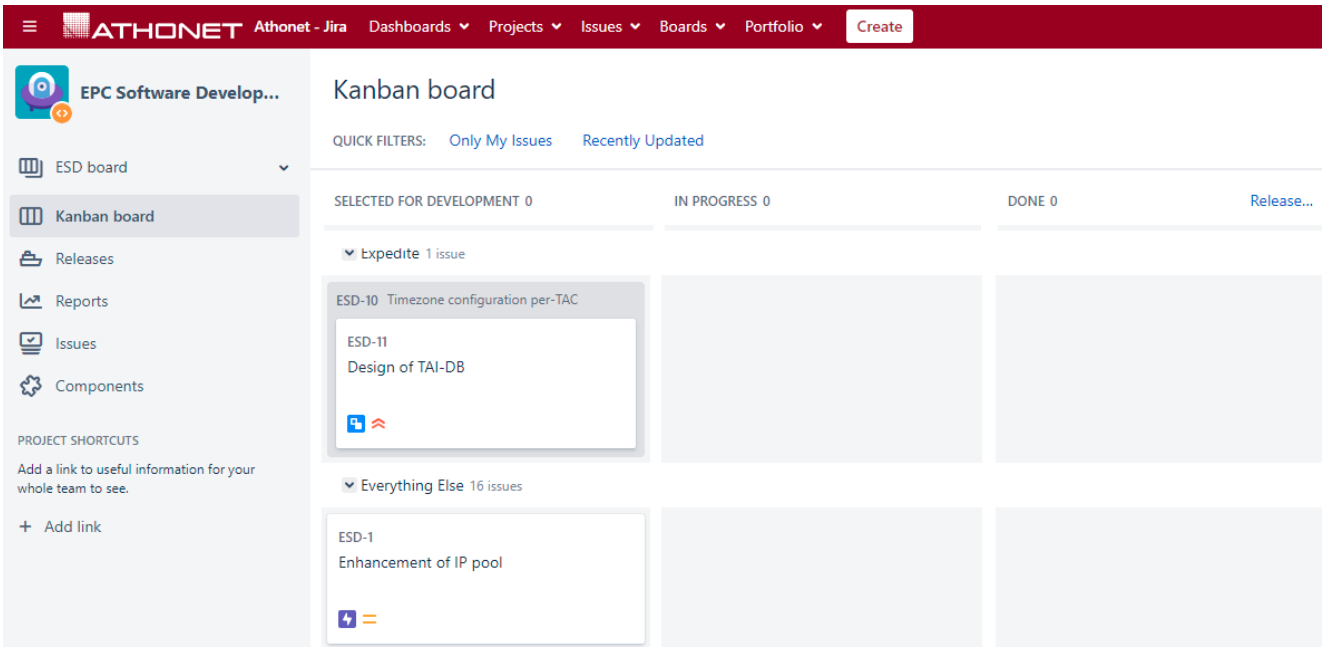

Figure 5.4: Kanban board from a Jira project

Installing Portfolio was the step that came after understanding the fundamentals of Jira and getting to know its basic features. This plugin, as told in 4.1.1 helps visualizing the issues on a roadmap, which was one of the most important requirements: *O05*.
As per the previous installation, Portfolio's was done by downloading and embedding the binary into Jira's instance.

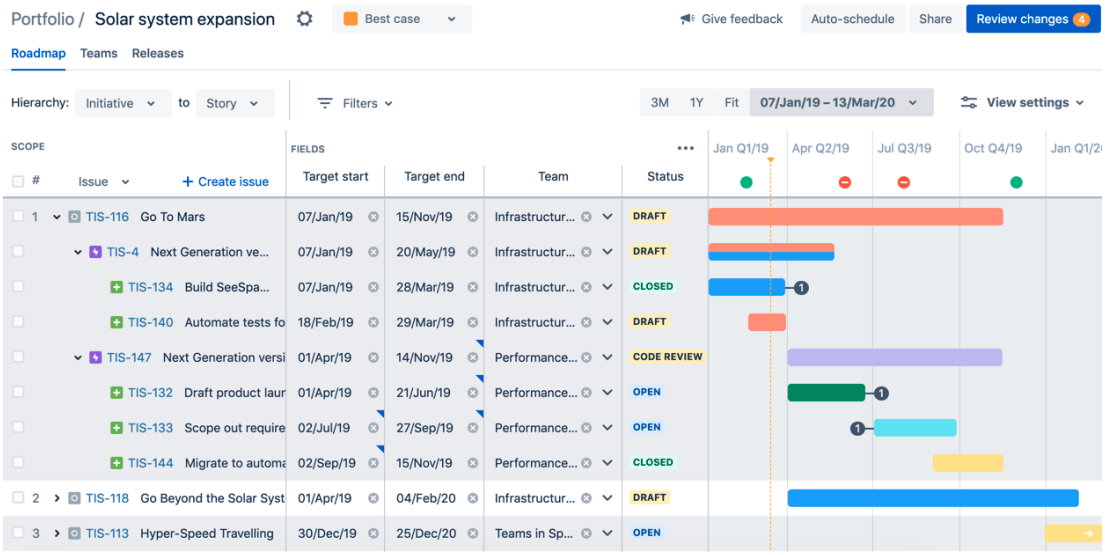

Figure 5.5: Screenshot of roadmap in a Jira Portfolio project[31]

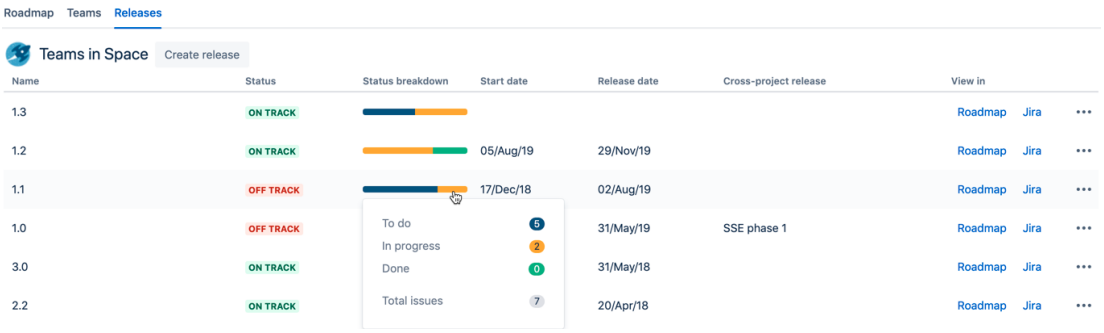

Figure 5.6: Screenshot of releases in Jira Portfolio project[31]

The CTO of the company and PriMo's product owner were interested in the possibility of using a real time roadmap to keep track of the work done while not going as in deep as seeing the tasks assigned to each developer.
After installing this plugin and creating plans for the previously created mock up projects, the next software installed was Jira Service Desk, to complete the configuration of the Jira instance. As described in 4.1.1 this piece of software is mostly used to communicate with the clients and having a portal from which a user can find information and request assistance with a product. In Jira Service Desk, the customer portal is the site where customers file and track requests. After installing this software, the first thin to do was creating a "*Help Center*"

to see from the client's point of view how it can be able to open an issue and what types he may have access to.

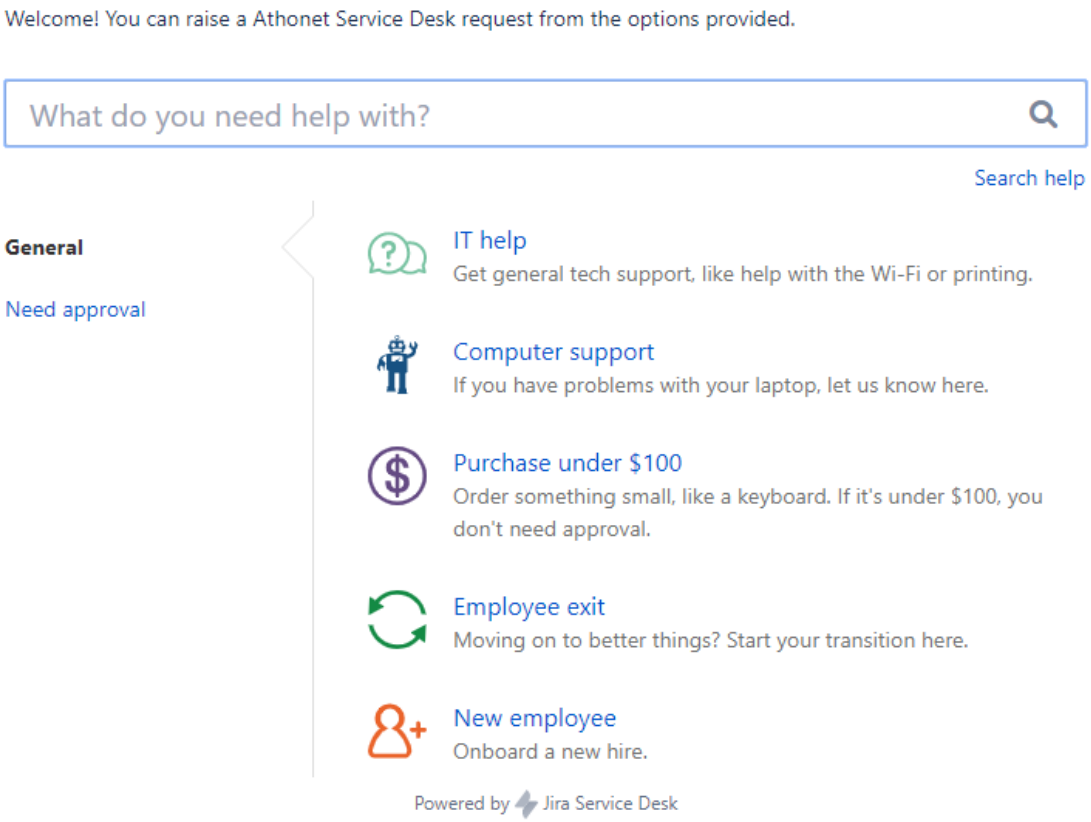

Figure 5.7: A Jira Service Desk portal

The installation of Confluence came after understanding all the basics of Jira and becoming familiar with it. For the newly installed software the H2 database was enough as for the previous one. Both software run on the same system, and their services are offered on port 8080 and 8090 respectively for Jira and Confluence via HTTP protocol.
During the installation of Confluence, the install wizard program allowed me to insert all data of the Jira instance in order to connect them. A knowledge base for the Service Desk project and a documentation space for the Software project have been generated in order to test the connection.
To allow a better integration between these tools, Atlassian allows to have a single database table containing the users information stored on either one of Jira or Confluence's instances. This was shown during the installation of Confluence as an option to link it with Jira's in-

stance was presented by asking for the administrative credentials for connecting to the database and using Jira's users. Jira and Confluence both use the open source Apache Tomcat web server. This particular web server has issues regarding the login when both software run on the same IP address: each program stores COOKIES$_G$ to allow users to log in without inserting each time username and password. On each log in though, users were automatically logged out of the system almost immediately. After a research on the Internet I found a post[39] in Atlassian's blog regarding this problem: the answer was that when run on the same IP address, cookies from Jira and Confluence go in conflict. This post was associated with a more detailed issue[40] in Jira's own portal.
The solution to this problem was to change the CONTEXT PATH$_G$ of both applications[41] in the "*server.xml*" file that contains the configuration of the web server. An example of changing the context path is:

`http://yourdomain.com:8080`

becomes:

`http://yourdomain.com:8080/jira`

For Confluence, where the default port is 8090, the context path would become:

`http://yourdomain.com:8090/confluence`

Contrary to what was said in the beginning, the tutor had communicated me that the IT department opted to maintain for using GitLab[10] instead of installing BitBucket[42], another Atlassian software. This because GitLab was already in use by the developers, so they would not need to learn a new software that could affect their work routine.

Figure 5.8: GitLab Logo

Connecting these tools together means that a developer can interact with Jira's issues from a GitLab project by typing its ID in the messages that he uses for commits, comments, merge requests and so on. GitLab's documentation has a dedicated page with instructions, examples and a guide that allow configuring both tools[43].

This functionality allows GitLab to interact with Jira's default workflows, which sometimes are too simple for some software projects.
Since the license for GitLab's hosted version is not "*Premium*" (or "*Silver*" for the online one) there is no interaction the other way around, from Jira to the repository[44]. Since these tools are not strictly related, there is no direct project mapping from GitLab to Jira, a commit message in the repository may reference multiple Jira issues from different projects, so it's the programmer's discretion to comment wisely. Later in the text it's described the installation of a plugin that allows these tools to connect with more functionalities.
A demo that touched all the previously viewed arguments was set in place for showing the progress made to the tutor. Because his availability at the moment was very little, from the time the meeting was set to the actual moment of it happened, the interface of the system was customized by adding the colors of the company and its logo.

Figure 5.9: Custom menu interface for Jira

Figure 5.10: Custom menu interface for Confluence

During the meeting, the tutor liked the progress that was made and asked for more elaborate mock projects for presenting the tools' combined functionalities to other company figures.
The projects registered until that moment in Jira and their related spaces in Confluence were deleted and, as hinted in 2.3, a snapshot of the VM hosting the software was taken.
Until this point the completely fulfilled objectives were: *O01*, *O02*, *O05*, *D03*.
Although the connection between Jira and GitLab was established, to fulfill O04 it was required that GitLab supported custom workflows for Jira Software projects.

5.3 First realistic mock projects and feedback

The previous paragraph marked the first three weeks of the internship. Starting the fourth week the task was to implement more realistic projects in Jira, connecting them to Confluence providing them with documentation and creating a link with GitLab. The objective

was to expand the demo so that it could be shown to other members of Athonet to understand, both me and them, how these software can be used in their departments.
The new Software projects created in Jira were called "*EPC*" and "*Dashboard*", linked to their documentation spaces in Confluence, respectively "*EPC Documentation*" and "*Dashboard Documentation*". To show the functionalities of Jira Service Desk as a knowledge base portal (sharing internal documents between employees), the "*Athonet Internal Wiki*" project was created.
For the first project a more articulate workflow has been implemented to resemble the realistic evolution of an issue inside the company.

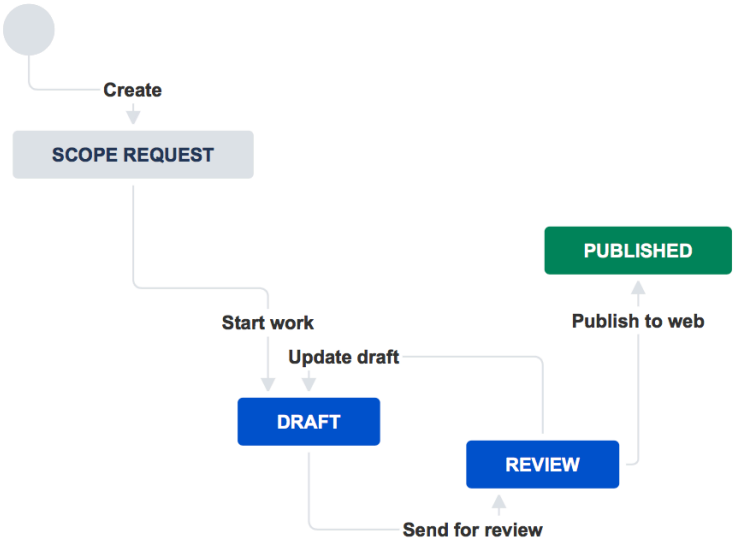

Figure 5.11: Custom example workflow that can be implemented in a Jira project

While working on the mock projects and creating issues, it was useful to note down all the most important customization that an administrative user can use to set up the software, not only for Athonet's specific purposes but in general.
To make the project more realistic three new user groups were created, besides the default ones, to which there were assigned three users each:

- Management
- Verification
- Developers

Every group had custom permissions allowing to demonstrate how basic security rules work in these tools (requirement *D06*).
A first draft of how the menu for creating a new issue can be customized was done as well, although not final (requirement *D05*).

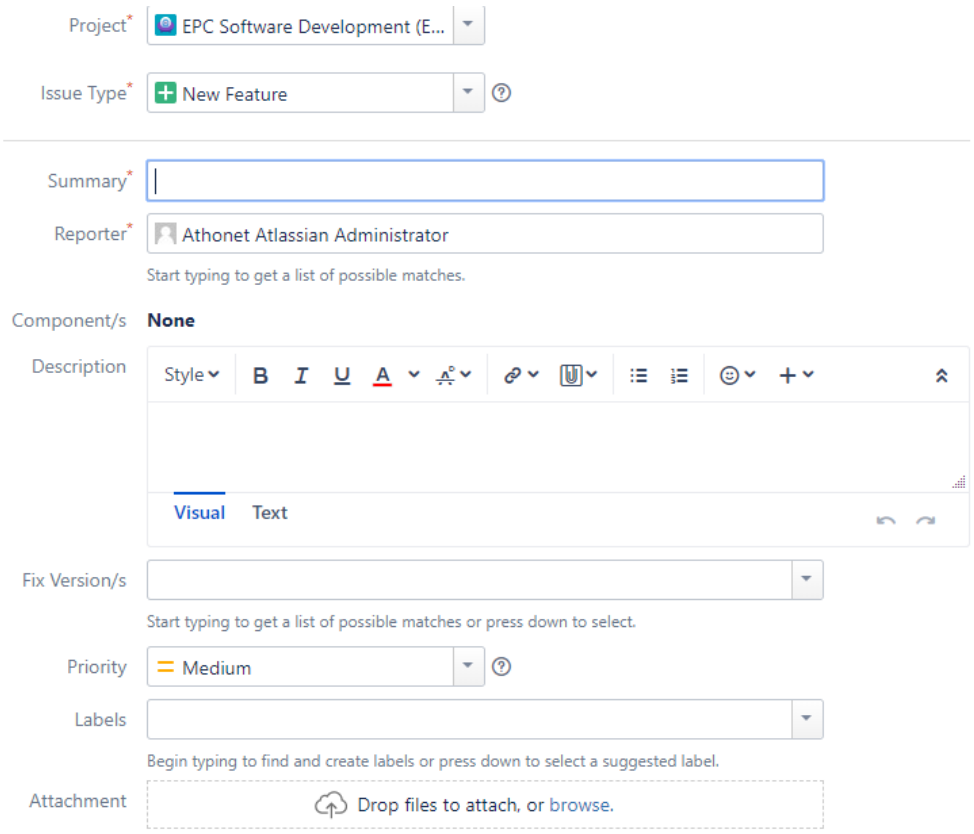

Figure 5.12: Draft for custom issue creation and editing menu in Jira

After talking to the tutor about the made, he was able to set a meeting with himself, the verification and developer managers. Slides were prepared in order to explain the nomenclature in Jira, described in this document in 4.2, and the features that have been paired to the demo projects.
The core of this meeting was a discussion between the managers focused on understanding how their ongoing projects in Redmine could be implemented in such a way that the employees would not be forced into using a strict Agile methodology, but to accommodate and let them understand how these tools work without creating chaos.
There was a discussion on how applying a strict Agile methodology could impact their time

division. Agile provides sprints that by definition last two weeks, and these produce deliverables that bring added value to the project. Athonet's line of work though is focused on adding value to the product upon new releases that are scheduled differently by their purpose:

- Bug fix
- Minor release (for patching issues with the code or to improve performance)
- Major release (for new features)

Sprints would disrupt the routines of their developer team, the verification team and of management. In this case it would not be the tool that adapt to the company but vice versa the company would need to change in order to use the tool and would result counterproductive.
The developer manager stated that he wanted a way to measure productivity, so I showed him the "*Reports*" section that is present for all kind of Jira projects. Here there are the burndown report charts that have been explained earlier in the text, in 3.6.

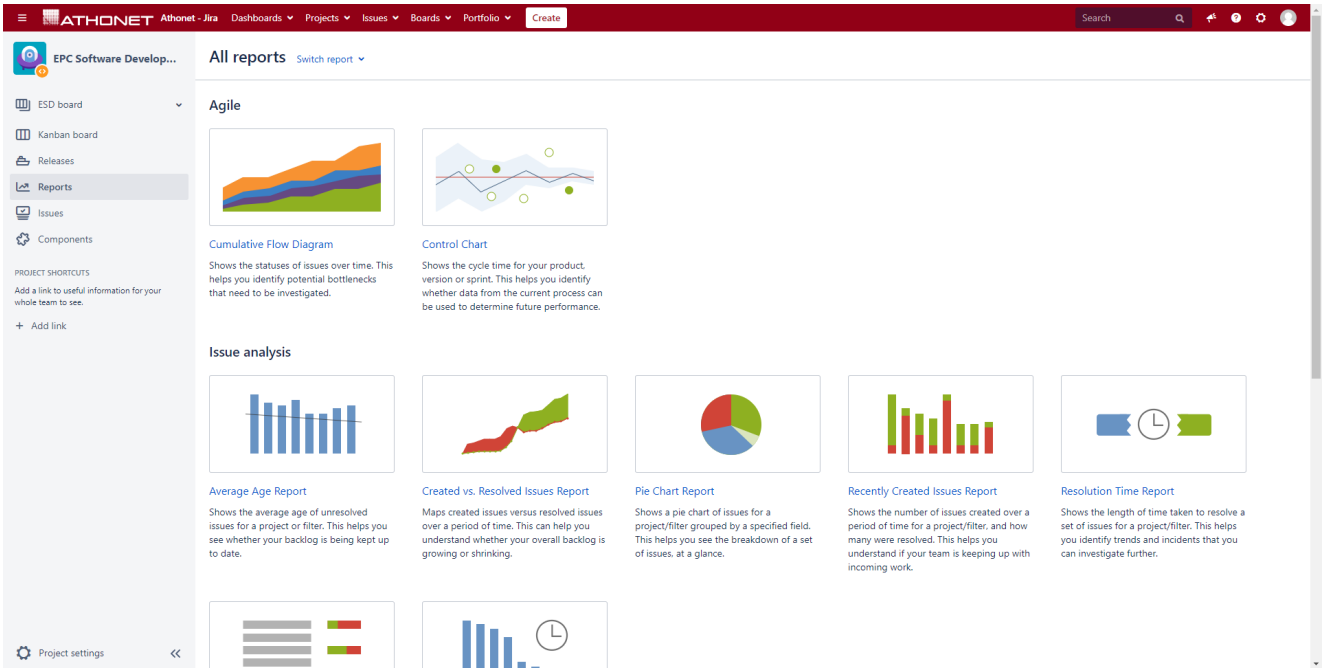

Figure 5.13: Kanban board from a Jira project

Both managers were more pleased to see that Kanban boards resembled better their way of working, even if they want to maintain some of Scrum's peculiarities. The developer manager also said that it would be better for the company's way of working since developers could plan their tasks based on what has to be completed for the next release, so for a longer period of time.
This implied there would be a more particular workflow that had to be implemented and the managers were pleased to see that it could be easily done, as for customizing the issue creation and modification menu.
They were also glad about the internal wiki project because it suggested there could be one single tool for both internal documentation regarding clients and general notes for employees. The following image is an example of how an article is visualized in this wiki.

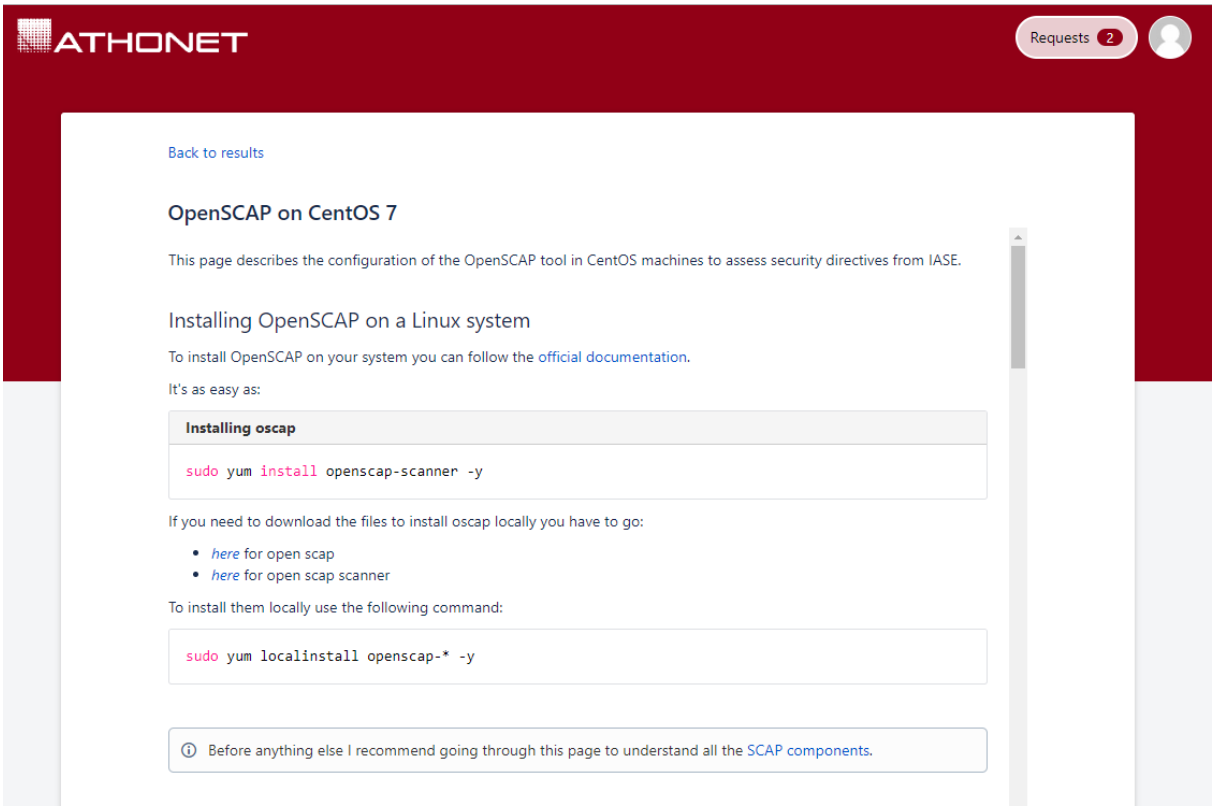

Figure 5.14: Screenshot of article in Jira Service Desk

Articles like this correspond to pages written in a Confluence knowledge base space connected to the Service Desk project.

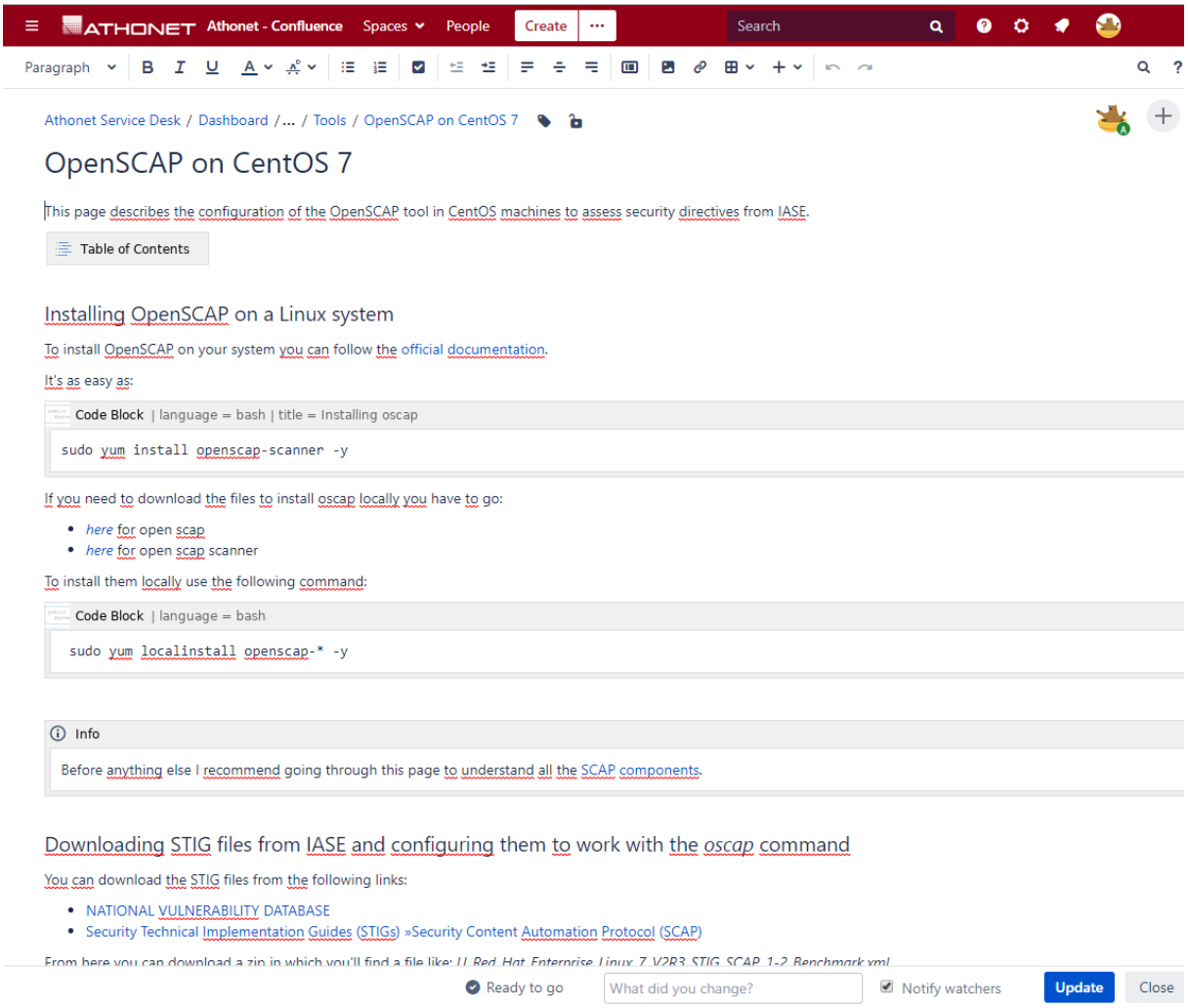

Figure 5.15: Screenshot of a page in Confluence that is being edited

The company members needed time for internal discussions that had to be done in order for the managers to understand how to adapt the workflow they had in Redmine for Jira and how to include Confluence in the mix. In this period the documentation was elaborated, anticipating it from the original plan in the "*Piano di Lavoro*".
The first one to be produced was the "*Installation an Configuration Guide*", written for the IT administrators and the people that would maintain, configure and eventually reinstalled if needed, the software. After this was drafted and approved from the tutor, I worked on the "*Final User Guide*", written in the wiki Confluence that was previously demoed. Both regarding requirement *O03*. Later on, a new meeting was scheduled with the product owner,

who wanted to understand better Portfolio and how the concept of "*product*" matches the "*project*" denomination in Jira.
Project and product sound similar and the two concepts are often confused with each other. A project is a temporary endeavour, with a clear definition of what needs to be delivered and by when, it has a beginning and end date. A product is designed to continually create value for customers by solving their problems[45].
For the product owner, one of the most important things was to see how releases are mapped inside the software, since Redmine did not offer such an intuitive interface and did not allow to see them on a timeline and applying custom filters. He approved the progress made after seeing it and also said that he wanted the credentials to start using the software and see how to map the ongoing projects in Redmine to new ones in Jira or how to eventually migrate them, as per requirement *F02*.
After using it for a couple of days and talking with the other managers, he told me how they want to implement the evolution of an issue. Since they were not sure letting users access Service Desk directly, an employee would create an issue that would represent the customer request through service desk. The customer request could be a new feature, a bug report, require support, etc. This would then be picked up by a team that would bring it inside a Business type of project by creating a linked issue. This project would have a particular workflow and contains the more high lever tasks like documentation, meeting notes, feature statuses, etc. These shared notes and documents would be stored in Confluence, where there would be links to them. After they have been approved for production another team would create a linked issue inside the software project. This would most likely be divided in many sub-issues by the team leader and these would be visible by the developers. Each developer could then pick an issue to work on and handle it as he wants by, for example, dividing it in smaller sub-issues.
The next week another meeting was scheduled with more company figures: a senior developer, the business strategy manage, the managers from the previous meetings, the CTO and the tutor.
As in the other meetings I have presented the software, the major use cases and progress done, referencing the objectives achieved and showing them the working system by creating issues in projects that would resemble the way of working described by them.
Even with the requirements changes that occurred, I was on schedule with the planning, a more detailed view can be seen in the Gantt chart in A.4.

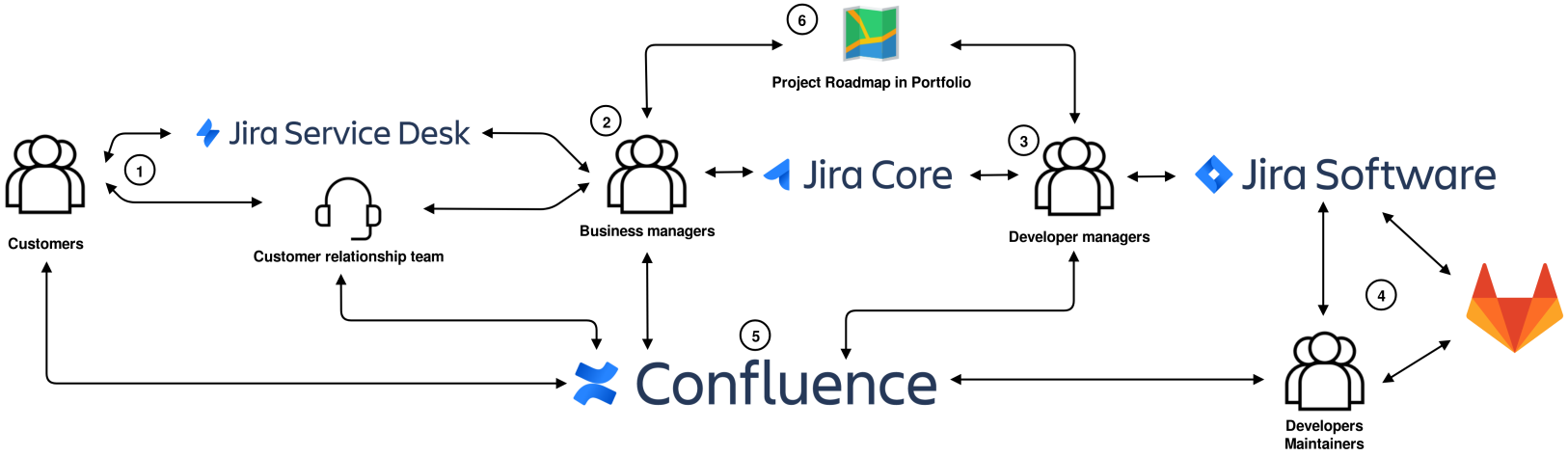

Figure 5.16: Evolution and transitioning of an issue

The main steps that an issue passes are:

1. The customer reports a bug or feature request to the Customer Relationship team or through a Jira Service Desk project
2. The request is then picked up by the Business managers that discuss it and link it to a Jira Core issue in a dedicated project
3. The features that are ready to be developed are sent in the Jira Software project by the development team managers
4. The developers choose what to work on and commit to GitLab, which is connected with Jira Software, using special keywords
5. Confluence is fully connected with every other Atlassian tool and allows users to store notes and documents
6. Jira's Portfolio plugin is always up to date with the status of the issues and can be consulted by the managers (or developers)

5.4 Transitioning into Production and final feedback

To better connect GitLab with Jira the plugin "*GitLab Listener*"[34] was installed. This allows for wider connectivity between the two products supporting custom workflows and more transitions with commit messages like:

```
New version #refs ALLGEN-12 GTTSLT-2 #time 1h 15m #action ToDone
```

This kind of command references two issues in two different Jira projects, telling the time spent and transitioning both to the "*Done*" column. Installing this plugin allowed to completely fulfill requirement *O04*.

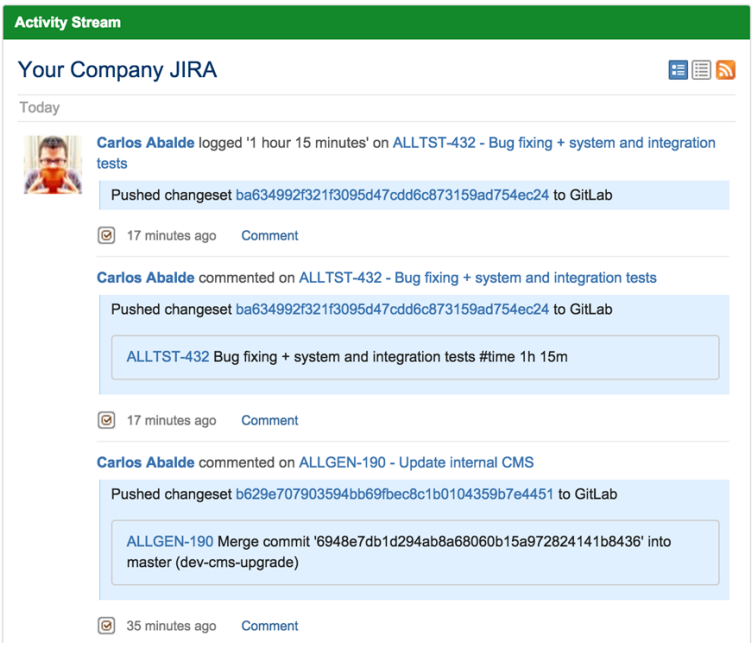

Figure 5.17: Screenshot of how GitLab's messages are seen in Jira's interface[34]

Although in the first planning this was a longer period of time, this last phase was shortened. This was due because of the many internal discussions that took place for the software to be approved and for the various managers of the departments to understand how to introduce it to their subordinates.
After the last meeting, described in the previous paragraph, new users were created for the managers that participated and the credentials were given so that they could see the interface and interact with the mock projects that remained there.

The tutor had credentials as well and he was very enthusiast of how the demo performed and the potentials of the software, although this had to be well understood by the other managers and members of the company as well.
Although the last snapshot of the VM that was made contained a working infrastructure ready to be moved in a production environment, the IT department opted to wait until they understood how to properly secure the server that was going to host it. The most important part would be creating a secure VPN TUNNEL$_G$ that would allow employees that are working from other locations outside the company to connect as if they would be inside the building. This method must ensure thought that there would be no possibility for any other to get, or even hack, inside the company's network and access their products' source code. Besides this, the other managers where glad to hear that the tools have been approved and where happy about the demonstration.
Because of this restrictions from the IT department, some of the requirements could not be fulfilled: *Do1*, *Do4*, *Fo1*, *Fo2*. These requirements could not be fulfilled because of the restrictions imposed by the IT department for interns.
It would have required me to access their production server and having access to the data that is contained in the Redmine instance, which is available only for the employees. Or in case of *Do1* I could not fulfill the requirement because I would have needed access to the administrative passwords of Office 365.
Requirement *Do2* was not fulfilled because while going on with the project, the tutor noted that there was no need for custom scripts.

5.5 How are these tools being used

Even if Athonet does go for a pure Agile development model, the tools it has chosen to use still help to improve the organization of their work. In fact, the tools fits their current needs, which reflect an hybrid model between Scrum and Kanban: "*Scrumban*". As in Kanban it helps visualizing workflow, limiting work in process (WIP), and measuring productivity are prioritized. Scrumban removes the practice of limiting WIP by time (i.e. velocity and Sprint boundary) and replaces it by limiting WIP for each stage. It's based on having a continuous flow of work, which is what Athonet has for their products.
In Jira, this methodology is called "*Kanplan*"[46], and it is well documented on the "*Agile Coach*" section on their website.

6 Conclusions

After having explained how this solution has been implemented, I would like to draw the conclusions on the project, internship experience and knowledge acquired.

6.1 Objectives achievement

Even thought the original objectives have changes during the internship, the software's documentation and the blogs presented contained the correct answers in order to correctly configure the software.
Here is a list of the requirements and their status at the end of the project:

- *O01*: ACHIEVED
- *O02*: ACHIEVED
- *O03*: ACHIEVED
- *O04*: ACHIEVED
- *O05*: ACHIEVED
- *D01*: NOT IMPLEMENTED
- *D02*: NOT IMPLEMENTED
- *D03*: ACHIEVED

- *D04*: NOT IMPLEMENTED
- *D05*: ACHIEVED
- *D06*: ACHIEVED
- *F01*: NOT IMPLEMENTED
- *F02*: NOT IMPLEMENTED

Note that the ones with "*NOT IMPLEMENTED*" status is because of internal rules that would not allow interns to interact with the production environment.
Although these have not been achieved, the knowledge on how to implement them has been acquired since their description is contained in the documentation that has been left in Athonet.

6.2 Improvement and future implementations

As many other projects there are many improvements that can applied. To complete the project presented in this thesis I would move it into a production environment and migrate the data from the Redmine instance. To have a better and more stable environment thought, there are various improvements that could be made. The first and most important one, in my personal opinion, would be separating the database from the Virtual Machine that hosts the software. This would bring advantages like:

- the machine with Jira and Confluence would be less stressed since it would not have so many reads and writes per disk
- the database could be stores in a single machine (or multiple ones) with dedicated disks arranged in a redundant way which adds a layer of security, in a disaster recovery plan, over regular backups
- maintenance to either one of the machines would be easier and would impact less the company's work
- etc.

Another important improvement would be separating the Confluence instance from the Jira one and, instead of having them running on a dedicated Virtual Machine each, creating a Docker [47] container$_G$ so that they would be easier to handle (copy, migrate, update).

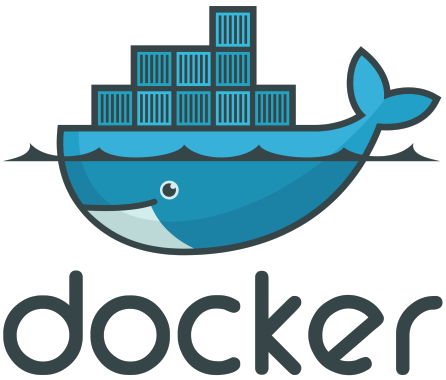

Figure 6.1: Docker logo

The use of containers would also allow a better usage of the host machine's resources like memory and CPU power by sharing them. These would otherwise be split and left unused by the VMs. This kind of improvements should be implemented when planning the migration of the services in a production environment.
Other future improvements can be installing plugins that would allow connectivity with the Microsoft Office suite allowing employees to produce documents and reports without learning how to do it in Confluence and just uploading them.

6.3 What I have learned

During this internship I had the possibility to better understand the hierarchy of a company, an argument well introduced in the Software Engineering (SWE) course. In Athonet I had the possibility to see this first hand and to interact with people on more levels of responsibility, from the CTO, to the product owner to the managers of verification and development teams.
About the tools, it is important to know how to use software like this not only for tracking the status of issues but on how to see other information about them as well. This kind of tools are are found in many IT companies, small or big, since they ease not only the work of keeping track of issues but the documentation of meeting notes, internal documents, release changelogs and many other features as well.
In a growing company like Athonet it is useful to set some boundaries, not only for lower level employees like developers, but for the management as well.

A

Appendix A: Time division

This Appendix contains two types of time division that represent the content of the "*Piano di Lavoro*": a table with the hours spent per task and a Gantt chart. For each of these there are two iterations, the "*estimated*" one and the "*final*" one, calculated before and after the internship.

A.1 Estimated time spent per task

Hours spent	Task
70	Learning the technologies
20	Requirements analysis
40	Setting up the environment
50	Configuration of the services
60	Integration
100	Test
20	Writing documentation
40	Migration
10	Acceptance test
Total number of hours	410

Table A.1: Estimated time spent per task table

A.2 Final time spent per task

Hours spent	Task
75	Learning the technologies
25	Requirement analysis
41	Setting up the environment
50	Configuration of the services
62	Integration
120	Test
22	Writing documentation
15	Acceptance test
Total number of hours	410

Table A.2: Final time spent per task table

A.3 Estimated Gantt Chart

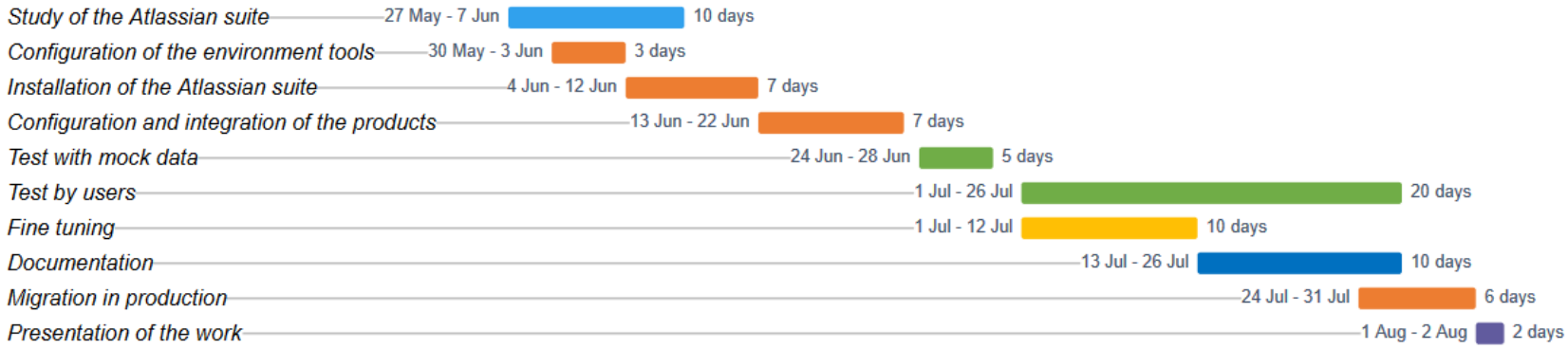

Figure A.1: Estimated Gantt Chart contained in the "*Piano di Lavoro*" document

A.4 Final Gantt Chart

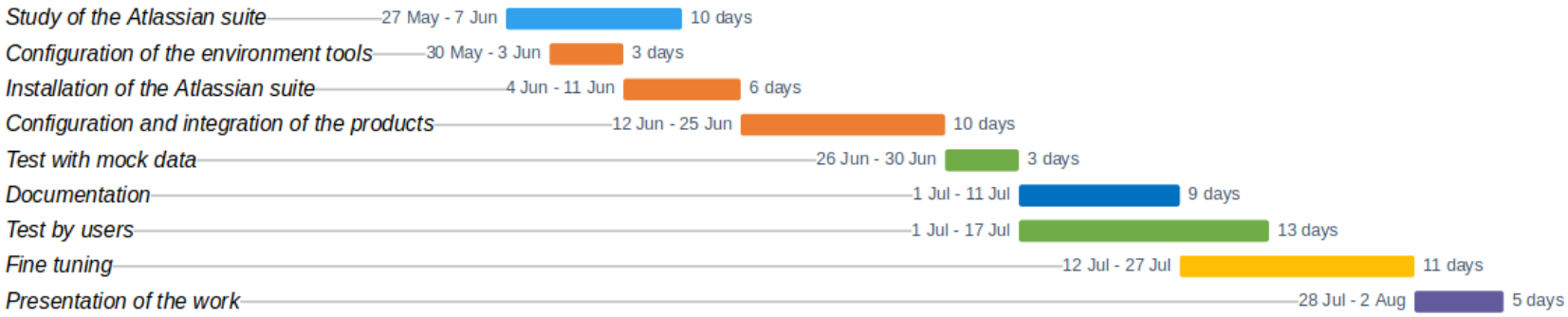

Figure A.2: Final Gantt Chart

B

Appendix B: Project requirements

B.1 Notation

Requirements in this text are referenced as:

- *O* ("*Obbligatorio*" in italian) for Mandatory: that have to be implemented, represent the core of the project
- *D* (Desiderabile in italian) for Desirable: not mandatory for the final objective but add greater value
- *F* (Facoltativi in italian) for Optional: add value to the project but not as much as the previous ones, carried out only if there is time left for them

Each requirement has a unique ID that is composed by the first letter of their importance (from the Italian word to resemble the "*Piano di Lavoro*") and an increasing number.

B.2 Original project requirements

The requirements that have been agreed with the tutor and that are included in the Piano di Lavoro are:

- Mandatory
  - *O01*: all the new software must be correctly configured
  - *O02*: Jira and Confluence must be fully interconnected
  - *O03*: it is required a clear and suitable documentation for both the users and the maintainers

- Desirable
  - *D01*: integration with Microsoft Office
  - *D02*: create scripts that will interact with the software that compose the environment

- Optional
  - *F01*: migrate existing data from the old system to the new
  - *F02*: migrate the system to the production environment to replace (even gradually) the existing one

B.3 Revised project requirements

Here is a list of the requirements that have been revised during the internship period:

- Mandatory
  - *O01*: all the new software must be correctly configured
  - *O02*: Jira and Confluence must be fully interconnected
  - *O03*: it is required a clear and suitable documentation for both the users and the maintainers
  - *O04*: connect with the repository hosted in GitLab
  - *O05*: there must be a live product roadmap that can be available for various members of the company

- Desirable
  - *D01*: integration with Microsoft Office
  - *D02*: create scripts that will interact with the software that compose the environment
  - *D03*: customize the interface with the logo and the colors of the company
  - *D04*: let client connect only via HTTPS protocol
  - *D05*: have a customizable interface for inserting issues
  - *D06*: create various security groups with different privileges

- Optional
  - *F01*: migrate existing data from the old system to the new
  - *F02*: migrate the system to the production environment to replace (even gradually) the existing one

References

[1] *Dilbert official website*.
https://dilbert.com/

[2] *Athonet official website*.
https://athonet.com/

[3] *Ericsson official website*.
https://www.ericsson.com/en

[4] *Gianluca, inventore di reti low cost ora entra alla corte di Facebook*.
https://www.ilgazzettino.it/vicenza-bassano/bassano/telecom-1936961.html

[5] *Il genio torna dalla Svezia per soffrire nella sua Italia*.
http://www.ilgiornale.it/news/interni/genio-torna-svezia-soffrire-nella-sua-italia-912407.html

[6] *Athonet, la startup italiana che trionfa al Mobile World Congress*.
https://www.millionaire.it/athonet-la-startup-italiana-che-trionfa-al-mobile-world-congress/

[7] *Redmine official website*.
https://www.redmine.org/

[8] *Microsoft Sharepoint official website*.
https://products.office.com/en-us/sharepoint/collaboration

[9] *Microsoft Planner official website*.
https://products.office.com/en-us/business/task-management-software

[10] *GitLab official website*.
https://about.gitlab.com/

[11] *Software Life Cycle*.
https://ase.in.tum.de/paid.globalse.org/paid1/courseDocs/Readings/SoftwareLifeCycle030398.pdf

[12] *ISO/IEC TR 24774:2010*.
https://www.iso.org/standard/53815.html

[13] *SDLC - Agile Model*.
https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm

[14] *To agility and beyond: The history—and legacy—of agile development*.
https://techbeacon.com/app-dev-testing/agility-beyond-history-legacy-agile-development

[15] *Agile & Development: Agile Manifesto*.
https://www.productplan.com/glossary/agile-manifesto/

[16] *Manifesto for Agile Software Development*.
https://agilemanifesto.org/

[17] *4 Values of the Agile Manifesto and 12 Agile Principles Easily Explained*.
https://medium.com/hygger-io/4-values-of-the-agile-manifesto-and-12-agile-principles-easily-explained-84cd429f69f

[18] *State of Agile Report*.
https://explore.versionone.com/state-of-agile

[19] *Exploring Key Elements of Spotify's Agile Scaling Model*.
https://medium.com/@media_75624/exploring-key-elements-of-spotifys-agile-scaling-model-471d2a23d7ea

[20] *Spotify engineering culture (part 1)*.
https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/

[21] *Leadership Principles*.
https://www.amazon.jobs/en/principles

[22] *How Amazon Became Agile*.
https://www.forbes.com/sites/stevedenning/2019/06/02/how-amazon-became-agile/

[23] *Roles on Agile Teams: From Small to Large Teams.*
http://www.ambysoft.com/essays/agileRoles.html

[24] *Epics, Stories, Themes, and Initiatives.*
https://www.atlassian.com/agile/project-management/epics-stories-themes

[25] *The Massive Downside of Agile Software Development.*
https://www.inc.com/adam-fridman/the-massive-downside-of-agile-software-development.html

[26] *18 Best JIRA Alternatives For Agile Project Management in 2019.*
https://www.workzone.com/blog/jira-alternatives/

[27] *Atlassian releases new wiki: Confluence 1.0.*
https://www.theserverside.com/discussions/thread/24701.html

[28] *Atlassian Marketplace.*
https://marketplace.atlassian.com/

[29] *Say hello to JIRA and JIRA Service Desk 3.*
https://confluence.atlassian.com/migration/jira-7/server_jira+jsd_product-changes#Server_JIRA+JSD_Productchanges-Uniquefeatures=applicationsforyourteams

[30] *What are the differences between JIRA Software, JIRA Service Desk and JIRA Core.*
http://www.akeles.com/what-are-the-differences-between-jira-software-jira-service-desk-and-jira-core/

[31] *Portfolio for Jira.*
https://marketplace.atlassian.com/apps/1212136/portfolio-for-jira

[32] *Akraino Wiki.*
https://wiki.akraino.org/

[33] *A brief overview of Jira.*
https://www.atlassian.com/software/jira/guides/getting-started/overview#key-terms-to-know

[34] *GitLab Listener*.
https://marketplace.atlassian.com/apps/1212989/gitlab-listener

[35] *Compare Atlassian cloud vs server*.
https://confluence.atlassian.com/cloud/compare-atlassian-cloud-vs-server-744721664.html

[36] *JIRA Documentation*.
https://confluence.atlassian.com/jira/

[37] *Installing Jira applications on Linux*.
https://confluence.atlassian.com/adminjiraserver/installing-jira-applications-on-linux-938846841.html

[38] *Accessing JIRA's H2 embedded database*.
https://confluence.atlassian.com/jirakb/accessing-jira-s-h2-embedded-database-776818136.html

[39] *User is constantly logged out of JIRA*.
https://confluence.atlassian.com/jirakb/user-is-constantly-logged-out-of-jira-192872663.html

[40] *Configure Tomcat so that it has a unique SESSION_COOKIE_NAME to prevent session overwriting for JIRA*.
https://jira.atlassian.com/browse/JRASERVER-36960

[41] *How to change the JIRA application context path*.
https://confluence.atlassian.com/jirakb/how-to-change-the-jira-application-context-path-225119408.html

[42] *BitBucket official website*.
https://bitbucket.org/

[43] *GitLab Jira integration*.
https://docs.gitlab.com/ee/user/project/integrations/jira.html

[44] *GitLab Jira development panel integration*.
https://docs.gitlab.com/ee/integration/jira_development_panel.html

[45] *What's the difference between a product and a project?*
https://harbott.com/whats-the-difference-between-a-product-and-a-project-aaacfc983340

[46] *Kanplan: where your backlog meets kanban.*
https://www.atlassian.com/agile/kanban/kanplan

[47] *Docker official website.*
https://www.docker.com/

[48] *Definition of Broadband.*
https://searchnetworking.techtarget.com/definition/broadband

[49] *What is a Docker Container?*
https://www.sdxcentral.com/containers/definitions/what-is-docker-container/

[50] *Configuring Contexts.*
https://www.eclipse.org/jetty/documentation/9.4.x/configuring-contexts.html

[51] *Definition of Cookie.*
https://www.techopedia.com/definition/7624/cookie

[52] *What is Low Latency?*
https://www.informatica.com/services-and-training/glossary-of-terms/low-latency-definition.html

Glossary

BROADBAND NETWORKING

In general, broadband refers to telecommunication in which a wide band of frequencies is available to transmit information. Because a wide band of frequencies is available, information can be multiplexed and sent on many different frequencies or channels within the band concurrently, allowing more information to be transmitted in a given amount of time (much as more lanes on a highway allow more cars to travel on it at the same time) [48]. 2

CENTOS

CentOS is an Enterprise Linux distribution based on the sources from Red Hat Enterprise Linux. 35

CONTAINER

A Docker container is an open source software development platform. Its main benefit is to package applications in containers, allowing them to be portable to any system running a Linux or Windows operating system (OS)[49]. 52

CONTEXT PATH

The context path is the prefix of a URL path that is used to select the context(s) to which an incoming request is passed[50]. 39

COOKIE

A cookie is a text file that a Web browser stores on a user's machine. Cookies are a way for Web applications to maintain application state. They are used by websites for authentication, storing website information/preferences, other browsing information and anything else that can help the Web browser while accessing Web servers[51]. 39

CORE NETWORK

The core network of a telecommunication infrastructure represents the services that interconnect the main components that create the backbone, which are usually routers and switches. 2

FRAMEWORK

A framework is an abstraction that establishes common practives for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application. 11

H2 DATABASE

H2 is an open-source lightweight Java database. It can be embedded in Java applications or run in the client-server mode. 36

LONG-TERM EVOLUTION (LTE)

Long-Term Evolution is a standard for wireless broadband communication for mobile devices and data terminals. 2

LOW LATENCY COMMUNICATION

Low latency describes a computer network optimized to process a very high volume of data messages with minimal delay (latency). These networks are designed to support operations that require near real-time access to rapidly changing data [52]. 2

MILESTONE

A milestone is a specific point in time within a project lifecycle used to measure the progress of a project toward its ultimate goal. Milestones are associated with working deliverables that, in case of future flops, can be used as checkpoints. 9

REMMINA

Remmina is a is free and open-source remote desktop client for Linux and other Unix-like systems that allows connecting with other systems from terminal or graphical interfaces. 35

REPOSITORY

A software repository is a storage location from which software packages, verisoned and well documented, may be retrieved and installed on a computer. 8

ROADMAP

A roadmap is a strategic plan that defines a goal or desired outcome and includes the major steps or milestones needed to reach it. 7

VIRTUAL MACHINE

A Virtual Machine (VM) is an emulation of a computer system. 35

VIRTUAL PRIVATE NETWORK (VPN)

A Virtual Private Network extends a private network across a public network, and enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network. 49