Isabel Filipa Carrusca Lopes de Sousa

Licenciada em Ciências de Engenharia e Gestão Industrial

Agile Project Management Model for Information Technology Projects: Glintt Case Study

Dissertação para Obtenção do Grau de Mestre em Mestrado Integrado em Engenharia e Gestão Industrial

Júri:

Presidente: Doutora Susana Carla Vieira Lino Medina Duarte, Professora Auxiliar, FCT-UNL

Vogal: Doutora Alexandra Maria Baptista Ramos Tenera, Professora Auxiliar, FCT-UNL

Orientador: Doutor António Carlos Bárbara Grilo, Professor Associado com Agregação, FCT-UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2018

Isabel Filipa Carrusca Lopes de Sousa

Licenciada em Ciências de Engenharia e Gestão Industrial

Agile Project Management Model for Information Technology Projects: Glintt Case Study

Dissertação para Obtenção do Grau de Mestre em Mestrado Integrado em Engenharia e Gestão Industrial

Júri:

Presidente: Doutora Susana Carla Vieira Lino Medina Duarte, Professora Auxiliar, FCT-UNL

Vogal: Doutora Alexandra Maria Baptista Ramos Tenera, Professora Auxiliar, FCT-UNL

Orientador: Doutor António Carlos Bárbara Grilo, Professor Associado com Agregação, FCT-UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2018

Agile Project Management Model for Information Technology Projects: Glintt Case Study

Acknowledgements

I would like to thank my coordinator Professor Doctor António Grilo for guidance, support and availability that led to this dissertation's success.

To Faculdade de Ciências e Tecnologia (FCT) where I learnt and grew so much, leaving many good memories as it turned out to be my second home during my student life.

To Glintt for accepting to be my dissertation's case study, a special thanks to Mécio Afonseca who accompanied me during this journey and taught me so much professionally.

To Daniel who patiently stood by my side during this work, gave me day-by-day motivation and inspiration, definitely a very important contributor in the achievement of this goal.

To my friends and colleagues for always cheering up my days with big smiles and laughs, keeping me in a positive mood during this journey.

Lastly but most importantly, to my mother Margarida and my father Luís who invested so much in my studies and always believed in me with great love. I hope to let you proud.

Abstract

Information Technology (IT) has been one of the major catalysts in improving how companies work, whether it is by automating tasks or by allowing communication to be more effective and efficient. Due to the capabilities of IT, companies have increasingly invested in this sector, which then raises its market competitiveness.

IT companies that work on multiple software projects need to control time, costs, and quality effectively. To do so, it is important to have suitable and well-structured project management processes. Having processes mapped is also important to guarantee the certification on quality standards, which helps companies to remain competitive within the market they operate.

Classical project management approaches, such as Waterfall, have struggled to cope with specific challenges in the IT sector such as fast-paced requirement changes, ambiguous functionality descriptions, and the fact that technology is constantly improving – Agile Project management arose to tackle such challenges.

In order to guarantee a successful implementation of Agile, it is important to pick the right methods and practices according to the company's characteristics and needs.

This dissertation reviews some of the most popular Agile methods in the literature and suggests a tailor-made project management process for a Portuguese IT company operating in the health sector, Glintt. In this case study, the company's current project management process was reviewed and a new Scrum-based process was suggested to fit the company's requirements accordingly. The suggested process also uses elements of Extreme Programming, Lean, and Kanban.

Keywords: Agile, Scrum, Extreme Programming, Lean, Kanban, IT, Project Management, Software Development, Process Modelling

Resumo

Tecnologias de Informação (TI) têm sido o maior catalisador na forma como as empresas funcionam, seja pelo automatismo de tarefas ou por permitir uma comunicação mais eficaz e eficiente. Devido ao potencial das TI, o investimento neste setor tem vindo a aumentar significativamente, o que leva a um aumento de competitividade neste mercado.

Empresas de TI que têm vários projetos de software a decorrer em simultâneo precisam de controlar tempo, custos e qualidade de forma eficaz. Para tal, é importante ter processos de gestão de projetos bem estruturados e adaptados às suas características. O mapeamento de processos também é relevante para garantir a certificação em normas de qualidade que por sua vez faz com que as empresas se mantenham competitivas dentro do mercado em que operam.

As abordagens clássicas de gestão de projetos, tal como Waterfall, não se têm mostrado eficazes perante os desafios do setor de IT tais como: rápidas alterações de requisitos, descrições ambíguas de funcionalidades e a constante evolução das tecnologias – a gestão de projetos Agile surge como forma de dar resposta a estes desafios.

Por forma a garantir o sucesso da implementação de processos Agile, é importante a escolha dos métodos e práticas mais apropriados às características e necessidades da empresa.

A presente dissertação revê alguns dos métodos Agile mais populares na literatura e sugere um processo de gestão de projetos feito à medida para uma empresa de TI portuguesa presente no setor da saúde, Glintt. Neste caso de estudo, o atual processo de gestão de projetos é revisto e um novo processo baseado em Scrum é proposto para se adaptar às necessidades da empresa. O processo aqui sugerido também utiliza elementos de Extreme Programming, Lean e Kanban.

Palavras-chave: Agile, Scrum, Extreme Programming, Lean, Kanban, TI, Gestão de Projetos, Desenvolvimento de Software, Modelação de Processos

Contents

List of Tables

List of Figures

List of Acronyms

ALN – Agile Leadership Network

ASD – Adaptive Software Development

CRM – Customer Relationship Management

DOI – Declaration of Interdependence

DSDM – Dynamic Systems Development Method

FDD – Feature Driven Development

GDP – Gross Domestic Product

IIDD – Iterative and Incremental Design and Development

IT – Information Technology

ITC – Information Technology Consulting

RAD – Rapid Application Development

ROI – Return on Investment

RUP – Rational Unified Process

XP – Extreme Programming

1 Introduction

1.1 Research context

Information Technology (IT) is "the study or use of systems – especially computers and telecommunications – for storing, retrieving, and sending information" (Oxford English Dictionary, 2015). Nowadays, IT surrounds us in everything we do, since we live in an era where our lives are directly related to technology. According to ITU's report on ICT Facts & Figures, between 2000 and 2015 global internet penetration grew from 6,5% to 43%, which corresponds to 3,2 billion people using the internet globally by the end of 2015. Moreover, in the same time frame, there are more than 7 billion mobile cellular subscriptions, corresponding to a penetration rate of 97% since 2000 (International Telecomunication Union, 2015). Since the past couple decades, we have clearly been facing a rising demand for technology and there is no evidence to expect this trend to slow down in the near future.

In order to succeed in the competitive and increasingly complex economy, it is important for companies to have their internal processes rightly structured an aligned with their strategic goals. The 200 largest Portuguese companies within the IT sector generated a 4.9 billion euros turnover in 2013 (ranking das 200 melhores empresas em IT, Jornal de Negócios, 2014), roughly 2.9% of the portuguese GDP in this year (Pordata, 2017).

Services are more and more related to technology, creating large information loads. For instance, simply by shopping or buying online, information resultant from transactions is stored. Also, by going to a hospital, medical data is stored and can instantly be accessed and updated by health professionals. These are just some examples of the IT dimension nowadays. Findings show that companies are willing to invest in Information Technology since it is appointed as one of the main drivers to unlock growth and profits (Rigby & Bilodeau, 2015).

Information Technology (IT) projects can usually have fast-paced requirement changes, ambiguous functionality descriptions, and the used technologies are constantly improving. Successfully managing this type of projects can then be challenging for companies. Although there is already an extensively documented project management methodology in Project Management Book of Knowledge Guide (Project Management Institute, 2013), its implementation in IT projects still leads to failure in many cases (The Standish Group, 2013). This is due to a blind, unsuitable application of project management principles in a domain of IT projects, in which concepts like unchanging plans and low execution risk are invalid. Logically, different types of projects require different methods of management. Some

projects, specially knowledge worker projects occurring in fast-moving constrained environments, call for an agile approach (Griffiths, 2012).

The cornerstone of Agile originated long before de Word Wide Web, when engineers adopted iterative and incremental design and development (IIDD) as a method (Glazer, Dalton, Anderson, Konrad, & Shrum, 2008).

Agile methods have been widely used, especially since its Manifesto in 2001, mostly in software development projects. These methods contrast with traditional project management approaches, such as waterfall, by emphasizing teamwork, frequent deliveries of working software, continuous design, keeping a flexible scope, embracing uncertainty and customer interaction, and a modified Project Team organization. Agile is iterative and incremental, seeking to avoid the standard approaches that emphasize early design and specification freeze, a fixed project scope, and low customer interaction (Serrador & Pinto, 2015).

A recent survey of development and IT professionals, conducted in 2015 by Hewlett Packard (HP), shows that "pure agile" organizations are on the rise, with 16% "pure agile" organizations and 51% "leaning towards agile". According to the 10th State of Agile in 2016, most companies are using Agile methodology to manage their projects in the IT sector. The same report also shows that 87% of respondents said implementing agile improved their ability to manage changing priorities. It states that in most cases it improved team productivity and project visibility. The main results are on-time delivery, product quality, and costumer/user satisfaction (Version One, 2016).

With the actual unstable economy (United Nations, 2015), in order to stay competitive, companies developing software could use an agile process (Cohn, 2010) that can reduce drastically time to market and improve services' and products' quality, competitive prices. To achieve this, it is essential to successfully implement project management practices that are suitable for each project.

1.2 Motivation and Scope

This dissertation's first goal is to review the suitability of agile methodologies for Glintt's Information Technology Consulting department. The second is to develop a proposal for a new agile project management method after reviewing the company's current processes. Glintt aims to obtain CMMI Level 3 accreditation for Software Development in Java, Microsoft and Outsystems. In order to do so, an assessment was undertaken by a consultant and it identified an agile process that was used by some teams but that not described in the organization. Therefore, it is necessary to define it according to the organization's needs.

Therefore, it aims to answer the following research questions:

1- What are the most referenced Agile methods and features for IT Consultancy companies?

2- How to adapt IT Consultancy project management processes to become more agile?

1.3 Research Methodology

The study starts by revising some of the most popular Agile methods' characteristics and reviews implementations of these methods in the IT sector. An extensive review of the main agile methodologies is done in order to systematize the main generic characteristics of these methodologies.

Considering the takeaways from the literature, the research methodology considers the Case Study research strategy, where an IT Consultancy company – Glint – is analysed and an Agile project management process is then developed and mapped accordingly.

1.4 Dissertation Structure

This dissertation is divided in five chapters. The first and present one shows the importance of the adopting and mapping an Agile project management process in the context of an IT company.

The second, reviews agile project management methods with emphasis on Scrum, Extreme Programming and Feature Driven Development. This chapter also reviews the implementation of such methods in the IT sector and sums key takeaways from these studies.

The third chapter presents a case study starting with the company's characteristics and an analysis on its current processes.

The fourth chapter includes the suggestion of an Agile model for the case study's company, starting with elements definition, followed by the Agile process.

The final chapter presents the study's conclusions, its limitations, its contributions for the company and literature, and provides guidelines for future work.

2 Review of Agile Methods

2.1 Agile Project Management

Literature on Agile methods' history states that it has originated from Iterative and Incremental Design and Development (IIDD), a method adopted by engineers over eighty-four years ago. Modern Agile methods emerged throughout the decade of 1990, when IIDD started to be widely implemented in the software community, including rapid prototyping, Rapid Application Development (RAD) and Rational Unified Process (RUP) (Glazer et al., 2008).

After the Industrial Revolution, the Information Revolution emerged. This last revolution is focused on information and collaboration, rather than manufacturing, turning knowledge (and the ability to use it) a more relevant asset than actual products and services (Griffiths, 2012).

Mike Griffiths, co-creator of Dynamic Systems Development Method (DSDM), uses the term "knowledge worker projects" to distinguish information-based projects from other types of projects. Different types of projects need different types of management and, as Griffiths states, "knowledge worker projects, occurring in fast-moving or time-constrained environments, call for an agile approach" (Griffiths, 2012).

Different definitions for agile methods can be found in the literature, as it varies in practice. Some researchers define Agile as a philosophy (Abbas, Gravell, & Wills, 2008). However, in a more practical point of view, Barry Bohem describes it as "In general, agile methods are lightweight processes that employ short iterative cycles, actively involve users to establish, prioritize, and verify requirements, and rely on a team's tacit knowledge as opposed to documentation. A truly agile method must be iterative (take several cycles to complete), incremental (not deliver the entire product at once), self-organizing (teams determine the best way to handle work), and emergent (processes, principles, and work structures are recognized during the project rather than predetermined)" (Boehm & Turner, 2005, p. 32).

On the other hand, Alistair Cockburn (the creator of Crystal methodologies) states that "Agile implies being effective and manoeuvrable. An Agile process is both light and sufficient. The lightness is a means of staying manoeuvrable. The sufficiency is a matter of staying in the game." (Jim Highsmith & Cockburn, 2001, p. 120). This definition has a more business-oriented view, meaning Agile methods can help to cope with business competitiveness.

Agile had a turning point back in 2001 when seventeen software and methodology experts gathered to reach a common ground concerning software development methodologies and, in the end, agreed to

"The Agile Manifesto" (Jim Highsmith, 2001). This group of specialists needed an alternative to the existing traditional approaches (such as waterfall) to better manage software projects. The Agile Manifesto gathers four values and twelve principles, which are currently shared by most Agile methods, and reads as follows (Beck et al., 2001):

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

1. Individuals and interactions over processes and tools;
2. Working software over comprehensive documentation;
3. Customer collaboration over contract negotiation;
4. Responding to change over following a plan;

That is, while there is value in the items on the right, we value the items on the left more.

We follow these principles:

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.“

This manifesto was initially considered extreme by many traditionalists in project management, but the agile community claims that it is not meant to be anti-methodology, in fact, it is supposed to restore credibility to the word methodology (Beck et al., 2001).

Another important milestone in Agile Project Management history was the Declaration of Interdependence (DOI), created in 2015 by the cofounders of Agile Leadership Network (ALN). This declaration serves as guiding principles to implement Agile and adaptive approaches and is focused on the project management side of agile projects. The DOI reads as follows (D. Anderson et al., 2005):

“Declaration of Interdependence

Agile and adaptive approaches for linking people, projects and value

We are a community of project leaders that are highly successful at delivering results. To achieve these results:

- We increase return on investment by making continuous flow of value our focus.
- We deliver reliable results by engaging customers in frequent interactions and shared ownership.
- We expect uncertainty and manage for it through iterations, anticipation, and adaptation.
- We unleash creativity and innovation by recognizing that individuals are the ultimate source of value, and creating an environment where they can make a difference.
- We boost performance through group accountability for results and shared responsibility for team effectiveness.
- We improve effectiveness and reliability through situationally specific strategies, processes and practices.”

2.2 Agile Methods

There are around twelve actively used agile methods (Griffiths, 2012). Despite the number of different Agile methods, most have in common the same principles presented in chapter 2.1. However, from an implementation point of view, each has its own set of practices, terminology, and strategies.

Some methods found in literature, and probably the most cited ones, are:

- Scrum (Takeuchi & Nonaka, 1986; Ken Schwaber, 1995; Ken Schwaber & Beedle, 2001)
- Extreme Programming (XP) (Beck, 1999a)
- Feature Driven Development (FDD) (Palmer & Felsing, 2002)
- Dynamic Systems Development Method (DSDM) (Stapleton, 1997)
- Crystal family of methodologies (Cockburn, 2002)
- Rational Unified Process (RUP) (Kruchten, 2000)
- Adaptive Software Development (ASD) (James Highsmith, 2000)
- Open Source Software Development (O'Reilly, 1999)
- Other related methods are Lean development and Kanban development.

In order to instantiate an agile methodology to the case study's company, it is necessary to review the methods above cited. This chapter aims to identify, describe, and review existing methods; while selecting potential practices and concepts that can be adopted inside a software development context.

2.2.1 Scrum

Looking at the origin of the term Scrum, it was used in Rugby as a strategy to restart play after a rule infringement. The players of each team would try to gain possession of the ball by pushing forward against the opposite team, packing together with heads down and arms interlocked (Oxford English Dictionary, 2015).

As referred by Abrahamsson et al. (2002), one of the first references in the literature to the term "Scrum" was found in an article written by Takeuchi and Nonaka (1986) where a new approach to product development process was proposed. The authors state that speed and flexibility are essential to cope with the fast-paced and competitive world of commercial new product development. The sport of rugby is used as a metaphor in that article: "as in rugby, the ball gets passed within the team as it moves as a

unit up the field". As a result of interviews with multi-level organization members from leading companies in the new product development sector, the authors gathered six successful management characteristics that, if used as a whole, create a strong set of dynamics that will make a difference in "moving the scrum downfield" (*i.e.* succeeding by keeping a sustainable competitive advantage). These characteristics are: built-in instability, self-organizing Project Teams, overlapping development phases, "multi-learning", subtle control, organizational transfer of learning (Takeuchi & Nonaka, 1986).

Later on, in the early 1990s, Ken Schwabler and Jeff Sutherland developed Scrum as a software development process to help organizations managing complicated development projects (Schwaber, 2004) . Scrum is now the most currently used agile method according to The Tenth Annual State of Agile Report by Version One in 2016. In this survey, nearly 70% of the respondents affirmed they practice Scrum (Version One, 2016).

Scrum relies on an iterative and incremental process (Schwaber, 2004). The main idea of Scrum is that software development includes several environmental and technical variables that are likely to change during the process. This makes the development process unpredictable, calling for flexibility in order to respond to the changes (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). Another important aspect about Scrum is that it depends majorly on progress and development rather than on planning. According to Schwaber (2004), the minimum plan necessary to start a project with Scrum consists of a vision and a Product Backlog – these terms will be detailed ahead. In this line of thought, Schwaber (2004) states that "Scrum controls the process of software development in order to guide work toward the most valuable outcome possible".

In order to describe the Scrum method, this chapter is divided into four sections: Roles, Practices, Artefacts and Process.

2.1.1.1 Roles

In Scrum there are three distinct roles: Scrum Master, Product Owner and Development Team. Each role has different tasks and responsibilities throughout the process, and all of them share management activities (Schwaber, 2004).

a) Scrum Master

The Scrum Master is responsible for ensuring that everyone involved in the project understands the Scrum method and implements it, as well as interacting with the Development Team and the Product Owner. Some might compare the Scrum Master to the Project Manager in a typical project (*i.e.* Waterfall), however, the Development Team in a Scrum project is self-managed. This role comes with the responsibility of removing impediments to progress, facilitating activities and providing coaching,

to make sure the team is working as productively as possible (Schwaber & Beedle, 2001). The Scrum Master ensures the process fits to the organization's culture and delivers the expected benefits (Schwaber, 2004).

b) Product Owner

The Product Owner is responsible for representing the interests of the stakeholders and maximizing the value of the product that will be delivered. This role has the responsibility for managing the Product Backlog (2.1.1.3 a)), which includes deciding what features are to be developed and frequently prioritizing them. Additionally, the Product Owner must always be available to the Development Team, ensuring they understand the requirements' specification, value and prioritization, but also to participate in effort estimating for development tasks. This way, the Product Owner is able to set Return on Investment (ROI) goals and release plans (Schwaber, 2004).

c) Development Team

The Development Team is the group of individuals who develop the product increments in each iteration, or, according to the Scrum terminology, Sprint (2.1.1.2 a)). In a Scrum project, the Development Team is empowered to manage its own work in order to deliver the required functionalities within each iteration. Its members are self-managing, self-organizing and cross-functional, being collectively responsible for the success of each iteration and of the project as a whole (Schwaber, 2004).

2.1.1.2 Practices

Practices in Scrum are management events that occur during different phases of the project to avoid chaos caused by unpredictability and complexity of projects (Schwaber, 1995).

The Scrum method includes the following practices: Sprint, Sprint Planning Meeting, Daily Scrum, Sprint Review and Sprint Retrospective (Schwaber, 2004).

a) Sprint

Sprint is a time-boxed event (*i.e.* iteration) of twoo to four consecutive weeks. During one Sprint, the Development Team builds a potentially shippable product increment by completing the tasks required to develop the Sprint Backlog items (2.1.1.3 b)). Each sprint is similar to a mini-project. It includes a Sprint Planning Meeting, Daily Scrums, the development work, a Sprint Review Meeting and a Sprint Retrospective. During the Sprint, no changes are made that would affect the Sprint goal. However, the scope may be clarified as new information becomes available. At the end of a Sprint, a completed increment of a potentially shippable product functionality is ready to be demonstrated to stakeholders, as it was developed, tested and documented (Schwaber, 2004).

b) Sprint Planning

The Sprint Planning meeting is also a time-boxed event and has the total duration of four hours (for a two week sprint) or eight hours (for a four week sprint), which is divided into two parts of four hours (Schwaber, 2004). Its goal is to determine what will be delivered in the end of that Sprint (first part) and how the work will be achieved (second part).

During the first part, the Product Owner presents the highest priority items in the Product Backlog (2.1.1.3 a)) to the Development Team. The Product Owner clarifies the Development Team members about content, purpose, meaning and intentions of each requirement, to reach shared understanding. Then, the Development Team selects a group of items which believes it can be turned into a completed increment of potentially shippable product functionality by the end of that Sprint (Schwaber, 2004). The team does this by estimating the effort required to develop those items, based on projected capacity and past performance (Abrahamsson et al., 2002).

During the second part of the Sprint Planning Meeting, the Development Team decides how the team will organize to deliver the sprint goal and creates a plan to complete it. This includes breaking the selected items into tasks that will compose the Sprint Backlog for that Sprint (2.1.1.3 b)) (Schwaber, 2004).

c) Daily Scrum

The Daily Scrum is a fifteen-minute timeboxed daily meeting and its purpose is to synchronize the work of all Team members through communication, and to identify and remove issues to improve its process. The Scrum Master ensures that these meetings happen and helps remove any identified obstacles. In order to assess progress, The Scrum Master asks each team member the following questions (Schwaber, 2004):

- What have you done on this project since the last Daily Scrum meeting?
- What do you plan on doing on this project until next Daily Scrum meeting?
- What impedes you of meeting your goals to this Sprint regarding this project?

Once a team member answers all three questions, the Scrum Master asks the same questions to the next team member and makes sure that during this time there will be no discussion or debate, so that all team members answer the questions within the defined time-box. If necessary, the Development Team members and the Scrum Master can discuss about the raised issues once the Daily Scrum is finished (Schwaber, 2004).

d) Sprint Review

The Sprint Review is a four-hour time-boxed meeting that is held at the end of the Sprint. In this meeting, the Development Team presents the Potentially Shippable Product Increment (2.1.1.3 c)) that was developed during the Sprint to the Product Owner and to any other stakeholders who are interested in attending. After the presentation, the audience may point out some issues or clarify questions, giving feedback to the Development Team and to the Product Owner. The attendees collaboratively discuss any additional features and/or changes to the Product Backlog, prioritizing them (Schwaber, 2004).

e) Sprint Retrospective

The Sprint Retrospective is an inspect-and-adapt activity that usually takes place after the Sprint Review and before the next Sprint Planning meeting. The difference between the Sprint Retrospective and the Sprint Review is that the Sprint Review is used to inspect and adapt the product, while the Sprint Retrospective is used to inspect and adapt the process (Rubin, 2012).

The Scrum Master gathers with the Development Team, and optionally with the Product Owner, and asks the following questions to each Development Team member (Schwaber, 2004):

- What went well during the last Sprint?
- What can be improved in the next Sprint?

As a facilitator, the Scrum Master writes in summary each answer and helps the team to search for ways to improve the Scrum process in the next Sprint (Schwaber, 2004). In the end, the attendants agree on a set of actions to be applied in the next sprint, thus keeping a continuous process improvement (Rubin, 2012).

2.1.1.3 Artefacts

Scrum introduces new artefacts that are used during the Scrum process (Schwaber, 2004). These artefacts are crucial for the project since they allow monitoring by providing information about its status on progress and future development. In this chapter, three Artefacts are presented: Product Backlog, Sprint Backlog and Potentially Shippable Product Increment.

a) Product Backlog

The Product Backlog is a prioritized list of all the requirements needed to complete the project based on current knowledge (Abrahamsson et al., 2002). It is available to all project members as the single source of requirements and includes features, functional and non-functional requirements, improvements and fixes (Schwaber, 2004).

The Product Backlog is dynamic, meaning it evolves throughout the project as new information becomes available. It is frequently prioritized and some items might be added, removed or revised by the Product Owner. This process, also is also known as Grooming, and takes into consideration stakeholder's and development team's inputs such as estimations and limitations. Product Backlog items can be prioritized using criteria such as value, cost, knowledge, and risk. Higher-priority items are more detailed and its estimations are more precise than the lower-ranked items. (Rubin, 2012).

b) Sprint Backlog

The Sprint Backlog is the set of items selected from the Product Backlog during the Sprint Planning Meeting to accomplish the sprint goal. This set of items is selected as the high-priority items that the team can realistically accomplish during the next sprint while working at a sustainable pace[1]. In order to do that, the team breaks down items into tasks for better effort estimation (Figure 2.1). The set of these tasks along with their associated product backlog form the Sprint Backlog (Rubin, 2012).

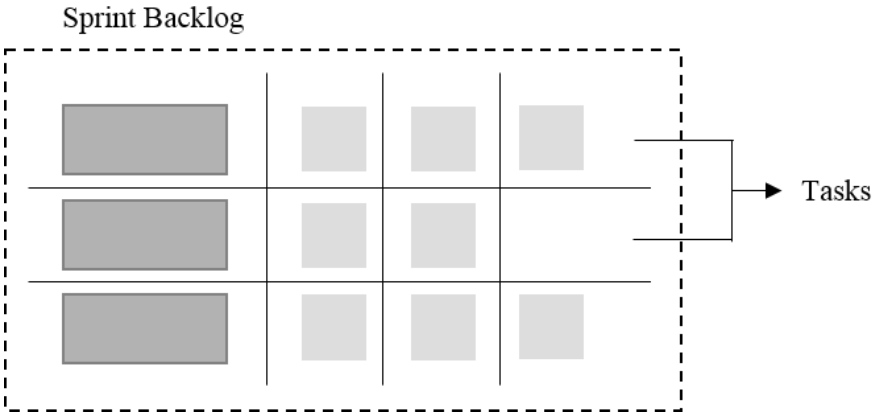


Figure 2.1 – Sprint Backlog (Rubin, 2012)

Effort estimation is crucial for the Product Backlog and for the Sprint Backlog because it measures size. Size is associated to cost, and the Product Owner needs to know an item's cost to properly prioritize it. Although Scrum doesn't indicate which measure to use, Rubin (2012) states that many teams measure item's size in Story Points or Ideal Days. These are both relative size measures, which means they measure size by comparing items (*i.e.* item A is four times larger than item B) (Rubin, 2012).

Measuring size enables the Product Owner to track the Development Team's Velocity in each Sprint, but also to monitor the project's progress with a Burndown Chart (Rubin, 2012).

[1] "A pace at which the development team can comfortably work for an extended period of time." (Rubin, 2012).

A Burndown Chart is a useful tool to visualize the amount of work remaining across time. This can be done using information from the Sprint Backlog and Product Backlog. The remaining work can be measured with Story Points, which are marked on the vertical axis. The time periods can be measured in days or Sprints, which are marked on the horizontal axis (Schwaber, 2004).

Figure 2.2 illustrates a Burndown Chart example. The blue line indicates the actual progress and, by computing a trend line, one can calculate the average velocity, thus allowing to forecast when will all the backlog items be finished using different scenarios: average velocity ends at sprint 9 (green line), high velocity ends at sprint 8 (orange line) and low velocity ends at sprint 10 (yellow line).

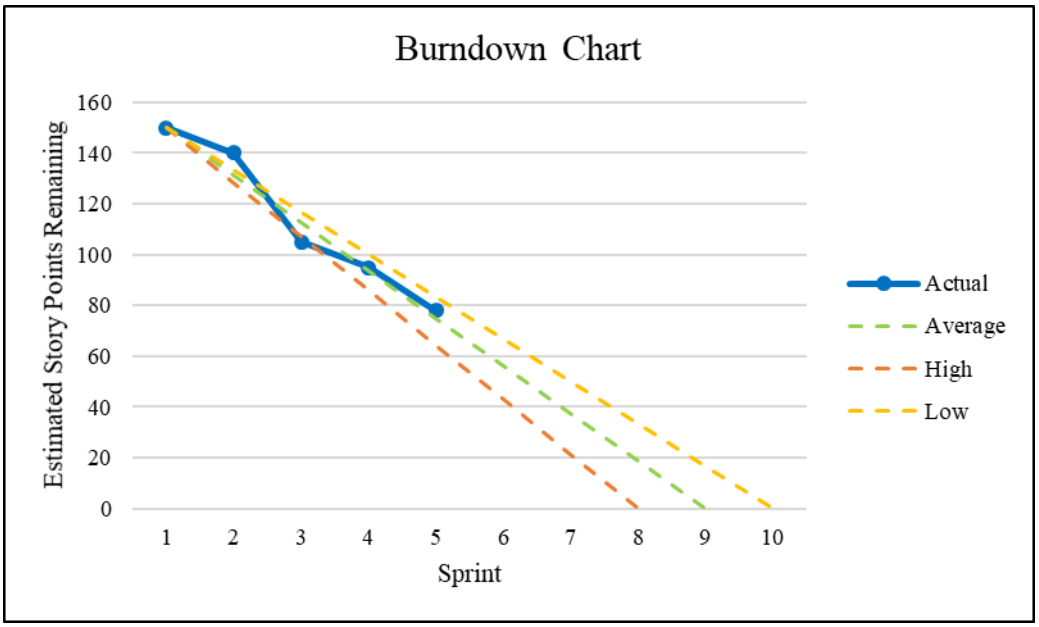


Figure 2.2 – Burndown Chart (Rubin, 2012)

c) Potentially Shippable Product Increment

A Potentially Shippable Product Increment is an increment that was developed during the sprint and conforms to the agreed definition of done (Schwaber, 2004). As defined by Ken Schwaber (2004), done means "complete as mutually agreed to by all parties and conforming to an organization's standards, conventions, and guidelines". In a more practical point of view, Kenneth Rubin (2012) defines done as "a slice of product functionality that is sufficiently functional and usable to generate feedback that enables the team to decide what work should be done next or how to do it". An example of a definition of done for software development can be a complete slice of product functionality that meets to the following acceptance criteria: designed, built, integrated, tested and documented (Rubin, 2012).

2.1.1.4 Process

Taking into consideration the gathered information in the literature, it is possible to draw a flowchart of the Scrum process, as illustrated in Figure 2.3.

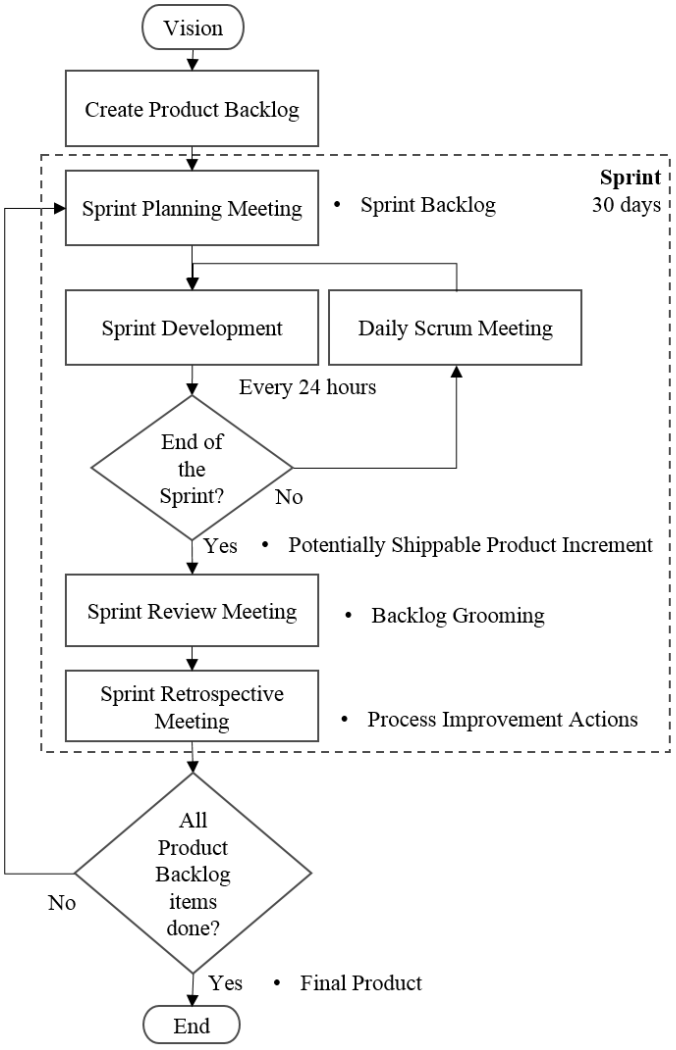


Figure 2.3 – Scrum Process

The above illustration summarizes the Scrum process and, for each activity, presents its artefacts as outputs. The process starts with a Vision, which is transformed into Product Backlog items which will lead to the Sprints. First, the Sprint Planning meeting takes place, where the Sprint Backlog is created for that Sprint (what will be developed and how). Then, the Sprint Backlog items are developed (Sprint Development) and every 24 hours the Daily Scrum is undertaken to synchronize work. At the end of the Sprint, the team presents the Sprint Backlog items that are considered done to the stakeholders and Product Owner in order to receive feedback and decide the next steps (Backlog Grooming). After this, the team discusses process improvement actions to apply in the next Sprint and, if there are still Product Backlog items to be developed, the process repeats until the final product is done.

The Scrum process provides guidelines that allow iterative development of product increments and constant customer involvement, thus facilitating work to deliver the right product at the end of the process.

2.2.2 Extreme Programming

Extreme Programming (XP) is an agile method created by Kent Beck in 1999 (Abrahamsson et al., 2002) which focuses on software development good practices (Beck, 1999a). This method has evolved as a response to limitations caused by traditional long development cycles in projects with vague or constantly changing requirements. XP is based on a set of four values that serves both human and business needs (Beck, 1999b):

- Communication – An essential key to project's success is communication between all members. With right communication, important information will not be lost and team members can synchronize their work better. Some XP practices focus on facilitating communication;

- Simplicity – The team must simplify the solution to be implemented as "the simplest thing that could possibly work", thus removing complexity, unnecessary features and waste. This value is related to communication, since the simpler the system is, the less information there is to communicate;

- Feedback – The team should get feedback throughout the project in order to solve issues early and improve the product while there is still time. This value works with communication and simplicity. The more feedback is given, easier it is to communicate. The simpler the system is, easier it is to test;

- Courage – It takes courage to work in fast-pace while collaborating side-by-side with team members. The team must take important decisions, make bold simplifications, change previous work or even start over. When combined with the first three values, courage becomes very valuable. Communication opens space for new ideas, simplicity keeps team members confident because the system is less likely break unknowingly, and feedback allows team members to know if the work is well done based on concrete results.

This chapter is divided into three sections to fully explain XP method – Practices, Roles and Project Life-cycle.

2.2.2.1 Practices

XP is well known for its set of software engineering practices which lead to successful software development (Abrahamsson, Warsta, Siponen, Ronkainen, & Ronkanen, 2003). These individual practices have been collected from already existing methods (Beck, 1999a) and combined to function with each other (Abrahamsson et al., 2003). The core practices of XP are presented in the following:

a) **The Planning Game** (Beck, 1999a, 1999b) – This practice is used to determine the next iteration's scope by combining technical estimations with business priorities. While customers are responsible for deciding about scope, priority, composition of releases and date of releases, the technical team is responsible for estimating, identifying consequences, defining the process (*i.e.* how work and team will be organized) and detailing the schedule. As new information becomes available, the plan is updated.

b) **Small Releases** (Beck, 1999a, 1999b) – The idea of small releases is to release a simple version of the system first and then release new versions gradually. Releases should be as small as possible. Each release should have the most valuable business requirements and make sense as a whole. Beck (1999b) suggests it is better to plan one or two months at a time than long periods such as six months or above.

c) **Metaphor** (Beck, 1999a, 1999b) – Metaphor is the way to describe the system to be developed in a set of stories. The description must be clear so that all stakeholders involved in the project can easily understand how the system should work.

d) **Simple Design** (Beck, 1999a, 1999b) – Design should be as simple as possible, avoiding duplicated code and ambiguity, making sure it runs all tests. It is reviewed iteratively to ensure it is appropriate for the project requirements and to remove extra complexity as soon as possible.

e) **Testing** (Beck, 1999a, 1999b) – XP is test driven, meaning the team writes unit tests prior to developing new code. Customers write functional tests to demonstrate that features are finished. Tests run continuously, giving early feedback and therefore flexibility to adapt. The system must pass all tests, however, sometimes business decisions must be made comparing the cost of releasing with defect and the cost of delay.

f) **Refactoring** (Beck, 1999a, 1999b) – Refactoring is the process of changing existing code in order to turn in simpler, remove duplication, improving communication and adding flexibility, without changing the system's behaviour. Although it may take more time to refactor then to build one functionality, it will save time when building other new functionalities.

g) **Pair Programming** (Beck, 1999a, 1999b) – All production code is written by two people at one machine. While one is writing the code, the other is evaluating if the whole approach is working, creating new test cases and finding a way to make the solution as simple as possible. This practice is dynamic, meaning pairs can change throughout the work day. This helps to share knowledge about the system through the team.

h) **Continuous Integration** (Beck, 1999a, 1999b) – New code is integrated in the system as soon as it is completed. Therefore, integrations are done many times a day. The system should run all tests correctly in order to accept the changes.

i) **Collective Ownership** (Beck, 1999a, 1999b) – The whole team is responsible for the whole system. Thus, anyone can change any code at any moment when an opportunity to create value is identified.

j) **On-site Customer** (Beck, 1999a, 1999b) – The project's real customer is full-time present and available for the team, ready to answer any questions. This can be the person who will use the system when it is in production, or a representative.

k) **40-Hour Week** (Beck, 1999a, 1999b) – The team must work a maximum of 40 hours per week. Overtime happens sometimes, but overtime two weeks in a row is not allowed. When this happens, it means there are bigger problems to be addressed.

l) **Coding Standards** (Beck, 1999b) – To simplify team work, where team members frequently work on different parts of the system, do integrations and change partners during the day, it is important to have coding standards. Coding standards should call for the least amount of work as possible, prevent duplicate code and emphasize communication during development., but essentially, it must be applied voluntarily by all team members.

m) **Open Workspace** (Beck, 1999a) – This XP practice suggests that the appropriate workplace is a large room with a specific order: individual cubicles for each team member are spread around the room and computers for pair programming are set up in the center.

n) **Just Rules** (Beck, 1999a) – Team members are committed to follow the rules, however, rules can be changed at any time. To do so, the whole team must agree to the new rules and the impact of that change should be evaluated.

2.2.2.2 Roles

XP has its own set of roles which have different responsibilities during XP life-cycle and practices. XP roles are (Beck, 1999b): Programmer, Customer, Tester, Tracker, Coach, Consultant and Big Boss.

a) **Programmer** – Programmers write code and often do refactoring to keep work as simple as possible. They also have the responsibility to write and run unit tests, making sure the system passes them. A very important skill is to have good communication with team members and keep the mindset of shared responsibility for the whole project.
b) **Customer** – While the programmer knows how to program, the customer knows what to program. In XP, the customer is responsible for writing user stories and writing functional tests. The customer also has the responsibility to make decisions such as defining priorities or changing user stories. A good customer is confident on the decisions, works closely with the team and uses the system being developed. According to Beck (1999b), the roles customer and programmer make the essential duality of XP.
c) **Tester** – The tester helps the client to choose and write functional tests. This role is also responsible for running functional tests regularly, reporting the results and making sure that the testing tools run well.
d) **Tracker** – The tracker keeps track and gives feedback of the project's progress. One of the tracker's responsibility is to collect the team estimates and compare them to the real duration of tasks, giving feedback to the team so that they make better estimations on the next iterations. The tracker is also responsible for keeping a log of all test results and reported defects, as well as evaluating whether the iteration goal is reachable within the limit of time and resources or if any changes should be applied to the process.
e) **Coach** – The coach is responsible for the process as a whole, deeply understands the XP process and helps everyone following it. It is important to have a coach when shifting to XP method, but as the team matures, the role of a coach diminishes.
f) **Consultant** – Consultant is a member of the team, but is an external person with deep technical knowledge that helps solving some problems that are keeping the team stuck. A consultant teaches the team how to solve the problem so that next time they are stuck for the same reason, they know how to solve it without the help of the consultant.
g) **Big Boss** – The Big Boss is responsible for making decisions and for keeping constant communication with the team to identify any difficulties or deficiencies on the process.

2.2.2.3 Project Lifecycle

An ideal XP project lifecycle is composed by six distinct phases (Beck, 1999b): Exploration, Planning, Iterations to First Release, Productionizing, Maintenance and Death. The workflow in Figure 2.4 illustrates all the six phases of the XP lifecycle.

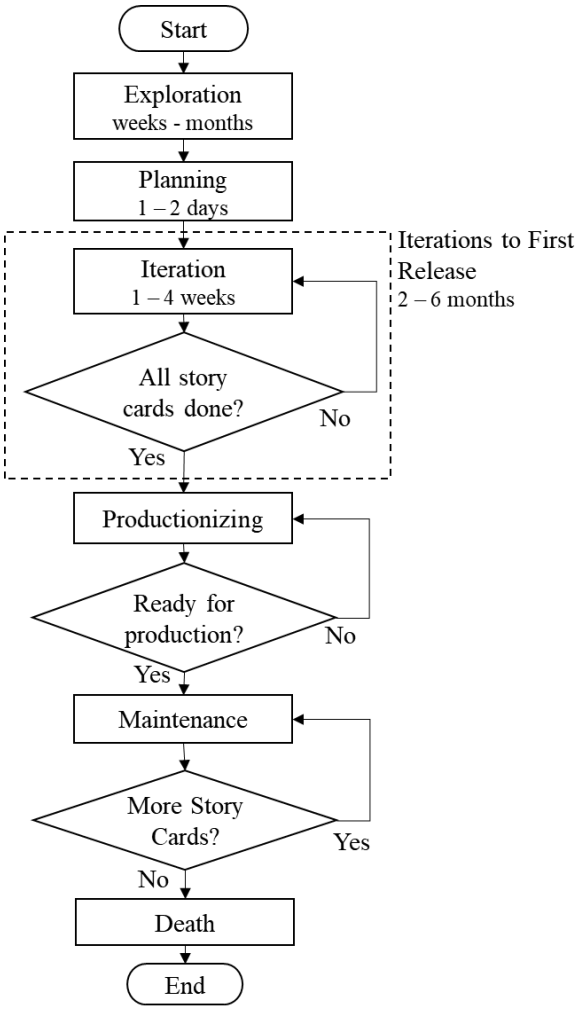


Figure 2.4 – XP Lifecycle Workflow

a) **Exploration** – An XP project starts with an exploration phase, where the customer writes all the story cards to be included in the first release (each story card corresponds to one feature), while the programmers get familiarized with the technology, tools and processes they will be using throughout the project. The programmers test the technology to be used and try different scenarios for a system architecture by developing a prototype. The exploration phase may take between a few weeks to a few months, depending solely on how comfortable the programmers are with the technology. This phase finishes when the customer completes all the story cards for the first release and the programmers are confident enough to make estimates (Beck, 1999b).

b) **Planning** – After the exploration phase, the customer and programmers practice The Planning Game (2.2.2.1 a)), where stories are estimated and prioritised, and both agree on a date for the first release, where the smallest and most valuable set of stories will be done. The planning phase takes around one to two days, and the schedule should cover between two to six months.

c) **Iterations to First Release** – The schedule for the first release is broken into one to four-week iterations, and each will produce a set of functional test cases for each story card. In the first iteration, the team must set up the system architecture. In the following iterations, the team builds the most valuable features in the story cards, prioritized by the customer. Ideally, at the end of each iteration, the customer runs all functional tests and they all pass. At the end of the last iteration, the system is ready to go into production.

d) **Productionizing** – The productionizing phase is when the system is released for customer use. This requires extra testing of the system performance before it is ready for production, while the feedback cycles get shorter, *i.e.* instead of three-week iterations, the team may take one week iterations and daily stand-up meetings. New changes may be identified, but the customer must evaluate and decide whether changes should be included in the current release or if they should be saved for later. The new ideas and suggestions are documented for later implementation during the Maintenance phase.

e) **Maintenance** – The maintenance phase is the normal state of an XP project. While the system is running in production, new functionalities are developed and incorporated in subsequent releases. Each release should start with an exploration phase, where the team can experiment big refactoring, a new technology or even different architectural ideas, while the customer explores new functionalities that are valuable for the business and writes them in story cards. The maintenance phase will also include customer support tasks. After exploration, the new release should be planned considering conservative estimations, since the team is working in a production system and therefore the development velocity tends to decrease. During maintenance, the team may change gradually, incorporating new people with personal coaching. This phase doesn't have an estimated duration, it should last as long as the project makes sense for the customer.

f) **Death** – The project reaches the death phase when the client is satisfied with the system and can no longer write stories to be implemented. In this phase, the whole system is documented, since the architecture, design or code will not be changed. The death phase may also occur if the system cannot deliver the desired outcomes, the client needs features that are too expensive to implement or if defect rate rises excessively.

2.2.3 Feature Driven Development

Feature Driven Development (FDD) is an agile and adaptive approach for building systems which derived from a set of industry-recognized best practices (Palmer & Felsing, 2002). According to Abrahamsson et al. (2002), FDD was first reported in (Coad, de Luca, & Lefebvre, 1999) and further

developed on a work done for a large software development project by Jeff Luca, Peter Coad and Stephen Palmer (Abrahamsson et al., 2002).

FDD provides methods, problem solving techniques, and reporting guidelines, ensuring information is shared by every stakeholder in a project. FDD does not cover the whole software development process, it rather focuses on the design and building phases, aiming to deliver frequent, tangible and working results iteratively. An FDD project begins with developing an overall model for the system, building a feature list, and planning the work. Then, the team goes through design and build iterations to develop the features (Palmer & Felsing, 2002).

This chapter is divided into three sections – Roles, Practices and Process – which correspond to the three elements Palmer & Felsing (2002) used to explain the FDD method.

2.2.3.1 Roles

FDD roles are classified into three categories: key roles, supporting roles and additional roles (Palmer & Felsing, 2002). According to Palmer & Felsing (2002) FDD roles are described as follows.

Key Roles:

a) **Project Manager** – Reports progress, controls budget and manages resources. The project manager is also responsible for providing a safe and productive environment for the development team.
b) **Chief Architect** – The chief architect is a deeply technical role and is responsible for system's overall design and makes the final decisions on all design issues. For projects complex in both domain and architecture, this role may be split into domain architect and technical architect.
c) **Development Manager** – This role has the responsibility for leading the development team in its daily activities and solving any conflicts. The development manager is also responsible for solving resourcing problems. This role can be combined with the chief architect or project manager role.
d) **Chief Programmer** – Experienced developer who participates in the high-level requirements analysis and design, and is responsible for guiding small teams through low-level analysis, design and development of features. The chief programmer combines great technical ability with people skills to lead the team into delivering complete features in a short period of time.
e) **Class Owner** – Developer who works under the guidance of Chief Programmer and is responsible for designing, coding, testing and documenting the system's features.

f) **Domain Expert** – The domain expert can be a client, user, business analyst, or a mix of these. This role has the business knowledge and is responsible for passing that knowledge to the developers, in order to ensure that a competent system is delivered.

Supporting Roles:

g) **Domain Manager** – In larger projects, the domain manager leads the domain experts and has the final word when there are indecisions in business requirements. In smaller projects, this role may be combined with the project manager role.
h) **Release Manager** – Controls and reports the project's progress to the project manager. The release manager receives progress reports from the chief programmers and holds short progress meetings with them.
i) **Language Lawyer/Language Guru** – A person who has deep knowledge of a specific programming language or technology. It is an important role when the team is working with a programming language or technology for the first time. Once the team has enough knowledge to work by itself, this role is no longer necessary.
j) **Build Engineer** – Responsible for setting up, maintaining, and running the regular build process, which includes managing the version control system and publishing documentation.
k) **Toolsmith** – Builds small tools for development, test and data conversion teams in the project. If necessary, the toolsmith may be responsible for setting up and managing a database or website for the teams' specific needs.
l) **System Administrator** – Responsible for configuring, managing and troubleshooting the servers, network of workstations and development, including the development environment and any specialized testing environments.

Additional Roles:

m) **Testers** – Responsible for verifying if the system's functionalities meet the user's requirements and that they preform correctly. Testers can be part of the Project Team or a quality assurance department.
n) **Deployers** – Deployers convert existing data to the format required by the new system, as well as participating in the deployment of new releases. Deployers can be part of the Project Team or an operation and system administration department.
o) **Technical Writers** – This role is responsible for writing and preparing user documentation. Depending on the organization, technical writers may be part of the Project Team or have their own department to serve all projects.

Usually, in smaller projects, different team members may play more than one of the listed roles.

2.2.3.2 Practices

FDD gathers a set of "best practices" which, according to Palmer & Felsing (2002), are not new, but this specific blend fits well to each case. To get the most benefit, all practices should be used, since no single practice dominates the whole process (Palmer & Felsing, 2002). The FDD practices are (Palmer & Felsing, 2002): Domain Object Modelling, Developing by Feature, Individual Class (code) Ownership, Feature Teams, Inspection, Regular Builds, Configuration Management and Progress Reporting. The FDD practices are presented below, as described by Palmer & Felsing (2002):

a) **Domain Object Modelling** – This practice consists of building a class diagram which identifies the most important objects within a problem domain and the relationships between them. It is a form of object decomposition and, according to its authors, the best technique is "modelling in colour" which allows to rapidly visualize information in the model.
b) **Developing by Feature** – The system's modules are decomposed into functional requirements which are small enough to be developed within two weeks. Each feature is expressed in the form of [action] [result] [object], with the appropriate prepositions between. Features are client-valued in order to facilitate the communication between the client and the development team. This way, the client is able to prioritize features according to its business value, as well as measuring the project's progress.
c) **Individual Class (code) Ownership** – Distinct pieces of code (classes from the domain object model) are assigned to a single owner, who knows exactly how a piece of code works and is responsible for its consistency, performance and conceptual integrity.
d) **Feature Teams** – Features are distributed throughout feature teams. Each feature team has a team leader (experienced developer) who is responsible for delivering the assigned features and to coordinate the efforts of multiple developers. Feature teams are small, typically three to six members, and dynamic. Between iterations, feature teams may change members in order to have the right class owners developing the assigned features. If needed, during a short period of time, class owners may be members of multiple feature teams at the same time (up to three teams concurrently).
e) **Inspections** – FDD emphasizes the practice of inspections to improve quality of code and design. Inspections have been proven to be very effective in detecting defects, but there are other benefits to it. Knowledge transfer happens when developers examine the code together and learn techniques from experienced developers. Lastly, once programmers know their code will be inspected, they are more likely to apply the agreed design and coding standards.
f) **Regular Builds** – This practice consists on integrating, at regular intervals, the source code of completed features and other components, building the complete system. The frequency of

builds depends on the project's size, *i.e.*in smaller projects have shorter intervals between builds. Regular builds allow to detect integration errors early, but also ensure that there is always an up-to-date system to demonstrate to the client.

g) **Configuration Management** – Configuration Management is applied specially for projects where the team is working simultaneously on different versions of a software system. Source code, analysis, design and any other artefacts that are used and maintained during the system's development should have a version control, thus keeping track of changes.

h) **Progress Reporting** – The project's progress reporting should be done at all levels, inside or outside the project, based on completed features. Within every feature, each of the six development phases is measured in percentage (Domain Walkthrough, Design, Design Inspection, Code, Code Inspection, and Promote to Build). The team identifies the current phase and accurately keeps track of the project's progress.

2.2.3.3 Process

According to Palmer & Felsing (2002), FDD consists in designing and building the system by following five sequential processes, illustrated in Figure 2.5: Develop an Overall Model, Build a Features List, Plan by Feature, Design by Feature and Build by Feature. The last two processes are iterative, which supports agile development, allowing late changes in requirements to fit business needs.

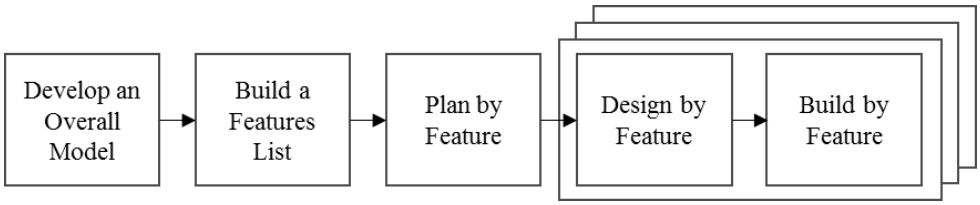


Figure 2.5 – FDD Process (Palmer & Felsing, 2002, p. 57)

Each process is explained below (Palmer & Felsing, 2002):

a) **Develop an Overall Model** – The team works together to develop an overall model for the system, with the guidance of a Chief Architect (2.2.3.1 b)). Domain experts (2.2.3.1 f)) start by presenting high-level requirements and its context to the team members – the authors call this a "walkthrough" – followed by more detailed walkthroughs for each area of the system's domain. After each walkthrough, domain and development team members work in small groups to create object models for that area of the domain. The models are then presented to the team for comparison and discussion, deciding on the most appropriate model for that domain area. The overall model shape may be adjusted, as needed.

b) Build a Features List – At this stage, the team has gathered valuable knowledge to start producing a comprehensive list of features. Use cases and functional specifications are used as inputs in this phase. Features are small and client-valued functions, composed by three elements: action, result and object. The team reviews each domain area and groups features into sets or major feature sets. Typically, a feature set corresponds to a specific business activity. Finally, users and sponsors review the feature list for validity and completeness.

c) Plan by Feature – Feature sets and major feature sets are sequenced, according to priority or dependencies, to form a high-level plan. Feature sets are assigned to chief programmers (2.2.3.1 d)), and the classes identified in the overall model are assigned to class owners (2.2.3.1 e)).

d) Design by Feature – A chief programmer selects a small group of features to develop during the iteration (up to two weeks long) and identifies the classes involved in those features. Class owners form feature teams according to the identified classes. Each feature team creates a detailed sequence diagrams, and then conducts a design inspection.

e) Build by Feature – The class owners develop code for their class, run unit tests, integrate, and finally inspect code. Once the code is inspected and the chief programmer validates the work, the completed features are promoted to the main build. A new set of features is selected for the next iteration, repeating process d) and e) until all features are done.

2.2.4 Other Methods

As mentioned in the beginning of chapter 2.2, many agile methods are found in literature. In this chapter other relevant methods are briefly presented, such as Dynamic Systems Development Method (DSDM) (Stapleton, 1997), Crystal family of methodologies (Cockburn, 2002), Rational Unified Process (RUP) (Kruchten, 2000), Adaptive Software Development (ASD) (James Highsmith, 2000), Lean Software Development and Kanban Development (D. J. Anderson, 2010).

a) Dynamic Systems Development Method

Dynamic Systems Development Method (DSDM) emerged in 1994 as a framework for Rapid Application Development (RAD), according to its consortium website[2]. The manual for this method, DSDM Atern Handbook, is available at the DSDM consortium website with open-access.

The main concept under DSDM is to fix time and resources while adjusting the number of functionalities, which is the opposite of a traditional approach, as illustrated in Figure 2.6 (DSDM Consortium, 2014, Chapter 2).

[2] www.dsdm.org 19.08.2017.

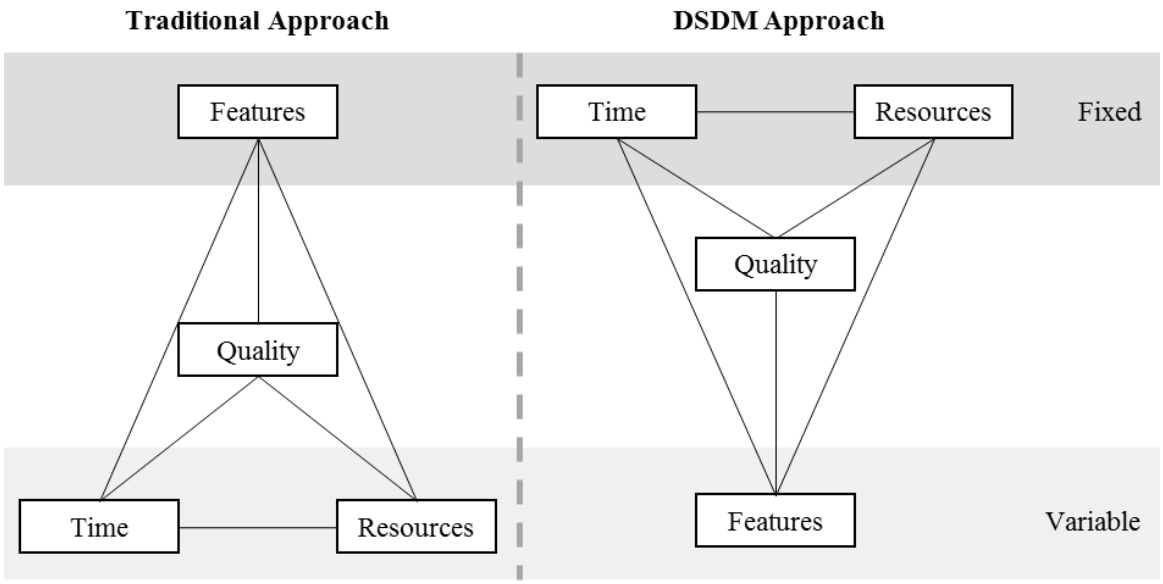


Figure 2.6 – DSDM Approach (DSDM Consortium, 2014, Chapter 2)

There are eight principles in DSDM approach (DSDM Consortium, 2014, Chapter 4):

- Focus on business needs
- Deliver on time
- Collaborate
- Never compromise quality
- Build incrementally from firm foundations
- Develop iteratively
- Communicate continuously and clearly
- Demonstrate control

The DSDM life-cycle consists in seven different phases, illustrated in Figure 2.7: Pre-project, Feasibility, Foundations, Exploration, Engineering, Deployment and Post-Project (DSDM Consortium, 2014, Chapter 6).

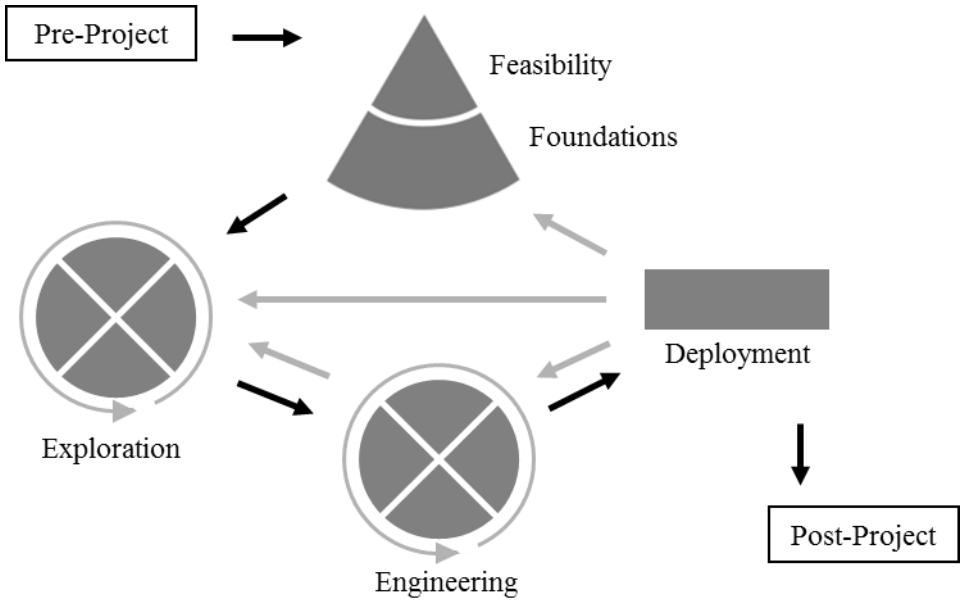


Figure 2.7 – DSDM Life-Cycle (DSDM Consortium, 2014, Chapter 6)

The DSDM phases are described according to DSDM Atern Handbook (2014):

- Pre-project – Formalize the project's proposal, clearly define the business objectives and identify who will be the project's Business Sponsor and Business Visionary.
- Feasibility – Evaluate the project from technical and business perspectives to assess its feasibility and decide whether it is cost-effective or not. Other approaches to the problem's solution should be outlined, a rough estimate of timescale and cost is carried out, and Foundations phase is planned.
- Foundations – Create a list of high-level requirements, establish fundamental understanding of the business process, design the solution's architecture, define how quality will be assured, describe how progress will be measured and reported, create a baseline for development and delivery and, finally, describe, assess and control the project's risk.
- Exploration – Iteratively and incrementally explore detailed business requirements and produce a functional solution that meets the business needs. Provide early vision to the wider organization of the final product where they will operate, support and maintain.
- Engineering – Refine and evolve the solution produced in the Exploration phase to ensure that it meets the agreed acceptance criteria regarding performance, capacity, security, sustainability and maintainability. This phase may work iteratively with the Exploration phase.

- Deployment – Implement the project's solution (or increment of it) into live use to confirm performance and viability. It is a key review point to re-plan as needed. During this phase, documentation and training is provided to any support staff. With the final deployment, the project is formally closed and the overall performance can be reviewed from a technical, process and/or business point of view.

- Post-project – This phase occurs after the last planned deployment of the solution. It is meant to assess whether the benefits described in the business case were actually achieved with the deployed solution. It starts as soon as the value can be measured (three to six months after the project's completion).

b) Crystal

Crystal gathers a set of different methods to choose the most suitable method for each project (Cockburn, 2002). As illustrated in Figure 2.8, projects are characterized by criticality (vertical axis) and team size (horizontal axis).

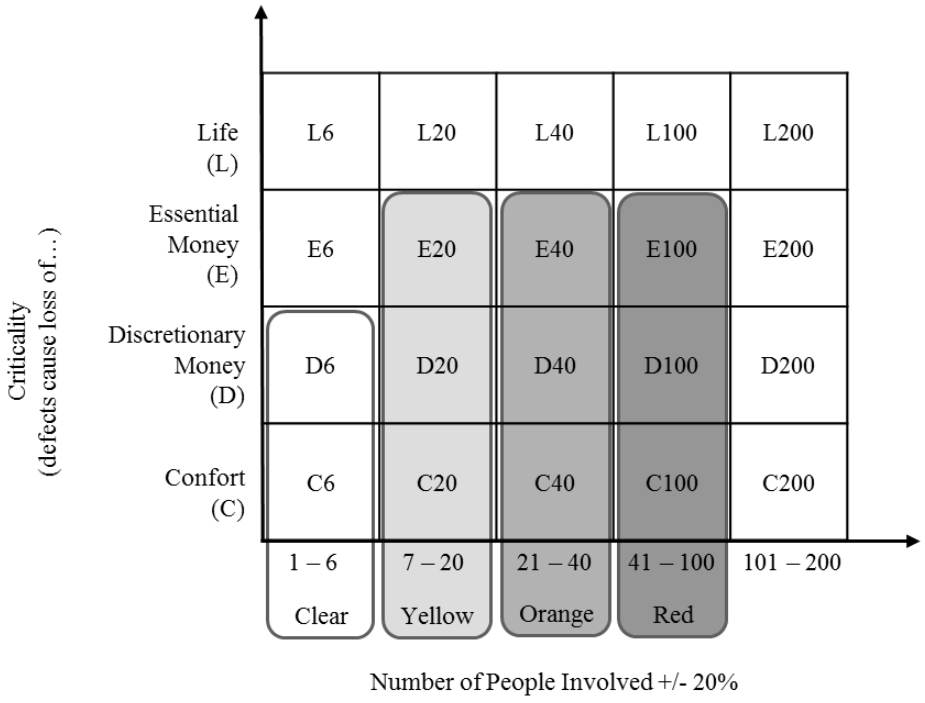


Figure 2.8 – Crystal Methods by Project's Characteristics (Cockburn, 2002)

The author associates a colour to each method, indicating how heavy it is. A Crystal Clear is aimed to small projects with low criticality, which should be light and require less bureaucracy. However, larger projects with more people involved and higher criticality should use a heavier method with more

documentation and rules to ensure efficient communication and introduce additional validation and traceability (Cockburn, 2002).

By observing the Figure 2.8, a life-critical project should not use an agile approach, i.e. the project's criticality does not show any relevance on finding and appropriate agile method. On the other hand, the number of people involved in a project defines how agile the adopted method should be – the less people are involved in a project, the more agile-prone it is.

The Crystal methodology does not provide specific guidelines on how to implement the different methods. However, all Crystal methods share the same agile principles (Cockburn, 2002):

- Frequent delivery – Building increments and deliver frequently to obtain acceptance.
- Continuous improvement – The team regularly reflects on how to improve work and implements changes as needed.
- Personal safety – Team members have a work environment where they feel safe to raise issues and ask questions.
- Osmotic communication – Team members work in the same location in order to efficiently share information.
- Focus – Team members have well defined tasks and have time to work on them without interruptions.
- Easy access to expert users – Expert users are available to give feedback to team members.
- Technical environment – Automated tests, configuration management and frequent integration.

c) Rational Unified Process (RUP)

The Rational Unified Process is a process framework for iterative software development and it was created by Rational Software Corporation – a software company acquired by IBM. RUP is considered an agile method since it "embraces change" and its processes are based on iterative development. However, during the project's lifecycle RUP adopts "heavy documentation" which contradicts agile principles (Borth & Shishido, 2013). RUP was built based on the following software best practices (Kruchten, 2000):

- Develop software iteratively – Allows user feedback, spreads workload throughout the project's lifecycle, facilitates an objective monitoring of the project's status, sharing it with the stakeholders.

- Manage requirements – Identify which requirements add most value to the company, organize and document the system's required constraints, analyse and measure the impact of requirement changes, and tracking and documenting trade-offs and decisions.
- Use component-based architectures – System's architecture description allows shared understanding.
- Visually model software – Facilitates the development team to specify, construct and document the system's behaviour.
- Continuously verify software quality – The system's quality is assessed continuously regarding functionality, reliability and performance to avoid fixing problems after deployment.
- Control changes to software – Coordinating developers' activity is important in order to track changes, find and correct problems.

RUP is composed by two structures – the Static Structure and the Dynamic Structure. As shown in Figure 2.9, the Static Structure (organization along content) represents how workers and activities interact across workflows, while the Dynamic Structure (organization along time) illustrates the phases, iterations and milestones in a project's lifecycle (Kruchten, 2000).

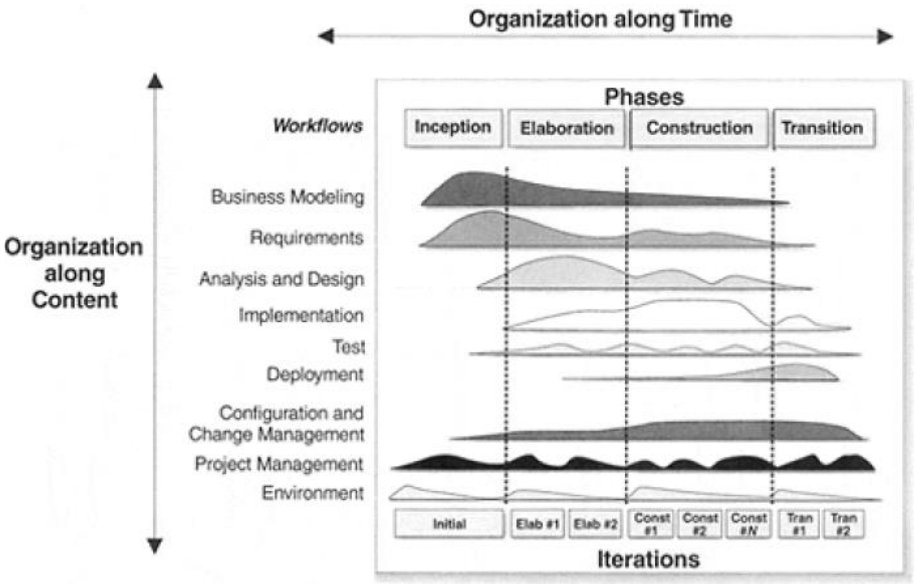


Figure 2.9 – RUP Process Structure (Kruchten, 2000)

RUP workflows contain the following activities (Kruchten, 2000):

- Business Modelling – Develop a vision and define processes, roles and responsibilities in the organization, using a business use-case model and a business object model.

- Requirements – Identify functional requirements for the system, as well as non-functional requirements such as usability, reliability, and performance.
- Analysis and Design – Specify the identified requirements into a concrete design model.
- Implementation – Organize the code to be developed, implement and test components as units, and integrating them into a working system.
- Test – Run tests on the component's interaction and its integration, validate if the requirements are compliant to its specification, and find and fix all possible defects.
- Deployment – The software is in its final operational environment where it is tested, packed, distributed and installed as a final product. This phase includes training for end users.
- Configuration and Change Management – To track and maintain the integrity of evolving project assets (with version control), and gather metrics related to functionalities' changes.
- Project Management – Includes activities for people managing (hiring, training and mentoring), budget management, contract management, risk management, and project progress monitoring.
- Environment – Provides support regarding tools, processes, and methods, avoiding human-intensive and error-prone activities.

While the Static Structure is graphically documented, easing process' interpretation and team's responsibilities regarding activities, the Dynamic Structure is composed by four distinct phases across the project lifecycle, and end of each phase represents a project milestone (Kruchten, 2000):

- Inception – The project vision is created by the team, followed by business case and its scope. This phase ends with the Lifecycle Objective Milestone.
- Elaboration – The necessary activities and resources are planned, features and requirements are specified, and the system's architecture is designed. This phase is concluded with the Lifecycle Architecture Milestone.
- Construction – The team builds the product and evolves the project's plan and vision, ending this phase with the Initial Operational Capability Milestone.
- Transition – Transitioning the product to its end users. The Product Release Milestone marks the end of this phase, concluding the complete cycle.

What differentiate RUP from other Agile Methods is that is includes artefacts related to Project Management such as Product Acceptance Plan, Risk Management Plan, Measurement Plan, etc. However, RUP does not request the use of all its artefacts, since it is adaptable to different projects, serving a few guidelines for what artefacts to implement. RUP defines 30 different roles, including stakeholders, technical and administrative roles, and also reviewers and analysts for both technical and administrative tasks.

According to (Edeki, 2013), the Agile Unified Process (AUP) emerged from this method, which is a cut-down and simplified version of the RUP.

d) Adaptive Software Development (ASD)

Adaptive Software Development (ASD) suggests how projects should be approached from a cultural and organizational point of view. Its author, James Highsmith (2000), states that instead of a set of rules and tasks, ASD provides a framework of concepts, practices and guidelines.

The ASD lifecycle is based on six main characteristics (James Highsmith, 2002):

- Mission focused – Projects have mission statements that provide direction and criteria for making key project decisions, but also to guide the development work. The project's mission may evolve during the project's lifecycle;
- Feature based – ASD focuses on results instead of tasks. The results are broken into smaller pieces, identified as features, which are the customer functionalities to be developed during an iteration. Although documents are considered deliverables, they are secondary compared to features, since features provide direct results to the customer.
- Iterative – Development is done through iterations. Features may evolve over iterations as feedback from customers is provided.
- Time-boxed – Time-boxing is about focusing and forcing hard trade-off decisions in order to get work finished, not about forcing deadlines;
- Risk driven – Adaptive iterations result from analysing the critical risks, contributing to requirements' prioritization.
- Change tolerant – ASD embraces change as a competitive advantage, which is a characteristic present in every agile method.

ASD development life cycle is divided into three different phases (James Highsmith, 2002):

- Speculate – The planning stage where there is some uncertainty associated.
- Collaborate – Development is carried out based on teams' cooperation.
- Learn – Project's progress review and evaluation in order to identify possible changes on requirements. Each cycle should last between four and eight weeks.

ASD recommends other practices such as pair programming, shared ownership, and customer deployment (James Highsmith, 2002), which are also mentioned in other agile methods.

e) Lean Software Development

According to Highsmith (2002), Lean Development can be used as a management tool for software development, which emerged from the lean manufacturing philosophy. It has some practices and techniques in common with other methods, such as Kanban, but what differentiates Lean Development is the relation with the Toyota Production System philosophies.

Lean Development has 12 principles, which are (James Highsmith, 2002):

- Satisfying the customer is the highest priority;
- Always provide the best value for the money;
- Success depends on active customer participation;
- Every Lean Development project is a team effort;
- Everything is changeable;
- Domain, not point, solutions – solutions that can be used in multiple domains;
- Complete, don't construct – obtaining available solutions should be considered first;
- An 80 percent solution today instead of 100 percent solution tomorrow;
- Minimalism is essential;
- Needs determine technology – objectives must be considered before the technology;
- Product growth is feature growth, not size growth;
- Never push Lean Development beyond its limits – the method's limits must be understood.

f) Kanban Development

Kanban can be applied in software development, based on the technique within lean production. It is mainly used to limit Work-in-Progress (WIP) and visualizing workflow. Kanban consists of using a board divided into columns, each representing a software development phase. Labels represent items in a backlog and are distributed by the columns according to the item's development stage (D. J. Anderson, 2010). A Kanban board is shown in Figure 2.10. The first column has the project's backlog, and the second column shows the selected items to be developed – the number of items cannot exceed the established WIP limit (ex. four items). If the limit is reached, no new items are moved to the column until other items have moved out. The Development and Acceptance columns also have WIP limits.

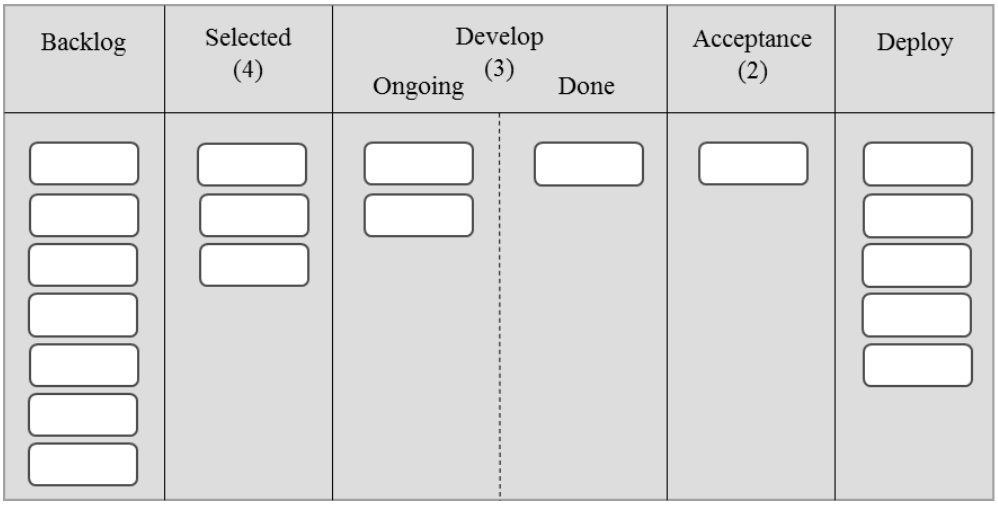


Figure 2.10 – Kanban Board for Software Development (D. J. Anderson, 2010)

There are five core properties in Kanban development (D. J. Anderson, 2010):

- Limit Work-in-Progress – Keeping the amount of work in progress low helps visualizing issues and bottlenecks, thus facilitating continuous improvement.
- Visualize Workflow – Visualizing the workflow is important for organizing, optimizing and tracking its progress.
- Measure & Optimize Flow – Tracking the workflow through a system can help identifying issues and changes can be measured for effectiveness.
- Make Process Policies Explicit – The team must have clear understanding how things work to ease open and objective discussions about improvements.
- Manage Quantitatively – Through scientific measurement and experimentation, the team should collectively improve the processes which are being followed.

2.3 Agile Methodology Applied on the IT Sector

Information Technology (IT) surrounds us in everything we do nowadays. IT is the use of computers and telecommunications equipment, with their associated microelectronics, to send, receive, store and manipulate data (Oxford English Dictionary, 2015).

To understand what an IT project is exactly, it is important to look at two distinct definitions:

- Project – "a temporary endeavour undertaken to create a unique product, service or result" (PMI PMBOK Guide 5th edition);
- Information Technology (IT) – "the study or use of systems – especially computers and telecommunications – for storing, retrieving, and sending information"(Oxford English Dictionary, 2015).

Just like any project, an IT project is temporary, meaning it has a beginning and closing time. During that time, the project requires an effort to achieve a certain result. Therefore, resources are assigned and a scope is defined. However, a unique product isn't necessarily a new product since it can be a modification to an already existing one, which makes it new in that context. An IT project is about the creation, modification, or study of computer or telecommunication system within a defined timeframe.

Projects in IT consulting are mostly software development projects, where it is very common to follow an Agile approach in many companies (Version One, 2015).

When compared to other types of projects – e.g., manufacturing – IT projects are usually short-term and have greater uncertainty. Because of the technological challenges relating to hardware and software misconfiguration, interoperability issues, or security risks, IT projects have higher failure rate. (Weigartner et al., 2015)

IT Consulting is "the study or use of systems – especially computers and telecommunications – for storing, retrieving, and sending information" (Oxford English Dictionary, 2015).

Most of the agile implementations found in the literature have a Scrum baseline, which is expected, since it is the most adopted method within the industry according to the *State of Agile Survey* (Version One, 2016).

Lagerberg et al. (2013) is an example of a Scrum implementation case study in a large-scale software development project where the authors concluded that the method can improve knowledge sharing, project visibility, and coordination effectiveness. The study also suggests that the method may improve productivity. However, no concrete evidence proves this observation (Lagerberg, Skude, Emanuelsson, Sandahl, & Ståhl, 2013).

In contrast, Carvalho & Mello (2012) studied the implementation of Scrum in a smaller context, also concluded that the method improves collaboration and communication. In this case, the authors observed a productivity increase with the implementation (Carvalho & Mello, 2012).

Despite Scrum being the most common method in the case studies found, Abdullah & Abdelsatir (2013) is an example where XP was implemented. The authors did not observe improvements in productivity or quality during the case study. However, the project team members' engagement was positively impacted (Abdullah & Abdelsatir, 2013) – which can bring more benefits in a longer term.

The combination of both Scrum and XP is also present in the literature as implementation case studies, which is the second most common implemented method according to the *State of Agile Survey* (Version One, 2016). Santos & Córdova (2017) show how the implementation of a Scrum/XP hybrid method can be deployed to bring efficiency increases in the software development process (Santos & Cordová, n.d.). Gannon (2013) also shows how Scrum and Kanban can be effectively adopted in software development projects (Gannon, 2013).

Fetouh et al. (2011), on the other hand, developed a hybrid method between agile and plan-driven for an ERP implementation project. Showing the importance of considering the characteristics of both the company and the project when adopting agile methods (Fetouh, Abbassy, & Moawad, 2011). Ferreira et al. (2011) also shows how agile can be tailor made to fit organisational structures – in this case for the collaboration between UX designers and software developers (Ferreira, Sharp, & Robinson, 2011).

Lastly, Diaz et al. (2009) show how agile methods and the Capability Maturity Model Integration (CMMI) overlap, so a company can simultaneously comply to industry standards and adopt agility to improve its software development processes (Diaz, Garbajosa, & Calvo-Manzano, 2009).

In retrospective, the implementation of agile methods can improve productivity, team engagement, and communication. However, it is important to adapt the implemented method according to the project's characteristics. It is possible to combine different agile methods and combine both plan-drive and agile to better fit the project needs. Finally, even though agile methods promote the use of lightweight documentation, they can still coexist with demanding standards in terms of process definition such as the CMMI.

3 Case Study

3.1 Glintt: Company's Presentation

3.1.1 History

Glintt, which stands for Global Intelligent Technologies, is a technology and consultancy multinational quoted in Euronext. It was founded in 2008 as a merge of the companies ParaRede and Consiste. Currently, it is one of the biggest Portuguese technological companies (Oliveira, 2017). Glintt employs more than 1000 people, who work at 10 offices located in 6 countries: Portugal, Spain, Angola, Brazil, the United Kingdom and Ireland. The operating income in 2017 totaled €6,2M.

Associação Nacional de Farmácias (ANF) is Glintt's larger shareholder, owning 76,4% of its equity. Glintt is part of ANF group, which is also formed by 19 other companies such as Farminveste, HMR and Alliance Healthcare amongst others, as shown in Figure 3.1.

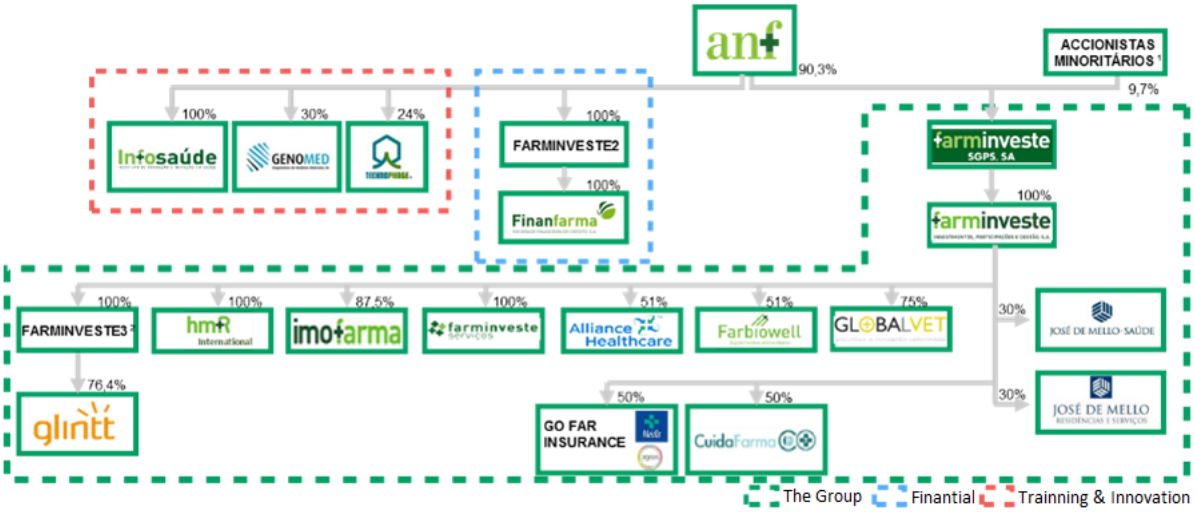


Figure 3.1 – Glintt's Organization Chart

Glintt offers IT and consulting solutions focused in the Healthcare area. More than 200 hospitals and clinics benefit from Glintt's solutions, and over 12.000 pharmacies in the Iberian Peninsula use Glintt's management software. Glintt offers a wide range of products and services such as shop design, automation software, infrastructure solutions and consumables, among others.

Apart from the Healthcare market, Glintt also serves Financial Services, Telecommunications and Public Administration sectors, where it has contributed for some of Europe's biggest and most competitive public and private entities, having implemented highly complex transformation projects, and providing application and infrastructure management services.

Glintt values innovation, making significant investments in Research and Development, establishing partnerships with universities, and participating in many international consortiums. Glintt is part of the Portuguese Scientific and Technological System, is certified by ISO9001, NP4457, ISO27001, and accredited by Gabinete Nacional de Segurança.

Glintt operates in five markets:

- Healthcare – Solutions specifically designed for public and private organisations, in the fields of Pharmacy, Nursing, Biomedicine, Technology, IT Systems, Mathematics and Management. These cover all hospital services, from patient admission to prescription, recording of consultations and medicinal products, invoicing and contacts between patients and the healthcare facility.
- Pharma – Design, together with Pharmacies, health and wellness areas where architecture and profitability are integrated with the most recent technologies. The main focus areas are Architecture, Software, Hardware, Construction, Building Works, Robotics and Image, to create innovative pharmacies. For 16 years, Glintt has completed more than 950 building and remodelling projects for Pharmacies and Clinics. Glintt Pharma's services are: Project's and Design, Management and Operations, Digital Technologies, Automation and Logistics.
- Financial Services – The financial services industry is facing digital transformation, both in Portugal and abroad, where businesses are replacing its physical model with digital systems, changing the traditional model to a model based on new customer interaction channels and solutions. Glintt operates in this market for 20 years, acting on digital transformation to help companies improve their business processes, making them more efficient and easier to control.
- Telecommunications – Glintt has established partnerships to provide innovative solutions for the Telecommunications and Utilities sectors and meet market's needs. Glintt responds to modern IT demands, particularly in the Data Centres area, from Convergence to Virtualisation, including SDx and NFV solutions. Using technologies such as RAD, App Factory, BPM, ITSM, BigData, Managed Services and proprietary products, amongst others, Glintt participates in digital transformation processes.
- Public Sector – Projects for administrative modernisation, process simplification for public administrative bodies, digital transformation and the automation of services, whether internal or oriented to citizens and businesses, resulting in increased efficiency.

In order to operate in those markets, Glintt offers products and solutions divided into six categories:

- Business Consulting – Provide pharmacies and hospitals with useful management tools and know-how.

- IT Infrastructure – Cloud solutions, networking configuration and projects related to IT security.
- Physical Design & Automation – Robotic solutions and software solutions for pharmacies and hospitals.
- Software Solutions – Digital management solutions for accessibility and mobility in hospitals, software for managing beds in hospitals, schedule appointments in hospitals and clinics, process optimization solutions, and management software for pharmacies, among others.
- Support Services – Integrated application and equipment support, setup and warehouse services.
- IT Consulting – Technology and applications consulting services, such as project management consulting, configuration and implementation of proprietary and third-party applications. Development and implementation of tailored IT solutions, including mobile platform and open source applications, as well as application maintenance. With a strong presence in the Healthcare and Pharmacy markets, projects were carried out in other sectors such as Insurance, Public Sector, Financial Services, Telecommunications, Industry and Retail. Projects were implemented in Portugal, Spain, Brazil, the United Kingdom, Ireland, Belgium, France and Angola, amongst other countries.

3.1.2 ITC Solutions Delivery

Information Technology Consulting (ITC) department provides technology and applications consulting services, such as Business Process Management, Applications Consulting, Data Analytics and Technology Consulting. This department also offers assistance for the development and implementation of tailored IT solutions, including mobile platform and open source applications, as well as application maintenance. Teams are multidisciplinary, dynamic, collaborative, including IT professionals and experts in business management and operations. The ITC department offers the following services:

- Business Process Management – Provides consulting services through the parameterization and development of solutions that allows acceleration and simplification of business processes implementation as well as solutions for managing business rules. Many projects are carried out with the use of Gx, a Glintt product that facilitates rapid modelling, implementation and optimization of business processes, giving autonomy to its end users.
- Applications Consulting – Offers services such as E-Signing (possibility of dematerialization of forms, agreements or other types of documents that require signatures) multi-channel marketing automation processes (based on behaviour and dynamic segmentation of customers) and general regulation for data protection.
- Data Analytics – Services are based on development and implementation of business intelligence solutions that help interpret facts, relationships and trends for new business opportunities, to research, discover and sustain changes to more efficient and cost-effective

processes. This area also covers implementation of end-to-end, integrated solutions, accessible from multiple devices, providing management information in a quick and decision-oriented manner.

- Technology Consulting – Offers custom application development across multiple platforms and programming languages, including mobility applications and open source applications (Low-Code/Java/.Net/Opensource/Outsystems). Implements projects related to mobile solutions (native, hybrid or web responsive), integrated platforms for administration and applications monitoring, as well as performance monitoring and troubleshooting.

The ITC department is divided into two areas: Solutions Design and Solutions Delivery as shown in Figure 3.2.

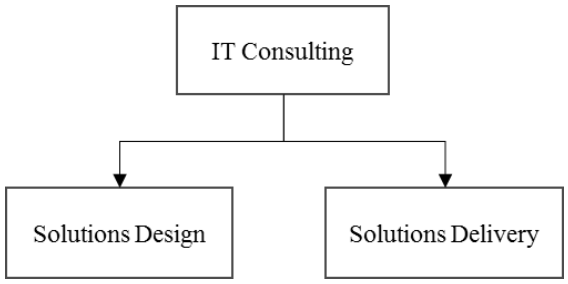


Figure 3.2 – Gintt's ITC Department Structure

Both areas serve three markets: pharma, healthcare and corporate. Solutions Design works on pre-sales activities such as identifying opportunities for new projects, communicating with potential clients and designing solutions that result in projects' proposals. Solutions Delivery is focused on projects' management and implementation after its adjudication.

Solutions Delivery area is composed by 296 workers spread by 11 competence centers, as shown in Figure 3.3. Clinical, Hospital Management, Pharmacy and Logistics, and MCDT are competence centers for Health Systems (HS). Social&Experience, Middleware, Java Software (SW) Development and Microsoft SW Development are competence centers for Application Development (App Dev). Transversal competence centers are App Management & PMO, Outsystems SW Development and BPM/ODM/ECM & RPA. In the past year, the ITC department had around a total of 400 projects, 50 of which were undertaken with Outsystems technology using agile approaches.

Glintt's internal process standards cover all company's processes and are certified by ISO9001. This is the international standard in the family that specifies requirements for quality management systems when an organization needs to prove its ability to provide products and services consistently, meeting the customer and regulatory requirements, as well as improving customer satisfaction with effective system applications (ISO, 2009).

Figure 3.3 – ITC Solutions Delivery Competences

3.2 Current Processes Analysis

Glintt has internal process standards which are available for all employees at the company's internal website. As this dissertation focuses on IT projects solely, the process analysis will only address Glintt's IT project management process. The current project management process used at Glintt is the RUP method and it is composed by five phases: initiation, planning, execution, monitoring, and closing, as shown in Figure 3.4 and explained in Table 3.1.

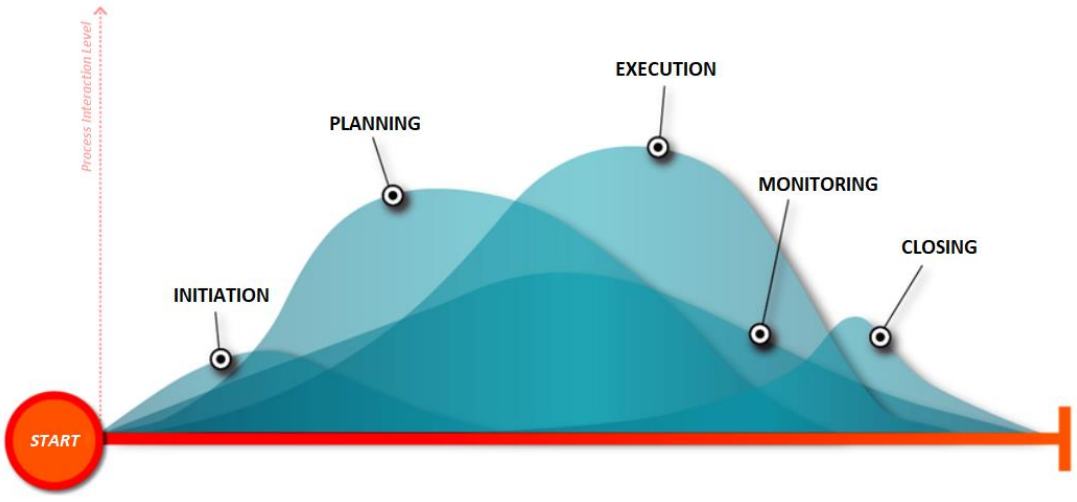


Figure 3.4 - Glintt's Project Management RUP Method

Table 3.1 – Glintt Project Management Activities

Phase	**Activities**
Initiation	• Perform internal kick-off • Review project's scope • Create project folder • Configure project management and execution process
Planning	• Plan project management process • Define high level chronogram and project baseline • Plan risk response • Review project plan with client • Perform external kick-off • Define project's goals and select work items • Identify activities • Create project chronogram
Execution (technical process)	• Implement standard management solutions • Implement client infrastructures • **Software development** • **Agile software development (process to be defined)** • Implement healthcare solutions • Construct / remodel pharmacies • Business consulting healthcare
Monitoring and Controlling	• Follow-up meetings • Register consumed effort • Produce project's financial report • Quality assurance • Communicate and follow project's progress • Manage stakeholders team • Manage change requests • Share information
Closing	• Obtain project's acceptance • Guarantee period • Assess client's satisfaction • Register lessons learned • Update and archive project's folder • Update competences

Execution phase comprises the software development process, used in ITC projects, which is composed by four main phases as shown in Figure 3.5: **1. Definition, 2. Conception, 3. Construction,** and **4. Transition**. There are six roles in this process – Analyst, Tester, Project Manager, Architect, Technical Manager, and Developers – and each role's activities are well defined in the process.

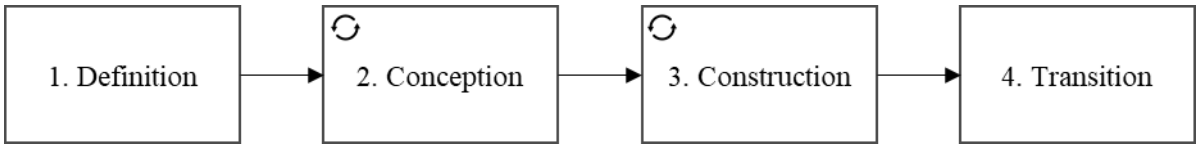


Figure 3.5 – Glintt's Software Development Process

1. **Definition** – The purpose of this phase is to establish the project's vision and high-level plan at the beginning of the project, outline architecture and obtain acceptance.

    1.1. Define System's Vision – Analyst and Stakeholder work together to define the project's vision, capturing customer needs and characteristics for the system to be developed and define project's scope. The Project Manager proposes a project plan that includes milestones for the four phases with defined duration for each phase. This plan and the allocation of resources for the project may evolve throughout the project to fit its pace in the different phases. The project plan may integrate with the other plans that affect the project.

    1.2. Identify and Outline Requirements – Analyst understands and prioritizes the stakeholders' needs and associated requirements for the system, specifying them in a way that supports effective communication and collaboration between Stakeholders and the development team. Requirements should be recorded in the Functional Requirements Document as well as their priorities in order to facilitate project planning. Requirements (high level) which must cover the entire project's scope. The requirements are described and detailed in Use Case Specification and Supplementary Requirements (non-functional requirements, and Interface Requirement Specification) with enough detail to provide information for the Architect to create the architecture prototype and for the Project Manager to define the scope project's plan.

    1.3. Outline Architecture – Architect outlines the system's architecture to provide high-level design and maximize benefit to Stakeholders, complying with the existing constraints in the system to be developed and the development team. Based on the requirements that are captured and detailed in parallel to this activity, the Technical Manager analyzes the technical restrictions and uses the available technology and resources to propose an architectural proof of concept. This architecture prototype is used both to demonstrate the feasibility of the project, and to mitigate significant risks regarding system's architecture. At this stage is defined whether all components are to be developed internally or if there is a need to purchase.

1.4. Set-up Development Environment – Technical Manager configures the environment by installing Software Development Tools on servers and / or team members' workstations, as well as platforms to support the proposed solution (application servers, portals, etc). It is also necessary to create a repository dedicated to sources versioning, according to the structure defined by the Architect.

1.5. Control Deliverables Quality – Project Manager verifies deliverables for conformity to specifications prior to customer delivery – in this phase, the deliverables are requirements specification, system architecture, and project's plan.

1.6. Obtain Deliverables Acceptance – Client validates the deliverables and the acceptance is formalized with acceptance minutes, in project meeting minutes or sent by e-mail. This activity occurs at the control points defined in the plan. All change requests, corrective actions or improvements resulting from this activity must be recorded for further analysis and processing in the project monitoring and control task. Change requests that require formal client authorization must follow the "Change Management" procedure.

2. **Conception** – In this phase one or more iterations are planned and executed to establish project's architecture and mitigate major technical risks. The architecture is tested to assess if it is compliant with non-functional requirements and has strong performance, to then start the development.

2.1. Identify and Outline Requirements – Analyst captures functional and non-functional requirements. These requirements form the basis of understanding and communication between stakeholders and the project team about what the system should do to meet customer needs. The purpose of this activity is to understand the high-level requirements in order to determine the work scope to be performed. These requirements will be subject to more detailed analysis before implementation begins.

2.2. Detail Use-cases – Analyst specifies the details of most significant requirements from a high-risk architecture point of view to be possible to use this information as input to the architecture and development activities in the current iteration. Use cases are described with enough detail to be validated and understood, according to client's expectations and to be implemented by the team accordingly.

2.3. Detail Cross-system Requirements – Analyst specifies cross-system requirements (non-functional requirements, rules, and interfaces) that are not described in the use-case specification. These documents are approved in the end of this phase.

2.4. Implement Solution – Developers do code implementation to provide functionality or fix defects.

2.5. Integrate and Create Build – Developer integrate all the changes made by all team members to the repository where the code is stored and to perform minimal tests on the system increment in order to validate the respective build. The goal is to identify integration problems as quickly as possible, so they can be easily fixed. This phase must follow the procedure of Continuous Integration.

2.6. Refine Architecture – Technical Manager describes the architecture decisions that will serve as a basis for development, considering all the design and implementation products that have been developed to date. The architecture documentation is updated to reflect changes made during development. This activity's results are registered and communicated to the project team using the Architecture Notebook.

2.7. Control Deliverables Quality – Project Manager verifies deliverables for conformity to specifications prior to customer delivery. All nonconformities and defects found must be recorded in the Occurrence Management Tool (Mantis).

2.8. Obtain Deliverables Acceptance – Similar as described in 1.6.

3. **Construction** – Software is developed in one or more iterations which increment the system's functionalities. In the end of each iteration, functionalities are tested for quality assurance shared to obtain client feedback.

3.1. Create Test Cases – Tester defines test cases and test data so that the requirements of this iteration can be tested.

3.2. Refine the Solution – Technical Manager identifies the key elements of the architecture, their interactions, behaviors, relationships and the data needed to perform a set of functionalities. This activity focuses on a part of the system and not on the whole project. It is applicable to a small set of requirements selected according to their criticality of business and / or greater technical risk – the result should be documented in the Software Architecture Document.

3.3. Implement Unit Tests – Developer implements one or more tests that allow individual elements validation in the produced code, during its development.

3.4. Implement Solution – Developer does code implementation to provide functionality or fix defects and this activity may be done in parallel with 3.5 activity.

3.5. Execute Unit Tests – Developer runs tests in individual deployment elements to verify that their operation occurs as specified.

3.6. Integrate and Create Build – Developer integrate all the changes made by all team members to the repository where the code is stored and to perform minimal tests on the system increment in order to validate the respective build. The goal is to identify integration problems as quickly as possible, so they can be easily fixed. This phase must follow the procedure of Continuous Integration. There are different procedures for each technology: Microsoft, Java, Outsystems.

3.7. Implement Test Scripts – Tester implements step by step test scripts to validate a solution build.

3.8. Run Tests – Tester runs the test scripts, analyzes results, reports issues (Mantis Management Tool) and registers the test results either in the Test Execution Tool used in the project or in the Test Reports documents on this site.

3.9. Control Deliverables Quality – Similar as described in 2.7.

3.10. Obtain Deliverables Acceptance – Similar as described in 1.6.

4. **Transition** – Once the system is developed, overall quality is refined in one or more iterations in order to guarantee it is ready to be deployed in production environment, followed by the provisory acceptance and then the definitive acceptance by the client.

4.1. Prepare Software Distribution – Technical Manager prepares users for final software distribution. The steps to be taken depend on the nature and scope of the project:

- Prepare Documentation: Finalize relevant system documentation such as operation and administration manuals, user manuals, delivery notes, etc.
- Preparing Training for Users: Preparation and implementation of training actions for users and elements of operations and support team.

4.2. Prepare Quality of Deliverables – Similar as described in 2.7.

4.3. Obtain Deliverables Acceptance – Client verifies completed deliverables, with provisory acceptance of the project. This activity occurs at the control points identified in the plan for this purpose. These outputs are necessary and sufficient for the final acceptance and to start Warranty phase. All change requests, corrective actions or improvements resulting from this activity must be recorded in the Mantis Management Tool for further analysis and processing in the project monitoring and control task.

In the current process, the plan is made at the beginning and approved by the client. Any changes to it are made just as necessary and there is a process for managing change requests, considering it should not occur so frequently. Requirements specification involves detailed and extensive documentation at the beginning with low client collaboration for establishing priorities. Follow-up activities are undertaken but not as frequently as every day stand-ups, and the client will test the system in the end of the project iteration – after the development has finished and quality assurance tests were taken.

As observed daily in the company, project's deliverables regarding requirements and test scripts are documented and shared with the client by e-mail, and most project's control activities are done using Excel. The process does not suggest the use of a centralized software to list all the project's deliverables, define and change priorities, set effort estimation and real effort, allocate tasks to team members, change tasks status, register quality assurance tests and acceptance tests results for each item, to assure overall traceability.

Execution phase refers an agile process that needs to be defined. Most software development projects follow the current process, however, at least 50 in 400 projects used agile methodology mostly based on Scrum. After speaking with some project managers in the company and their teams, some of them find that the current software development processes do not fit the project's needs and end up creating their own process based on agile approaches. This can be a problem because the processes used are based on each manager experience and the results are not analysed to verify if they bring value. Also, there is no agile standards at Glintt to allow process certification and continuous improvement.

4 Proposed Agile Model for Glintt

After reviewing the available agile methods and analysing the current project management process for ITC projects in Glintt, an agile model is proposed in this chapter. The proposed agile model is mainly based on the Scrum method, but with some adaptations to Glintt's reality. Scrum was chosen because the ITC department workers were already familiarized with the Scrum concepts and terminology. When asked about Agile methodology, Scrum was the first (and only) method that came to their mind. Workers were using the term "Sprint" to refer to short iterations and "Daily Scrum" for every-day status meetings. If the workers are already adapting their processes to Scrum, it is a sign that a Scrum-based model could be well received and implemented. Project teams are usually small, and there are many projects being developed with Outsystems technology (used for rapid application development) – both characteristics are highly compatible with Scrum.

The Scrum method does not include the Agile Project Manager role, however, it was necessary to adapt the method to Glintt's reality. This role is responsible for managing a portfolio of projects and coordinate its Product Owners, ensuring the project is on-time and on-budget. The Product Owner will be responsible for managing project's scope, items' priorities, functional testing, product delivery, and client's expectations.

The reason for this split of role is that Project Managers at Glintt struggle with all their assigned responsibilities, which are now shared between the Agile Project Manager and Product Owners. The Project Managers at Glintt also ensures that the project management process is being applied correctly, which is now a Scrum Master's responsibility. Depending on the project's size, the Scrum Master can have more than one co-located projects assigned since a single person with this role could raise project costs unnecessarily (i.e. the Scrum Master role does not require full availability throughout the entire project).

The Scrum method has Sprints for developing the product, which always have the same structure. In Glintt, depending on the phase of the project, Sprints are different. The proposed model has four types of sprints: Sprint 0, Sprint N, Wash-up Sprint (optional) and Transition Sprint. Each of these sprints is adapted to the project's current phase. Sprint 0 has preparation activities for starting the project, which is followed by Sprint N, where N={1, 2, …, N}. Sprint N is a sprint as described in Scrum, with Sprint Planning, agile development, Daily Scrum, Sprint Review, Sprint Retrospective and Backlog Grooming. The only difference from Sprint N and the Scrum's Sprint is that Sprint N has a "buffer", which is a specific timeframe for bug-fixing and preventing the accumulation of issues in the product backlog. If the buffer is not enough, optionally a Wash-up Sprint may be used if the project is at risk in terms of

quality. This sprint has the same duration of a Sprint N, but it is specific for bug-fixing and taking the project back on track. At the end, before closing, a Transition Sprint takes place, where late corrections and adjustments are made to deliver the product with high quality to the customer.

Other adaptations include the introduction of XP, Kanban, Lean and DSDM practices for the development activities to increase productivity in software development projects. The proposed method is presented in two chapters – Agile Elements and Agile Process.

4.1 Agile Elements

This section presents the agile elements (listed in Table 4.1) used in the proposed agile methodology. The agile elements are divided into five different categories – concepts, artifacts, roles, practices, and events. Finally, the identified roles' responsibilities for each event are presented in Table 4.2.

Table 4.1 – Agile Elements

Concepts	Artefacts	Roles	Practices	Events
• Timebox • Sprint • Budget	• Product Backlog • Sprint Backlog • Definition of Done	• Product Owner • Scrum Master • Project Team • Agile Project Manager • Client	• MoSCoW • Planning Poker • Affinity Estimating • Kanban • Pair Programming	• Sprint Planning • Daily Scrum • Sprint Review • Sprint Retrospective • Backlog Grooming

Concepts

- **Timebox** – Time box consists of limiting the duration of all activities in a project. Generally, the timebox principle is applied to the product planning delivery planning activities, as well as to the meetings held during the project and the sprints.
- **Sprint** – Sprint is a time-boxed iteration which as described in chapter 2.1.1.2, has a duration of 2 to 4 weeks and has the purpose of producing a potentially shippable product increment. In the proposed agile methodology, there are four different types of Sprints: Sprint 0, Sprint N, Wash-up Sprint and Transition Sprint (explained in detail in chapter 4.2).
- **Budget** – Budget defines the planned allocation of resources for the project. This method considers a fixed budget because the ITC projects are budget and time constrained. Once the

project's effort has been defined, it should not be changed. Any arising changes on requirements will result in a re-prioritization of previous requirements and not in a budget increase.

Artefacts

- **Product Backlog** – Dynamic list of all the functionalities that the final product will have, sorted by priority. At the end of each sprint, and taking into consideration the collected feedback, this list is reviewed (Backlog Grooming), and may or may not be changed.
- **Sprint Backlog** – Set of features with higher priority, selected from the product backlog, that will be developed during a specific Sprint. Only Project Team members can update the sprint backlog.
- **User Story** – User Stories are used to describe the business requirements in a way that is simple and easy to understand by all stakeholders. It states the who, what and why for each requirement. Each user story has a list of acceptance criteria which should be met to be accepted by the Product Owner and the Client.
- **Definition of Done** – The Definition of Done, also known as DoD, is the rules that define when any feature can be considered done. (e.g. developed, tested according to acceptance criteria, complies security standards). One backlog item (requirement) is accepted if its functionalities follow the established Definition of Done.

Roles

- **Product Owner** – Responsible for maximizing the value of the product, the Product Owner has the vision of the final product and must pass it on to the team, ensuring that it is aligned with the scope of the project. The Product Owner is the only person responsible for managing the Product Backlog, prioritizing its items, detail, value and visibility. Likewise, it should maximize the work done by the team and ensure the quality of the product. Only this role can make changes to the Product Backlog and ensure that it is accessible to everyone.
- **Scrum Master** – Since the proposed agile methodology is based on the Scrum method, this includes the Scrum Master which plays an important role in the project management process. Responsibilities include facilitating work sessions throughout the project, removing impediments to progress, coaching and ensuring the correct implementation of agile project management processes. The Scrum Master also provides support to the Product Owner in managing the backlog and communicating vision, objectives, and backlog items to team members. However, this role does not make any decision at project management level.
- **Project Team** – A team of multi-disciplinary professionals whose competencies fit the assigned project. Project team members develop the product increments in each sprint (iteration),

according to the sprint backlog. It is self-managed in terms of distribution of tasks and work progress.

- **Agile Project Manager** – The Agile Project Manager is responsible for managing a portfolio of projects, controlling budget, and coordinating Product Owners. Therefore, the team is not entirely self-managed, since the budget is controlled by the Agile Project Manager.
- **Client** – The client is the project stakeholder who is interested in the final product and usually represents the final user. This role is responsible for presenting the business needs to the Product Owner, approving the business requirements specifications, the priorities, and the acceptance criteria for each requirement. The client tests the product and gives the project's official acceptance.

Events

- **Sprint Planning** – Work session where the Product Owner and the Project Team decides what will be delivered at the end of the sprint, and how the sprint goal will be reached. The Product Owner presents the product backlog items to the Project Team to reach a mutual understanding. Team members plan what can be delivered at the end of the sprint based on estimates, ability, and past project experience to set the sprint goal. Sprint planning has a 4 hours time-box.
- **Daily Scrum** – Project team members meet every day at the same location, at the same time, previously set in sprint planning (typically in the morning). During 15 minutes, team members synchronize activities, communicate and raise questions. They share what they have done, what they will do next, and what obstacles are in the way. This session is useful for assessing progress toward the sprint goal. The Scrum Master ensures that Daily Scrum is done and helps to overcome the identified obstacles.
- **Sprint Review** – The Project team demonstrates the product increment to the customer and key stakeholders so they can give their feedback. The goal is to see if what has been developed so far corresponds to what was intended. It also defines what features are made, and what features are not yet made, according to the definition of done. Finally, the product backlog is reviewed and the next steps are defined.
- **Sprint Retrospective** – The Sprint Retrospective takes place after the sprint review. It is a reflection session with the objective of identifying improvements to be introduced in the next iterations / sprints (similar to lessons learned).
- **Backlog Grooming** – The Product Owner, with the collected feedback, reviews the product backlog and decides if changes are made. At the end, the list is always reordered by priority level. Before making any changes to the product backlog, the Product Owner must ensure it is compliant to what was established in the project contract.

Practices

A toolkit folder was created for the company with instructions on how to use the following practices:

- **MoSCoW** – MoSCoW comes from the DSDM process and can be used to prioritize Product Backlog items. The rules are:
    - **M**ust have – fundamental features;
    - **S**hould have – important features, normally mandatory;
    - **C**ould have – features that can be discarded if there is no time to produce them;
    - **W**ould have – features that would be nice to have but are not in the plan.
- **Planning Poker** – Planning Poker is a practice used to estimate User Stories' and features' size, and consequentially the Product Backlog. All the Project Team members have a set of cards with the Fibonacci Sequence numbers. Each team member evaluates the size of each User Story and attributes a number from the Fibonacci Sequence. Once all team members reach a consensus on the User Story size, that is the attributed size. This practice is useful during the Product Backlog creation. This practice can take some time, it is suggested for up to 20 items in each session (Mountain Goat Software, 2017).
- **Affinity Estimating** – For backlogs with more than 20 items, the affinity estimating is a quick way to estimate features' size. The product backlog items should be written in sticky notes so that the team can arrange in a white board. The white board will be a spectrum size from Smaller to Larger, where items are distributed according to its relative size. Then, the items are placed into relative sizing buckets (e.g. XS, S, M, L XL). Some items may not have enough detail to be estimated, and are put aside to be clarified. This practice does not remove the need for a more detailed estimation with Planning Poker (Sterling, 2008).
- **Kanban** – A white board to visually assist and track production. Each functionality has a set of tasks, and those tasks are distributed by four columns (Radigan, 2017):
    - **To Do** – Tasks that were not started;
    - **In Progress** – Tasks that started but are not completed;
    - **Testing** – Tasks that are completed and have yet to be tested;
    - **Done** – Tasks that are completed and successfully tested.
- **Pair Programming** – As mentioned in chapter 2.2.2.1, pair programming is an XP practice used to increase productivity in software development projects. Two people write production code in one machine. While one is writing the code, the other is evaluating if the approach is working, finds a way to make a simple solution, and creates test cases. Pairs may change throughout the work day, which helps to share knowledge through the team about the system (Wells, 1999).

The following Table 4.2 lists the responsibilities of each role on each event within the agile process.

Table 4.2 – Responsabilities Table

Event	Roles	Responsibilities
Create Product Backlog	Product Owner	Create product backlog and prioritize product backlog
	Project Team	Give insights to help prioritizing the product backlog
	Scrum Master	Event moderator and facilitator
Sprint Planning	Product Owner	Create the sprint backlog, write user stories, acceptance criteria, and the Definition of Done
	Project Team	Confirm the capacity to develop the sprint backlog during the sprint, split features into tasks, estimate the execution time of each task, and validate definition of "done"
	Scrum Master	Moderator and facilitator
Sprint Execution	Scrum Master	Facilitator who removes obstacles and provides a safe working environment
	Project Team	Perform tasks in sprint planning, run unit tests, prepare demo
	Product Owner	Functional tests
Daily Scrum	Project Team	Answer questions, define actions to overcome obstacles, make decisions regarding task execution, and decide what tools to use for tasks development
	Scrum Master	Moderator, facilitator; ensures that the daily scrum is practiced, and that the team takes advantage of it
Sprint Review	Product Owner	Experience the product increment, decide what functionalities are considered done, and collect feedback
	Project Team	Demonstrate the product increment
	Client	Test the product increment, and give feedback
	Scrum Master	Moderator
Sprint Retrospective	Project Team	Respond to the sprint retrospective's core issues, and propose improvement actions
	Scrum Master	Moderator e facilitator
Backlog Grooming	Product Owner	If necessary, delete, add or change features and reprioritize product backlog items
	Project Team	Support the Product Owner decision, estimate time required for each feature
	Scrum Master	Moderator e facilitator

4.2 Agile Process

The agile project management process applied to Glintt ITC projects has three main phases – Initiation, Development and Closing – as illustrated in Figure 4.1.

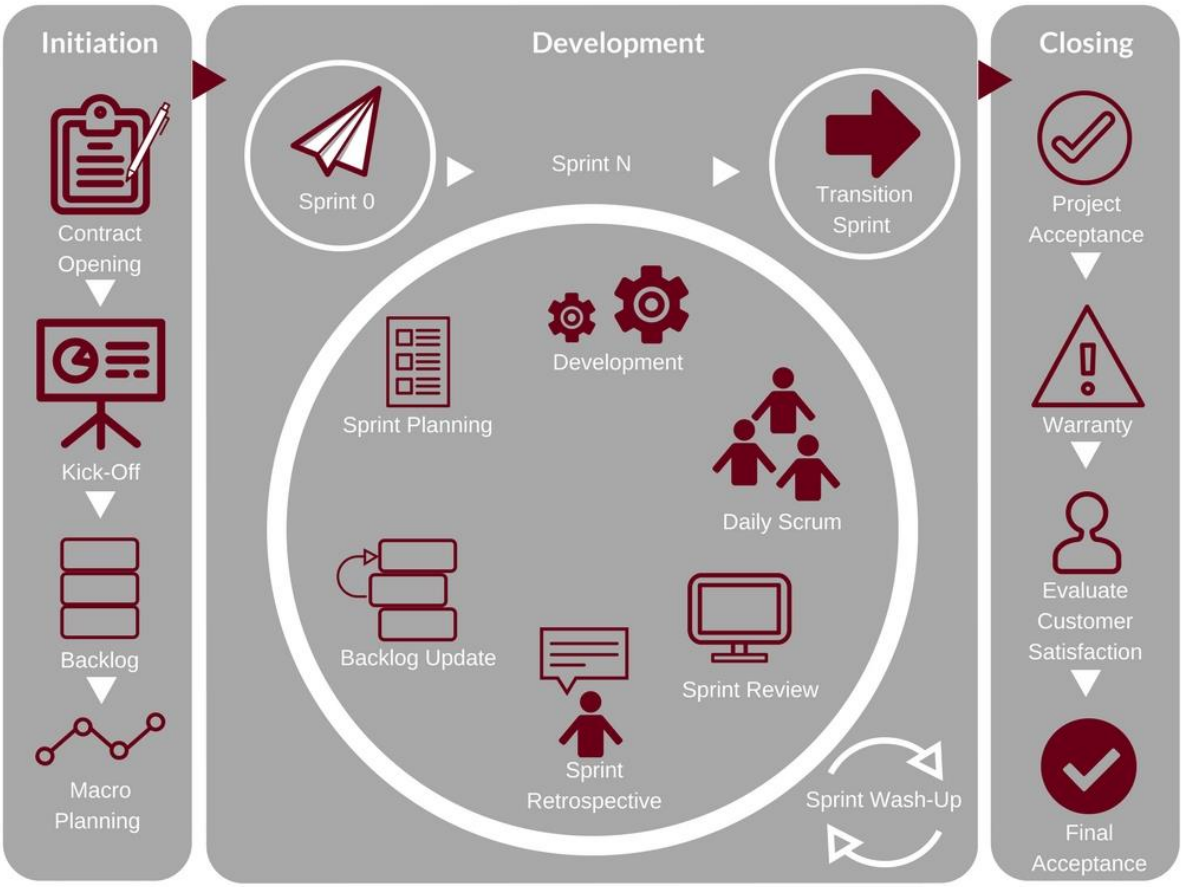


Figure 4.1 – Glintt Agile Project Management Process for ITC Projects

Initiation and Closing phases are similar to the current internal project management standards, due to the nature of ITC projects, with fixed time and scope (to some extent). The main difference is in the Development phase, where agile principles are applied, such as incremental and iterative planning and development, frequent delivery and constant communication.

a) Initiation

The Initiation phase holds the necessary activities for opening a project. The activities are represented in the flowchart of Figure 4.2.

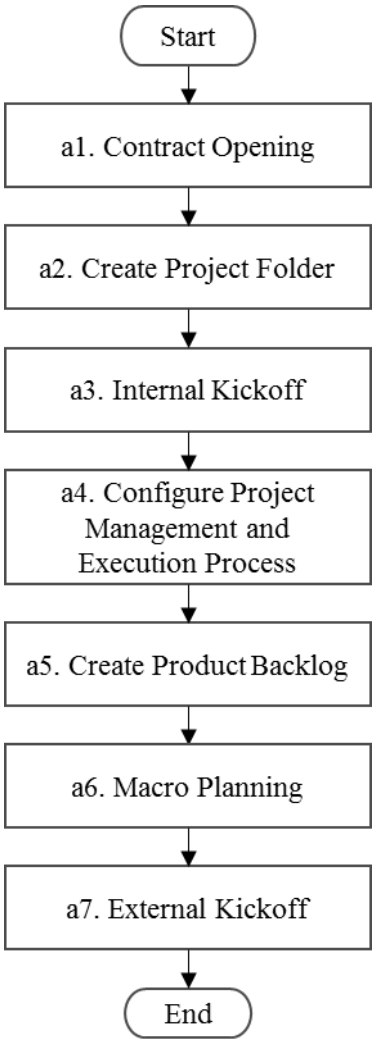


Figure 4.2 – Initiation Process

a1. Contract Opening – The process starts by formalizing the contract, establishing who will be responsible for it and allocating resources for the activities. The contract opening activities are described step-by-step with more detail:

- Fill in and approve the project commitment document – This document formally authorizes the existence of a project and gives the Agile Project Manager permission to allocate organizational resources for the project's activities.
- Contract Register – The CRM software notifies the existence of a new Project Cover resulting from the project's approval (or via Management Control resulting from the creation of an Investment/Research and development). The project cover depicts the different business areas (Products, Services, Support). The approval documentation is attached to the project cover (e-mail, contract, Purchase Order) so that its status changes to "win" in CRM. The project cover

is analyzed in order to verify the project's type of business. The project cover is automatically generated in the G-track system, where the project's financial management is done. The cover may need to be approved by management control department. The "win waiting for approval" status automatically generates the cover on G-track, but is exceptional and must be justified and approved at the top management. After receiving the formalization its status changed to "win", closing the CRM process.

- Designate the person responsible for carrying out the contract – An Agile Project Manager and a Product Owner, who will be responsible for executing the contract, is assigned to the project cover. This is done by the Competence Unit Manager, while the Unit Director, the Client Manager and the Technical Director must be informed.
- Designate the Scrum Master, the Product Owner and the Project Team – Identify the necessary competences for the proper execution of the project and use the g-SM platform to find resources with those competencies. If the Business Unit does not have resources with the necessary competencies, they may be subcontracted internally or externally to Glintt, and the Procurement and Suppliers Management process must be followed.

a2. Create Project Folder – A Project Folder (a directory or folder structure) accommodates all the documentation produced throughout the project. It is created in the intranet of each Business Division. For an agile project, the project folder should have the following structure

Directory Name	**Contents**
DGA0_Inbox	Optional folder to store received documents sent by the client, used in the management process. This folder may be divided into subfolders labelled with respective subject and date.
DGA1_Initiation	Documentation for project's initiation phase. It includes Project Opening Documentation, Agile Process Plans, Product Backlog, Macro Plan and Kick-off presentation.
DGA2_Planning	Detailed project planning documentation for each sprint and for the project as a whole.
DGA3_Execution	Documentation for project's execution. Project execution direction activities: minutes, activity logs, change requests and respective impact analyzes, etc.
DGA4_Control	Project control documentation: budget control, progress reports, etc.
DGA5_Closing	Documentation for phase or project completion. Includes acceptance notes, warranty period procedures, lessons learned from the project, etc.

DGA6_Outbox	Optional folder for storing documentation sent to the client, necessary for the management process. This folder can be divided into sub-folders cataloged by topic and date.
DGA7_Toolkit	Techniques and tools of estimation and management support.

a3. Internal Kickoff – At this stage, the Client Manager will present the project to the Project Team. Moving from the proposal context to the project context presupposes a meeting with proposal team members and Project Team members, if the two are different. Next, if applicable, the Project Team should conduct an internal meeting for initial project set-up (organization, roles and responsibilities, communication mechanisms, tools, etc.).

a4. Configure Project Management and Execution Process – The practices, templates and tools to be used according to the size / complexity / risk of the project, the type of contract and the processes of the Glintt management system should be selected. In ITConsulting, there are three types of configuration sheets:

- Projects of Implementation of Standard Management Solutions – Process configuration sheet
- Software Development Projects – Process configuration sheet
- Agile Software Development Projects – Agile process configuration sheet.

a5. Create Product Backlog – Product Backlog gathers a prioritized list of all product features. It is the only source of requirements (both functional and non-functional). At this stage, a risk response plan is also developed, as this plan may lead to activities that need to be included in the product backlog – because they require effort. Only the Product Owner is fully responsible for the Product Backlog. In addition to creating the entire Product Backlog, you are the only person allowed to make changes to it. With the backlog it should be possible to estimate the total effort for the project, and it is important to involve the development team in this process.Review project's scope and identify requirements to create a Product Backlog. This is a functional analysis exercise, in which the Backlog itens have a high level specification, just enough to do a Macro Plan, aligned with the Vision and Scope of the project (see Project Opening Document available on the site a PT / EN version document, per company). The Product Backlog is created by the Product Owner – according to the corresponding agile process configuration sheet – taking into consideration the Project Team's input. This list is dynamic since it is updated as the product evolves (at the end of each sprint).

a6. Macro Planning – With the product backlog information, the Agile Project Manager, the Product Owner, and the Project Team work together to create a high-level project

schedule and to set the project's baseline. It is necessary to estimate the effort required for the project to make an overall macro plan with the number of sprints, the project completion date, and the budget (see Annex I and Annex II).

a7. External Kickoff – Presentation of project kick-off to the client. This activity is where the project organization, team roles and responsibilities, communication plan, project planning, scope, presentation of deliverables and critical success factors are confirmed with the client. The main decisions taken at this meeting are documented in the Meeting Minutes or in the Kick-off document itself, and the actions taken are monitored at the project follow-up meetings. The roles present in this activity are Product Owner, Client, Agile Project Manager and, optionally, a Project Team representative.

b) Agile Development

The Development phase includes the necessary activities for developing the product. Figure 4.3 displays all the required activities.

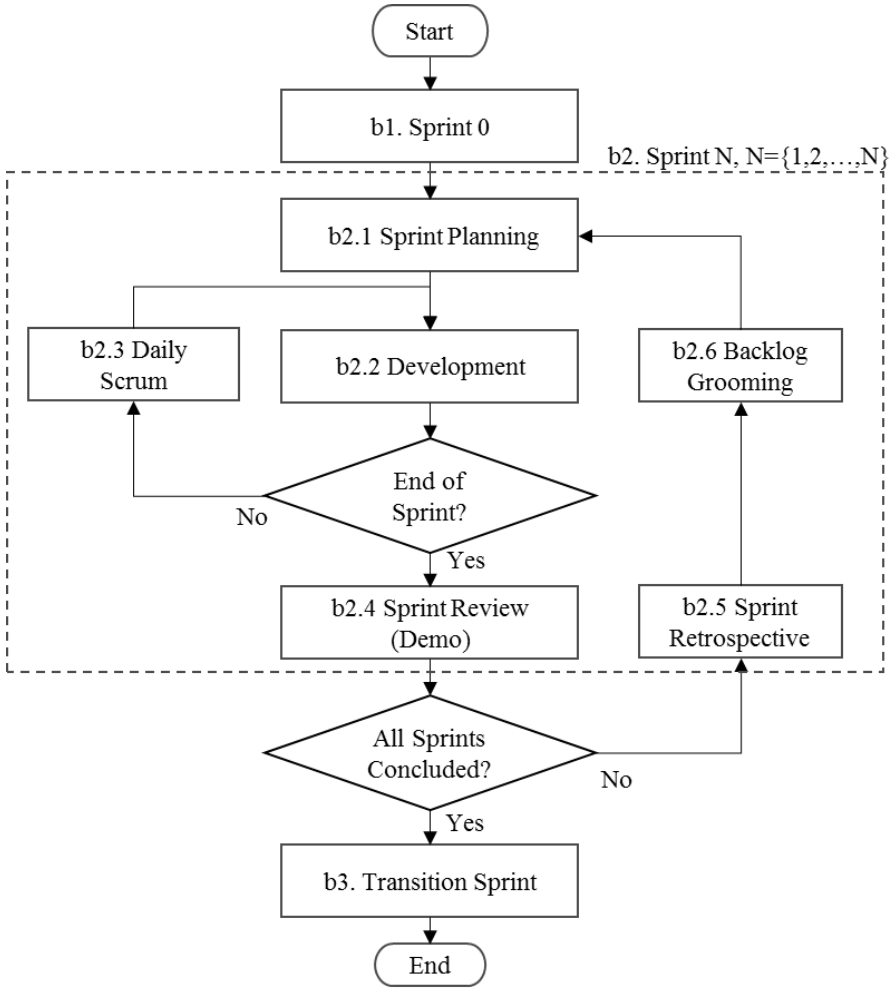


Figure 4.3 – Agile Development Process

b1. Sprint 0 – Sprint 0 is the preparation for Sprint 1. Although Sprint 0 it is different from the other sprints, it is called "sprint" because it has a fixed time-box, just like other sprints (*i.e.* 2 weeks), and accommodates sprint preparatory activities, such as:

- Refine macro planning;
- Update product backlog;
- Plan Sprint 1 (and possibly Sprint 2);
- Specify architecture;
- Specify Sprint 1 requirements;
- Other preparatory activities.

b2. Sprint N – Sprint N is a generic sprint, where N = {1, 2, ..., N}. This sprint has its processes well defined, starting with the sprint planning (or a refining of it), followed by the Development process and Daily Scrum meetings. The sprint ends with a demonstration of new features, which is called a Sprint Review. After the Sprint Review, a Sprint Retrospective takes place which works as lessons learned, and then the backlog is updated before moving to the next Sprint N. During a Sprint N, no changes are made that can affect the purpose of the sprint. However, the scope can be clarified as new information becomes available. The Project Team remains unchanged during the sprint, and all sprints have the same duration. There may be a need to introduce a Sprint Wash-up between certain sprints for bug fixing and refactoring purposes (particularly in software development projects). In this Sprint Wash-up requirements are not developed, as it only serves to solve the product's issues and return to stability. When all the sprints are complete, it is time for the Transition Sprint that will lead to the project closing. All sprints have the same duration. Each process of Sprint N is explained in detail as follows:

b2.1 Sprint Planning – Work session that determines what will be delivered at the end of the sprint and what tasks will be performed to achieve the sprint's goal. It is when the Product Owner selects the set of features that will be developed and presented at the end of the sprint. The artefact resulting from this process is the Sprint Backlog. The Scrum Master is the facilitator of this session. Sprint 1 does not need planning because it has already been done in Sprint 0. Sprint planning activities are:

- Set the Sprint Goal – Team elements in conjunction with the Project Manager and the Product Owner define the purpose of the sprint. The purpose of the sprint contemplates what will be delivered in the final so sprint.
- Create the Sprint Backlog – The Sprint Backlog contains the features that will be developed throughout a sprint to achieve the previously defined goal. This

backlog may have already been created for macro planning, but at this stage it is refined. In the process of selecting features, the Product Owner begins by specifying the user stories of the Product Backlog that are prioritized.

- Planning – The development team defines the tasks required to develop Sprint Backlog items, and estimates the time to complete each. For this, an exercise is performed to decompose functionalities into tasks and, later, decompose larger tasks into sub-tasks. Next, the team makes a plan to accomplish all the tasks identified throughout the sprint.
- Validate – All team members review the plan and confirm they are able to complete Sprint Backlog items during the sprint (time limit set at the initiation phase - 2 to 4 weeks). If necessary, adjustments are made (add or delete features) to ensure that the sprint goal is met.

Good practices for the Sprint Planning activiy are:

- Sprint Planning must have a time-box – typically 4 hours for a 2-week sprint;
- During Sprint Planning, the Product Owner describes the User Stories of the Product Backlog priority. The team asks as many questions as necessary to detail User Stories on smaller tasks that will be done during the sprint;
- Once Sprint Planning is complete, team members have agreed to perform all defined tasks, and Sprint Backlog remains unchanged until the end of the sprint;
- The plan must have a space to correct bugs identified by the client after its tests, called buffer (typically 10% of the sprint timebox).

b2.2 Development – Sprint development gathers the set of activities that will be done to achieve the sprint goal. The development activities are:

- Implementation (code in software development projects)
- Specification refinement of sprint backlog requirements;
- Testing functionalities;
- Audit;
- Buffer;
- Demonstration preparation for sprint review.

The Scrum Master is the facilitator throughout a Sprint and his responsibilities are:

- Remove obstacles so that the team can work properly;
- Facilitate cooperation between different actors;
- Ensure the correct implementation of the agile development process.

The development team may use the following practices and tools to keep track of work and increase productivity:

- Kanban (Whiteboard / Project Dossier – Toolkit)
- During software development, some Extreme Programming practices may be applied, such as:
    - Collective Code Ownership
    - Code Standards
    - Continuous Integration
    - Test-Driven Development
    - Refactoring
    - Simple Design
    - Pair Programming.

b2.3 Daily Scrum – Daily work session lasting 15 minutes. During this meeting, the development team synchronizes activities, communicates and raises issues. It is always held in the same place and at the same time and each member of the development team answers the following questions:

- What have you done since the last meeting?
- What will you do until the next meeting?
- What obstacles are in the way?

The Daily Scrum is an activity to keep track of progress towards the Sprint goal. The Scrum Master ensures that these meetings take place and help remove any identified obstacles.

b2.4 Sprint Review (Demo) – The development team has developed a product with potential to be delivered to the customer (an increment). Any stakeholders interested in the project may attend to the Sprint Review. The product is presented to the Product Owner and Stakeholders who give feedback about the functionalities and help to understand if the product developed so far corresponds to what is intended. At this stage, the client may suggest changes and these will be analysed by the team and by the project manager. The goal of this meeting is to demonstrate the functionalities developed to stakeholders and collect customer feedback to update product backlog and re-prioritize it. This meeting is iterative and moderated by the Scrum Master, who should follow the guidelines below:

- Demonstration – The team gives a real-time demonstration of the features made up to now. Note: "features made" means that they conform to the definition of "done" (developed and properly tested).

- Done Product Backlog Items – The Product Owner declares which features are considered done. To do this, the Product Owner checks to see if the features meet the acceptance criteria agreed upon in Sprint Planning Meeting in the Definition of Done and whether the sprint goal was met. Once considered complete, they are deleted from the Product Backlog.
- Measure Speed – Once the Product Owner considers the items done, they count as speed. The purpose of speed is to keep track of the project's velocity and to support estimations made by the Product Owner.
- Feedback – Finally, items that are to be done or partially done return to the Product Backlog. Stakeholders give their opinion, so that the Product Owner can prioritize the Product Backlog items according to their vision.

b2.5 Sprint Retrospective – This activity works as lessons learned. All team members share their view on what went well and what went wrong, in order to optimize the next sprint. The Scrum Master is the moderator of this meeting Each team member, including the Product Owner, should answer the following questions:

- What went well?
- What can be improved?
- What do we learn?
- What is blocking us?

After answering these questions, participants work together to identify corrective actions for the following sprints. There are good practices for this meeting:

- Organize problematic events in chronological order in a timeline;
- External people are not present in this meeting. Only the development team and the Scrum Master, so that team members feel comfortable talking about any subject and can honestly share their opinion.

b2.6 Backlog Grooming – The Product Owner, with the feedback from Sprint Review, goes through the product backlog to decide if changes should be made. Changes include the procedure of removing features, adding new features with greater value for the business, or just changing existing ones. The change may be in terms of readjusting the expected duration, or setting the scope better. It is also possible to split a large functionality, called Epic, into smaller features and therefore simpler to estimate. At the end, the list is always reordered by priority level. It is possible that the priority assigned to requirements will change as the project progresses. Features with lower business value can still be transferred to subsequent versions of the solution, freeing the project budget for new requirements, which are considered to

be higher priorities. Before making any changes to the product backlog, the Product Owner must ensure that they are compliant to the established scope in the project contract. All members actively participate, however, the Product Owner has authority to make changes to the Product Backlog, and is entirely responsible for it. The backlog grooming should hold the following activities:

- Re-prioritize Product Backlog;
- Pre-select user stories for the next sprint;
- Collect user stories details for the next sprint (Product Owner / Functional Analyst);
- Activity diagrams, wireframes, use cases, data models, sequence diagrams, accessibility tests, business rules, data fields (detail);
- Identify all tests / acceptance criteria for user stories.

One suggested practice is to prioritize by value/time – MoSCow (Project Dossier - Toolkit).

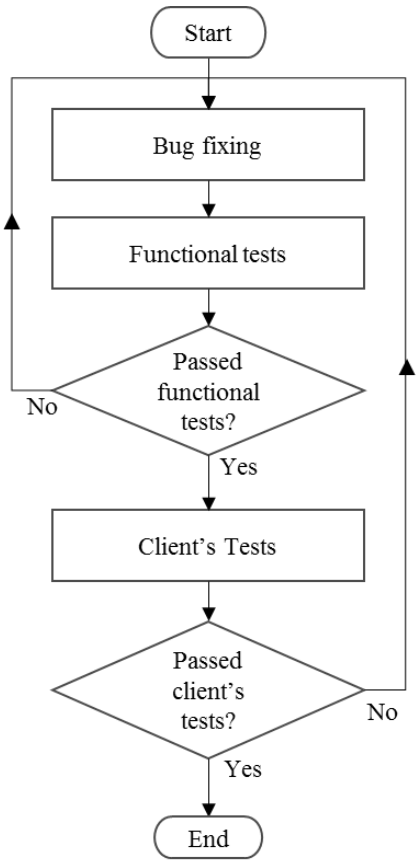


Figure 4.4 – Transition Sprint Workflow

b3. Transition Sprint – Once the N Sprints and Wash-up Sprints have ended, there is a two-week Transition Sprint which will prepare the project for its closure. One or more Transition Sprints may occur, depending on the amount of accumulated issues stored in the product backlog. The Transition Sprint's goal is to make amendments and final adjustments after the key users' tests, for a high quality product delivery (especially if there were no Wash-up Sprints). This sprint's workflow begins with bug fixing, followed by functional tests and client's tests for acceptance, as illustrated in Figure 4.4. Once the product backlog is empty, the closing process takes place.

c) Closing

The Closing phase holds the necessary activities for closing a project. The activities are represented in the flowchart of Figure 4.5.

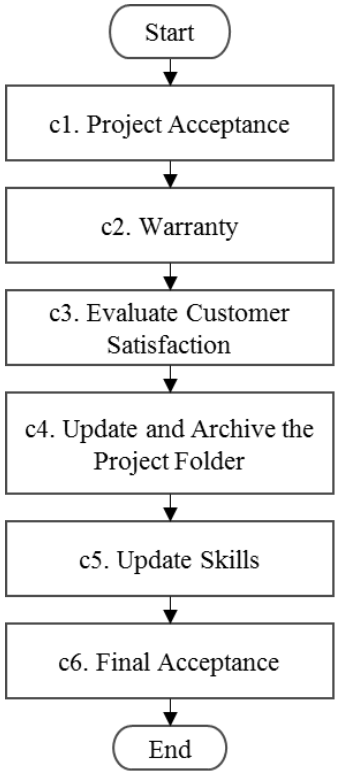


Figure 4.5 – Closing Process

c1. Project Acceptance – The Product Owner obtains the project acceptance (or acceptance of project phases) from the client. These acceptances may be recorded in provisional or definitive acceptance minutes, in e-mail, in Project Sheets, Technical Intervention Files, Acceptance Notice or in project monitoring minutes. The Process Configuration Sheet defines which document to adopt to obtain the final acceptance for each type of project.

When applicable, the provisional acceptance is signed at the beginning of the warranty period and the final acceptance at the end of that period.

c2. Warranty – If applicable, the guarantee period begins with the signature of the provisional or final acceptance minutes, with the system's deploy into production. The warranty period procedures are described in the commercial proposal delivered to the customer. The warranty is ensured by the project management.

c3. Evaluate Customer Satisfaction – At the end of the project, the Agile Project Manager requests the submission of the Customer Satisfaction Assessment Questionnaire and ensures that it is completed by the client. To send the questionnaire to the customer, a request must be made by opening a ticket in the tool g.Support for Quality.

c4. Update and Archive the Project Dossier – The Project Dossier should be updated with the documentation produced throughout the project. Whenever the project has been carried out at the client's premises, a copy of the Project Dossier for Glintt must be secured, which will be stored in the respective Intranet repository.

c5. Update Skills – The team members' CVs are updated as well as the Project References List carried out in the Unit and the Skills Base application (Competency Management Procedure, available in the Management Process of Human Resources). The respective file with the Case Study should be attached to this list. Whenever possible new skills should also be communicated to marketing, for communication purposes (press releases, case studies, etc.).

c6. Final Acceptance – The project is concluded with the official final acceptance.

5 Conclusions

5.1 Conclusions and Limitations

Companies have evolved a lot with technology in the past years and this trend does not seem to slow down. Naturally, this rises competitiveness, and to remain competitive, it is important to have internal processes well-structured and optimized, adapted to the company specific needs. In the IT sector, companies are valued mainly by quality certifications and successful projects. Both aspects are directly related to processes. In one hand, to obtain a quality certifications and accreditations, companies need to have their processes well designed and prove that they are being implemented accordingly. In the other hand, projects' success depends on teams following a well-tailored project management process.

Glintt's ITC department has many software development projects in a variety of technologies – .net, outsystems, IBM BPM – each with specific needs. There is only one internal process defined for project management that is used across all these projects, but some areas ask for a different approach. Agile project management seems to offer the flexibility needed by some technologies, and is already being implemented by some teams, but there is no internal process defined for this. This case study suggested an Agile project management process tailored for the ITC's projects.

An extensive review of agile methods was taken, and Scrum is the most used method. However, there are agile practices that can be merged with this method. It is important to develop processes that are adapted to the company's needs. In this study, Scrum was the basis for the proposed model, but some adaptations were made to better improve the implementation success of the new method. One of the most notorious adaptations made was the introduction of a buffer to support some unpredictability during projects. An optional Sprint Wash-up was suggested for correcting large amounts of identified defects. Another was having the role of Project Manager (which is not specified in Scrum) who has the responsibility of managing a portfolio of projects, controlling budget, and coordinating Product Owners. A ToolKit folder was also created to support the adoption of suggested tools.

The study's main limitation is the absence of concrete evidence on the methods' adoption success since no practical implementation was undertaken.

5.2Contributions

Most projects at Glintt follow a RUP and Waterfall management model, which is well defined in Glintt's internal standards and in which quality certifications are based on (such as ISO9001). However, some projects follow an agile approach, similar to Scrum, because there is the need for an iterative and incremental type of development. But these projects are not following internally defined standards, so the proposed model has contributed for agile standardization and, consequently, help obtaining quality certifications in the future.

One of Glintt's main goal is to obtain the CMMI level 3 certification, which indicates that the company has a level of maturity where processes are well defined and followed by all workers. To achieve this, there is a set of criteria that must be fulfilled. One of the criteria is that the company must have an Agile methodology for software development so, this way, the proposed agile model has contributed for the process of obtaining the CMMI level 3 certification.

Every year, Glintt opens an academy to hire and train new employees, usually recently graduated. During the 2016 academy, the proposed model was presented to the new workers as part of its implementation roadmap. During 2017, Glintt also offered Agile training for Scrum Fundamentals and Certified Scrum Product Owner to 21 of its workers. The proposed model is compliant with CMMI and is aligned with the training and certifications.

5.3 Future Work

Publishing and implementing the proposed model in Glintt's projects while measuring results and making appropriate adjustments, is proposed as future work in order to overcome the limitation described previously. As suggested by Hoda & Noble (2017), agile implementation should be done progressively (Hoda & Noble, 2017) – starting with small pilot projects and making changes in some parts of the process. The proposed process for project management must be reviewed to guarantee CMMI compliance. Then, once the process has been standardized and workers have achieved maturity in implementing it, concrete evidence should be gathered to confirm that Glintt is able to apply for quality standards certification and CMMI accreditation.

As future work it would be interesting to compare results of similar projects where some were not following the proposed method and others were following it, to assess if the proposed model has brought benefits for the company's needs.

6 Bibliographic References

Abbas, N., Gravell, A. M., & Wills, G. B. (2008). Historical roots of agile methods: Where did "Agile thinking" come from? *Lecture Notes in Business Information Processing*, *9 LNBIP*, 94–103. https://doi.org/10.1007/978-3-540-68255-4_10

Abdullah, E., & Abdelsatir, E. T. B. (2013). Extreme programming applied in a large-scale distributed system. *Proceedings - 2013 International Conference on Computer, Electrical and Electronics Engineering: "Research Makes a Difference", ICCEEE 2013*, 442–446. https://doi.org/10.1109/ICCEEE.2013.6633979

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods review and analysis. *VTT Publications*, (478), 3–107. https://doi.org/10.1076/csed.12.3.167.8613

Abrahamsson, P., Warsta, J., Siponen, M. T. M. T., Ronkainen, J., & Ronkanen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. *Software Engineering, 2003. Proceedings. 25 International*, *6*, 244–254. https://doi.org/10.1109/ICSE.2003.1201204

Anderson, D., Augustine, S., Avery, Christopher Cockburn, A., Cohn, M., DeCarlo, D., Fitzgerald, D., … Wysocki, R. (2005). Declaration of Interdependence. Retrieved from www.pmdoi.org

Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Technology Organizations*. Blue Hole Press.

Beck, K. (1999a). Embracing change with extreme programming. *Computer*, *32*(10), 70–77. https://doi.org/10.1109/2.796139

Beck, K. (1999b). *Extreme Programming Explained: Embrace Change*. *XP Series*. https://doi.org/10.1136/adc.2005.076794

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Agile Manifesto. Retrieved July 20, 2017, from www.agilemanifesto.org

Boehm, B., & Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE Software*. https://doi.org/10.1109/MS.2005.129

Borth, M. R., & Shishido, H. Y. (2013). A Comparative Analysis of Two Software Development Methodologies: Rational Unified Process and Extreme Programming. *Revista Vértices*, *15*(3), 143–157. https://doi.org/10.5935/1809-2667.20130035

Carvalho, B., & Mello, C. (2012). Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequenaempresa de base tecnológica. *Gestão & Produção*, *19*(3), 557–573.

Coad, P., de Luca, J., & Lefebvre, E. (1999). *Java Modeling In Color With UML: Enterprise Components and Process*. Pearson PTR.

Cockburn, A. (2002). *Agile Software Development*. Addison-Wesley Professional.

Cohn, M. (2010). *Succeeding With Agile*.

Diaz, J., Garbajosa, J., & Calvo-Manzano, J. A. (2009). Mapping CMMI Level 2 to Scrum Practices: An Experience Report. *Communications in Computer and Information Science*, *42*, 93–104.

DSDM Consortium. (2014). DSDM Atern Handbook. Retrieved from http://dsdm.org/dig-deeper/book/dsdm-atern-handbook

Edeki, C. (2013). Agile Unified Process. *International Journal of Computer Science and Mobile Applications*, *1*(3), 13–17.

Ferreira, J., Sharp, H., & Robinson, H. (2011). User experience design and agile development: Managing cooperation through articulation work. *Software - Practice and Experience*, *41*(9), 973–974.

Fetouh, A. A., Abbassy, A. el, & Moawad, R. (2011). Applying Agile Approach in ERP Implementation. *IJCSNS International Journal of Computer Science and Network Security*, *11*(8), 173–178.

Gannon, M. (2013). An agile implementation of SCRUM. *IEEE Aerospace Conference Proceedings*, 1–7. https://doi.org/10.1109/AERO.2013.6497388

Glazer, H., Dalton, J., Anderson, D., Konrad, M., & Shrum, S. (2008). *CMMI ® or Agile : Why Not Embrace Both ! (CMU/SEI-2008-TN-003). Software Engineering Institut*. https://doi.org/10.1109/AGILE.2006.30

Griffiths, M. (2012). *PMI-ACP Examp Prep*.

Highsmith, J. (2000). *Adaptive Software Development: a collaborative approach to managing complex systems*. New York: Dorset House Publishing Co.

Highsmith, J. (2001). History: The Agile Manifesto. Retrieved March 11, 2017, from http://agilemanifesto.org/history.html

Highsmith, J. (2002). *Agile Software Development Ecosystems*. Indianapolis: Addison-Wesley.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*. https://doi.org/10.1109/2.947100

Hoda, R., & Noble, J. (2017). Becoming Agile: A Grounded Theory of Agile Transitions in Practice. *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017*, 141–151. https://doi.org/10.1109/ICSE.2017.21

International Telecomunication Union. (2015). *ICT Facts & Figures. The world in 2015*. Retrieved from http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2015.pdf

ISO. (2009). ISO 9001:2015 Quality management systems -- Requirements. Retrieved August 26, 2017, from https://www.iso.org/standard/62085.html

Kruchten, P. (2000). *The Rational Unified Process: An Introduction* (2nd ed.). Massachusetts: Addison Wesley Longman.

Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., & Ståhl, D. (2013). The impact of agile principles and practices on large- scale software development projects. *IEEE International Symposium on Empirical Software Engineering and Measurement*, 348–356. https://doi.org/10.1109/ESEM.2013.53

Mountain Goat Software. (2017). Planning Poker. Retrieved July 20, 2017, from https://www.mountaingoatsoftware.com/agile/planning-poker

O'Reilly, T. (1999). Lessons from open-source software development. *Communications of the ACM*, *42*(4), 33–37. https://doi.org/10.1145/299157.299164

Oliveira, P. M. (2017). Ranking: As maiores empresas da tecnologia em Portugal. *Exame Informática*. Retrieved from http://leitor.exameinformatica.pt/#library/exameinformatica/28-10-2017/edicao-38/reportagem/listagem-ranking-2016

Oxford English Dictionary. (2015). Oxford English Dictionary Online. Retrieved from http://dictionary.oed.com

Palmer, S. R., & Felsing, M. (2002). *A Practical Guide to Feature Driven Development*. *... Guide to Feature Driven Development*.

Pordata. (2017). Gross Domestic Product (Euro). Retrieved September 10, 2017, from https://www.pordata.pt/en/Europe/Gross+Domestic+Product+(Euro)-1786

Project Management Institute. (2013). *Software Extension to the PMBOK Guide Fifth Edition* (5th Editio).

Radigan, D. (2017). Kanban: How the kanban methodology applies to software development. Retrieved July 20, 2017, from https://www.atlassian.com/agile/kanban

Rigby, B. D., & Bilodeau, B. (2015). *Management Tools & Trends 2015*. *Bain and Company, Inc.*

Rubin, K. S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process* (1st ed.). New Jersey: Addison-Wesley Professional.

Santos, D., & Cordová, P. (n.d.). Combinação de métodos ágeis no processo de desenvolvimento de software: um estudo de caso, 6–23.

Schwaber, K. (1995). Scrum development process. *Proceedings of the Workshop on Business …*, (April 1987), 10–19. https://doi.org/10.1007/978-1-4471-0947-1_11

Schwaber, K. (2004). *Agile Project Management with Scrum*. *Microsoft Press* (Vol. 7). https://doi.org/10.1201/9781420084191-c2

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum*. *cdswebcernch* (Vol. 18). https://doi.org/10.1109/2.947100

Serrador, P., & Pinto, J. K. (2015). Does Agile work? - A quantitative analysis of agile project success. *International Journal of Project Management*, *33*(5), 1040–1051. https://doi.org/10.1016/j.ijproman.2015.01.006

Stapleton, J. (1997). *DSDM, Dynamic Systems Development Method: The Method in Practice*. *DSDM Dynamic Systems Development Method The Method in Practice*.

Sterling, C. (2008). AFFINITY ESTIMATING: A HOW-TO. Retrieved July 21, 2017, from http://www.gettingagile.com/2008/07/04/affinity-estimating-a-how-to/

Takeuchi, H., & Nonaka, I. (1986). The New New Product Development Game. *Harvard Business Review*, *64*(1), 137–146. https://doi.org/10.1016/0737-6782(86)90053-6

The Standish Group. (2013). *CHAOS MANIFESTO 2013: Think Big, Act Small*. *The Standish Group International*. Retrieved from http://www.standishgroup.com

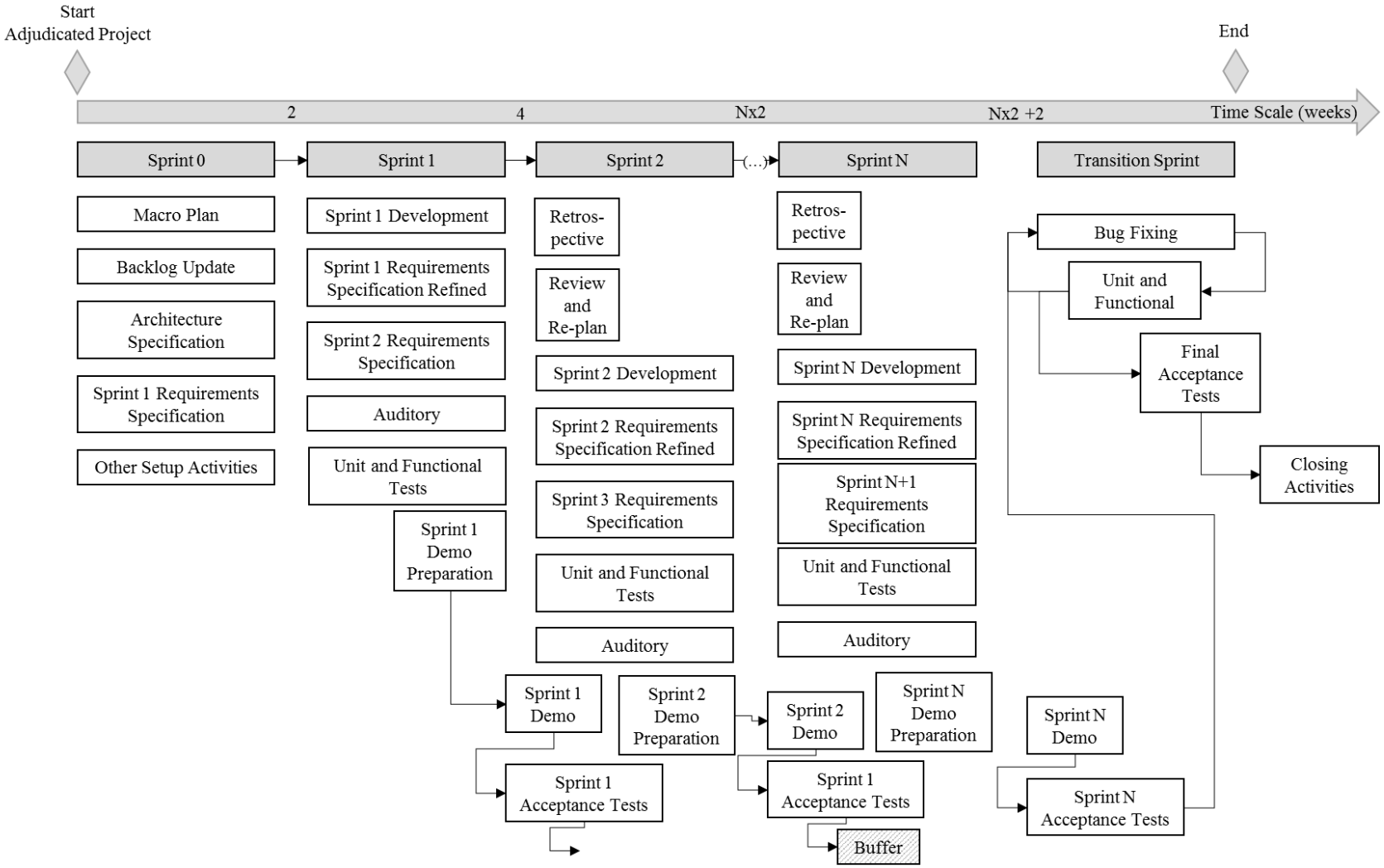
United Nations. (2015). *World Economic Situation and Prospects 2015*. New York.

Version One. (2015). *9th State Of Agile Survey*. *9th Annual* (Vol. 54). https://doi.org/10.2777/17146

Version One. (2016). *The 10th Annual State of Agile Report*.

Wells, D. (1999). Pair Programming. Retrieved July 20, 2017, from http://www.extremeprogramming.org/rules/pair.html

Annex I – Agile Macro Planning I



Annex II – Agile Macro Planning II