MESG

MESTRADO EM ENGENHARIA DE SERVIÇOS E GESTÃO

Developing software as a medical device with an Agile methodology

João Miguel Amaro dos Santos Martin Afonso

Master Thesis

Supervisor at FEUP: Professor Jorge Grenha Teixeira

Supervisor at Glintt - Healthcare Solutions, SA: Engenheiro Tiago Amaral

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

09-2020

à minha mãe, pela força
ao meu pai, pelo exemplo,
ao meu tio, pela dedicação
à minha avó, pela memória
aos antigos
aos de sempre, para sempre
aos novos

Abstract

The recent Medical Device Regulation 2017/745/EEC applies new definitions to software as a medical device, in which software can be classified as a legitimate medical device. This European regulation, together with international standards, define the base for developing software as a medical device. Specifically, the IEC 62304:2006 defines the software development life cycle and the requirements needed for software compliance.

Further, with the growing importance of software in the healthcare industry, methodologies and frameworks can be applied to improve the entire development process and the quality of the final product. Agile methodologies create the conditions and the environment around the development of software, helping the development teams to focus on incremental and quick delivery.

Responding to strict and regulated guidelines towards medical device software and integrate them into an agile environment can bring some challenges. Specifically, when trying to create a software process that responds to several different requirements imposed by the IEC 62304, from risk management to a problem resolution process.

Thus, the main goal of this work is to define a life cycle model for the software development process, based on an agile approach, ensuring compliance with the standards defined by the IEC 62304.

The most crucial methods and activities required to provide compliance as a medical device will be applied in a seamlessly agile experience. Besides applying agile principles, the defined life cycle model leverages agile characteristics to implement a different approach to quality assurance, implements risk management and problem resolution processes and creates a well-documented development environment to address safety critical software.

Resumo

A nova Regulação para Dispositivos Médicos 2017/745/EEC aplicou novas definições em software, sendo que o próprio software pode agora ser classificado como dispositivo médico. Esta regulação Europeia, juntamente com *standards* internacionais, definem o ponto de partida do desenvolvimento de software com este fim. Especialmente, a IEC 62304:2006 define o ciclo de vida do desenvolvimento de software e os requisitos necessários para o software atingir conformidade como dispositivo médico.

Adicionalmente, com a crescente importância de software na indústria de saúde, diversas metodologias e *frameworks* podem ser aplicadas para melhorar o processo de desenvolvimento e a qualidade do produto final. Metodologias *agile* criam as condições e um ambiente de desenvolvimento de software propicio a ajudar as equipas de desenvolvimento a focarem-se em entregas rápidas e incrementais.

Responder a regulações rígidas relativas à classificação de softwares como dispositivos médicos e integrá-las num ambiente *agile* pode trazer alguns desafios, nomeadamente a tentar criar um processo de desenvolvimento de software que responda aos diferentes requerimentos impostos pela IEC 62304, desde gestão de risco ao processo de resolução de problemas.

Portanto, o principal objetivo deste trabalho é definir um modelo de ciclo de vida para o processo de desenvolvimento de software, baseado numa abordagem *agile*, garantindo conformidade com os *standards* definidos pela IEC 62304.

Os métodos e atividades necessários para garantir que o software seja classificado como dispositivo médico serão integrados numa experiência *agile*. Para além de aplicar princípios ágeis, o modelo de ciclo de vida definido potencia características presentes nas metodologias *agile*, de maneira a mudar o paradigma relativo ao controlo de qualidade, a implementar processos de gestão de risco e resolução de problemas, e a criar um ambiente de desenvolvimento fortemente documentado. Todas estas alterações são especialmente conceptualizadas para software critico a nível de segurança.

Acknowledgments

First, I want to address my deepest appreciation to my supervisor, Professor Jorge Teixeira, for encouraging me, always saying the right word, and helping me to gain the motivation needed for the difficult development of this project.

With Glintt and all the people that have crossed paths with me, I will always be deeply grateful. For creating the perfect conditions to attend the Master's, for allowing me to have a space to develop this thesis and for accompanying me in my professional and personal growth. I want to thank, especially, to Francisco Correia and Miguel Henriques for, in a very early stage of this work, have demonstrated immediate willingness to create the conditions necessary for its development. A special thanks also to Tiago Amaral, my Glintt supervisor, for helping me direct this work in the right direction and for the knowledge transmitted, inside and outside of the project.

I would like to express my gratitude to my family, for promptly supporting any decision, with encouragement and wisdom. And for never let me go down.

To all my friends that directly or indirectly influenced me, not only during this project but for many years now, through tears and laughs.

This win is shared with all these people.

Table of Contents

List of Tables

List of Figures

List of abbreviations

MDR - Medical Devices Regulation

EU – European Union

SaMD - Software as a Medical Device

IMDRF - International Medical Device Regulators Forum

CE – Conformité Européenne

MDF - Medical Device Framework

ISO - International Organization for Standardization

IEC - International Electrotechnical Commission

QMS – Quality Management System

FDA - U.S Food and Drug Administration

QA - Quality Assurance

DoD - Definition of Done

UI – User Interface

PR - Pull Request

1 Introduction

In recent years, there has been an incremental need in developing software capable of providing different responses across the healthcare sector, due to the increasingly high importance of software included in medical devices. From 2002-2010 software-based medical devices affected more than 1,527,311 devices (Fu, 2011).

This growth, both in number as in responsibility, has increased the need of developing trustworthy software. The software in the healthcare industry can directly affect, both positively and negatively, the patient, since this is used to perform a critical assessment, like the glucose levels of a person with diabetes. In this context, the software developed to give a response to such needs must have an exceptionally high degree of safety and effectiveness (Fu, 2011).

Recently, in the year of 2017, the Medical Device Regulation (MDR) 2017/745/EEC entered into force, changing the previous directive 2007/47/EC. This directive, in 2007, introduced the new definition of the standalone software that can be considered a medical device, without being included in an electronic device. With the new regulation in 2017, this definition was extended to include prediction and forecasting devices. With this regulation in force, software that can be classified into this category is gaining more power, also increasing the need to verify its quality.

Furthermore, it is necessary to subject the development process to certification and regulatory testing. This need sets the base and primary motivation for developing this work. To get this certification, strict guidelines and standards, following the European Union (EU) normative, must be met, which can lead to a considerable period where the software is verified or not as a medical device.

A critical issue when addressing this topic of software performing as a medical device is that many medical devices are embedded systems, making software a fundamental part of it. Having this in mind, to apply the regulatory requirements previously addressed, it is necessary to define rigorous software development methods that can ensure reliability and protect public health. For achieving this goal, testing and requirements have a significant influence (I. Lee et al., 2006).

Thus, due to these constraints, the goal is to develop a life cycle model that encompasses what are the requirements of software as a medical device, including them into a software development process.

1.1 Problem Description

All the software developed in a healthcare environment, have the mission and objective to ancillary the healthcare professionals in all their day to day activities and increase their efficiency at the highest possible level. Focused in these need to increase the efficiency, software in different areas, such as patient admission, patient prescription, and clinical trials, is being developed, and most of it is already having a significant impact in numerous hospitals and clinics, improving the processes and inherent tasks of healthcare professionals.

Having this background in mind, it is essential to think about the problem from two distinct, but complementary, points of view.

The first one is the methodologies that have been emerging in the last few decades. These methodologies define approaches for the software development life cycle and can play a critical factor in the quality of the software being delivered. With the increased software complexity and dynamic user requirements, agile methodologies are gaining more importance, setting a change

from traditional software development models to agile based development. These agile methodologies are characterized for having incremental deliveries, are more suitable to handle requirements changes and have shorter development cycles (Matharu, Mishra, Singh, & Upadhyay, 2015).

The other point of view is the classification of the software as a medical device (SaMD). This term is defined by the International Medical Device Regulators Forum (IMDRF) as “software intended to be used for one or more medical purposes that perform these purposes without being part of a hardware medical device." (Group, 2013).

To classify a software as a medical device, several norms and regulations must be followed and be aligned with standards for different stages in the life cycle development of medical device software.

Hence, the problem is the correlation between the used methodologies during the process of developing healthcare software and if they provide the needed documentation and safety assurance required to classify a SaMD. To answer this problematic, a software development life cycle model will be developed, to integrate all the needed requirements.

As such, this study research questions are the following:

- RQ1 - How can agile development methodologies be incorporated into healthcare software development?
- RQ2 - How can agile development methodologies improve the process of classifying software as a medical device?

1.2 Study and Project Development at Glintt

This project was developed at Glintt, a healthcare-focused company, which develops software for the most varied clinical areas.

The project was carried out for seven months, from February to September. During this time, the initial focus was a theoretical approach towards software as a medical device and a period researching European regulation and international standards, to understand the normative foundations of the work.

Subsequently, a practical approach towards software practices and agile methodologies was taken, working on defining a model comply with the strict and regulated requirements of international standards.

1.3 Report outline

This dissertation is divided into seven chapters, addressing the following topics: a brief introduction to the subject of interest, the literature approached to this work, the problem characterization, the methodology used and the data collection method, the data resulting from the interviews, the proposed solution and finally the conclusions.

In the first chapter, an early introduction to the subject is made, characterizing the project and the problem, and explaining what the main motivations are to perform this work. Also, the research questions are defined, serving as a basis for the following work.

In the second chapter, is performed a review of the existing literature regarding software as a medical device and software development processes critical for including in the proposed model.

Also, in this chapter is approached the evolution of methodologies in software development and specifically agile methodologies in safety-critical environments.

The third chapter presents a problem characterization, at the light of the topics addressed in the literature review. There, are shown the main problems and difficulties of complying software as a medical device. The points raised in this chapter work also as a motivation to find a solution capable of facing these problems.

The fourth chapter is related to the methodology used to collect data, which will define some of the foundations used in the final solution.

The fifth chapter depicts the data collected in the interviews. This data is divided into several major topics of interest and, in each case the results are coded according to the subjects addressed. This way a more precise explanation of concepts is achieved, and the interview results can be understood at the light of the issue of work of this dissertation.

The sixth chapter presents the proposed solution to overcome the problems addressed in the third chapter. The solution is presented in the form of a development life cycle model, that aims to integrate several different software development practices and activities into an agile methodology. There are detailed all these activities and how they help to achieve compliance with medical devices regulations.

The final chapter details the conclusions drawn from this work and identifies the opportunities for future work.

2 Literature Review

2.1 Software as a Medical Device

Until the end of the ’90s, there was no specific regulation to control the different approaches in regulating devices and each member state had its own.

Thus, to overcome this obstacle and to promote the internal market in Europe, the European Council introduced new regulations known as the “new approach directives”. These directives defined the essential requirements needed to guarantee the safety and performance of the device and were applied to all countries. This way, if a device receives a Conformité Européenne (CE) mark in one member state it is automatically verified in all member states. This mark certifies device safety and ensures that it is functioning according to the manufacturer's purpose (Sorenson & Drummond, 2014).

Therefore, for applying these regulations the Medical Device Framework (MDF), has the primary objective of providing common rules for the free movement of medical devices throughout the EU while ensuring the safety of usage by all the EU citizens. To achieve this a set of basic safety requirements have been defined. Until 2017, the MDF was mainly composed by three different directives: Council Directive 90/385/EEC concerning active implementable medical devices; Council Directive 93/42/EEC concerning medical devices and Council Directive 98/79/EC concerning in vitro diagnostic medical devices (Quinn, 2017).

In 2007 the 2007/47/EC directive emended these three directives. Changes are related, mainly, to the essential requirements that the medical devices must satisfy before being placed on the market, as well as the conformity assessment procedures and the classification of devices. Article 2 of Directive 93/42/EEC states that “Member States shall take all necessary steps to ensure that devices may be placed on the market and/or put into service only if they comply with the requirements laid down in this Directive (…)” (Commission, 2009).

This sets the objectives for a device be placed on the market.

This MDF is not only applied to physical devices. It can be used to software also. This software, classified as a medical device, can have different variations. It can be a component of a larger medical device or can be a medical device by itself, called “stand alone software”. For software to be classified as a medical device, it must be complying with all the MDF’s requirements (Quinn, 2017).

Consequently, this introduction of software on medical devices, to complete their propose, started to impose questions about whether and when the software is considered a “medical device”. A different variety of items have been raised about this possible classification of the software as a medical device, such as “Can the software be used by itself, or only together with another medical device?” and “Is the software intended for users with specific medical conditions?” (Fillmore, 2019).

To answer these questions, it is necessary to understand the criteria for the qualification of this type of software, under the Regulation (EU) 2017/745 – Medical Devices Regulation defined by the European Parliament on April 5, 2017 (Parliament, 2017).

Due to the Covid-19 crisis, the European Council and the Parliament are adopting the Regulation 2020/561 amending Regulation (EU) 2017/745 on medical devices. The objective of this new Regulation is to postpone the date of application for most MDR by one year, meaning that it will be applied from May 2021, so that the healthcare practitioners and manufacturers can focus

entirely on any pandemic related priority. As this amendment is only associated with the dates of application, for the scope of this project the Regulation (EU) 2017/745 will serve as a starting point to all criteria related to MDR (Parliament, 2020).

This medical device regulation was constructed to improve the 90/385/EEC and 93/42/EEC directives and their emends. This regulation has a transition period of three years to be implemented and introduce some significant changes when compared with the 2007/47/EC directive. First of all, the new regulation has expanded the definition of the term "medical device", to include products that can ensure the prediction and prognosis of diseases, and also those products that do not have a direct medical intent. The 2017 directive will allow the reclassification of some categories of devices to class III and will have more strict designation requirements and roles. Regarding the classification for class III (high-risk medical devices), it will encompass more rigorous procedures to achieve this classification. In conclusion, this medical device regulation will present solutions to previous gaps, namely because of the expanded definitions and reclassifications of devices, due to the technological evolution (Migliore, 2017).

Therefore, it is essential to clarify the definition of 'medical device'. According to article 2 of the Regulation (EU) 2017/745, a 'medical device': "means any instrument, apparatus, appliance, software, implant, reagent, material or other article intended by the manufacturer to be used, alone or in combination, for human beings for one or more of the following specific medical purposes (…)" and has the 'intended purpose' of: *"*use for which a device is intended according to the data supplied by the manufacturer on the label, in the instructions for use or promotional or sales materials or statements and as specified by the manufacturer in the clinical evaluation" (Parliament, 2017).

To classify a medical device, it is necessary to understand its intended purpose and inherent risks. The low risk devices are classified as class I, medium risk devices as class IIa or IIb, and the ones with the higher risk as class III. These classifications are further explained in table 1.

Table 1 - Classification of medical devices according to Regulation (EU) 2017/745 (Parliament, 2017)

Classification	Description
Class I	Typically, simple devices in terms of functionalities and design. There presents no risk to human health, with a history of safe use. They are non-invasive devices.
Class IIa	Present low risk to human health.
Class IIb	Present a higher risk to human health compared with Class IIb.
Class III	This type of device is very invasive and can present a severe risk of illness or injury. Usually, they are used to support or sustain human life.

Software as a Medical Device is defined, according to the International Medical Device Regulators Forum, as "software intended to be used for more than one or more medical purpose that performs these purposes without being part of a hardware medical device". SaMD may be used in combination with other products, including medical devices. Furthermore, it can be interfaced with other medical devices, such as hardware medical devices and other SaMD software (Group, 2013).

2.2 IEC 62304 and its applicability when developing a SaMD

To obtain the classification of medical device, the software must comply not only with the European regulations, addressed in the previous section, but also with international standards. For that ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) work together to develop standards and guides on conformity assessment.

Starting to analyze the standards relevant for the topic, two of them are highlighted. It is essential for medical device manufactures to have into consideration the ISO 9001:2015 (Standardization, 2015) and the ISO 13485:2016 (Standardization, 2016). The first one defines the requirements needed for a quality management system (QMS) and the second one can be considered an expansion of it, addressing the same topic (Geremia, 2018).

Besides a quality management system, another critical aspect must be taken into consideration, the risk management compliance. Addressing this topic, the ISO 14971:2019 (Standardization, 2019) is the international standard that should be followed.

For this topic of interest and as the main objective of this work is to achieve medical device compliance through software developing process, the core international standard that must be studied is the IEC 62304:2016 (Standardization, 2006). According to the IMDRF, this standard "is a European harmonized standard, which provides presumption of conformity with legal requirements on development life cycles for software which are incorporated in medical devices and software which are medical devices in themselves". Using this standard when analyzing and design the development life cycle provide compliance with the legal requirements necessary to have a software categorized as a medical device (Committee, 2015).

Although the IEC 62304 defines the software life cycle processes needed to develop software for medical devices, it is more detailed regarding the objectives that can be achieved by a specific process activity. This way, it does not restrict or name specific software engineering methods and techniques suitable for performing the desire goals (Huhn & Zechner, 2010).

The relevance of the present work is validated by this availability to tailor the software process and methods accordingly.

This standard defines life cycle-processes for medical devices software. Each one of these processes includes key activities and tasks. To determine the rigor needed in the quality assurance of the software, this standard follows a risk-based approach. This rigor is inferred through the risk of the medical device itself, which can vary from the three classes of risk (Huhn & Zechner, 2010).

Succinctly, the classes are described in table 2.

Table 2 - Risk classes according to IEC 62304

Class	Description
Class A	No injury or damage to health possible
Class B	Nonserious injury is possible
Class C	Death or serious injury is possible

As previously stated, the IEC 62304:2006 provides software life cycle processes. These processes are divided into five major groups:

- The development process: For the primary process, the standard defines the creation of a development planning document, where all the activities involved in the development process are listed. This document must also contain a system safety classification.
- The maintenance process: After customer feedback, it must be documented all the changes that are applied to the software.
- The risk management process: A process for risk management should be developed, according to the ISO 14971:2019.
- The configuration management process: Here are addressed all the items regarding versioning and tracing of the software.
- The problem resolution process: For each identified problem, a specific report with strategies to overcome the problem must be developed (Zamith & Gonçalves, 2018).

The processes and each sub-task are detailed in table 3.

Table 3 - Description of each process according to IEC 62304

Number	**Description**
Part 5	**Software development process**
5.1	Development planning
5.2	Requirements analysis
5.3	Architectural design
5.4	Detailed design
5.5	Unit implementation and verification
5.6	Integration and integration testing
5.7	System testing
5.8	Release
Part 6	**Software maintenance process**
6.1	Establish software maintenance plan
6.2	Problem and modification analysis
6.3	Implementation of modifications
Part 7	**Software risk management process**
7.1	Analysis of software contributing to hazardous situations
7.2	Risk control measures
7.3	Verification of risk control measures
7.4	Risk management of software changes
Part 8	**Software configuration management process**
8.1	Configuration identification

8.2	Change control
8.3	Configuration status accounting
Part 9	**Software Problem Resolution Process**
9.1	Prepare problem reports
9.2	Investigate the problem
9.3	Advise internal parties
9.4	Use change control process
9.5	Maintain records
9.6	Analyze problems for trends
9.7	Verify software problem resolution
9.8	Test documentation contents

When conceptualizing a new medical software, that aims compliance and medical device classification, the first aspect that must be considered is its classification. As seen in table 2, there are different classifications for the software according to its risk.

Therefore, an initial assessment of the software risk that is going to be developed is needed. This means that each software can have a different classification. According to the risk degree of the software, several IEC 62304 requirements must be fulfilled. Appendix C shows the requirements necessary to obtain each risk classification. As class C presents the higher risk for the healthcare user, any software that desires to obtain C classification must ensure compliance with every software life cycle process, while to get A classification software only need to comply with some processes (Hrgarek, 2012).

Table 3 and appendix C are complementary and must be used together when assigning risk classification to medical device software.

2.3 Software development processes

After understanding what European regulation and international standards must be followed to comply a software as a medical device, it is necessary to understand and define the software development process itself, what directives it follow and what are the most critical processes and activities that should be addressed, taking into account the level of quality that is needed to achieve medical device compliance.

Humphrey (1988), defines software engineering as “the disciplined application of engineering, scientific, and mathematical principles and methods to the economical production of quality software”. Although being an early definition, in an era where a growing interest over software engineering was felt, it can be focused on two essential aspects that remain strong principles until today, discipline and quality. The focus on the process is a substantive contribution to obtain discipline and quality, while a strong emphasis on meeting customer needs and expectations is always present.

Furthermore, in the same document, Humphrey (1988), defines the software engineering process as “the total set of software engineering activities needed to transform a user’s requirement into

software". These activities are compounded by a wide array, from requirements specification to installation and documentation.

In this chapter, two critical activities regarding this focus on discipline and quality will be further studied: software quality assurance practices and a version control system.

2.3.1 Software Quality Assurance

A mandatory aspect that must be accomplished when developing software is its quality. Software quality has several perspectives that can be analyzed, but it is mainly defined by the way how software meets the desired customer requirements and needs, and how it is compliance with explicitly stated performance requirements, development standards and characteristics expected from professional software (Galin, 2004).

To obtain the desire software quality, it is necessary to have an assurance that practices to achieve this goal are being implemented. Thus, software quality assurance is focused on planning and implementing, systematically, a set of actions that have a substantial impact on the software development process. These actions will have a direct effect when obtaining conformity with technical requirements and managerial requirements as well as improving the efficiency of the developed software (Galin, 2004).

It is understood that it does not exist a concrete list defining all the activities that must be followed to obtain quality, according to the objectives previously defined. Instead, the quality process can be integrated into an agile environment, giving response to frequent changes in the requirements. So, quality in agile can be defined as the capacity of response of the developed software to change. This means that the software must be tested incrementally, accompanying the changes made in each iteration. Two examples of activities that helps in this incremental testing are unit testing and continuous integration (Mnkandla & Dwolatzky, 2006).

2.3.2 Version control system

According to Loeliger and McCullough (2012), a version control system is imperative in software projects, due to the necessity of saving an archive and having a back-up strategy, in such a volatile industry. Essentially, this version control system, in software, is a repository that allows saving all the code that is being generated in the project, where each developer can make how many changes as we want. This repository is evolving at the same time as the project grows, and it represents the entire history of the project, providing access to a log where all editions and changes to the project code are stored.

This need for traceability is validated when the IEC 62304 defines requirements to have a management process for the software configuration and, in particular, a system to control change.

Git is a version control system that, since its origin, in 2005, allowed to handle large projects, though a fully distributed concept, a simple design and interface and supporting non-linear development, with a system of parallel branches. Git has the advantage of dealing with data in a snapshot system. This means that every time that some code is inserted into the repository or some file is changed, Git saves a reference to the state of the system at that moment, just like a snapshot for the system, allowing to easily see the entire history of the project (Chacon & Straub, 2014).

Working with Git also provides the feature of working with branches. In a nutshell, a system of branching allows the development team to have a main branch of development, usually called "master". Then according to each specific feature that is going to be developed, a new branch is

created, diverging from the mainline of development. This means that the code developed under the new branch will not conflict with the code of the "master" branch (Chacon & Straub, 2014).

Using a version control system, in special Git, allow implementing several activities of code verification. One of them is pull request. Following GitHub definition of a pull request, performing this activity allows notifying other team members about a change that is going to be added to a specific branch. Opening a pull request is an opportunity to perform code review, where potential improvements to the code are discussed and reviewed by all collaborators (GitHub).

2.4 Risk Management

Complying with a risk management system is part of the process for certifying a device as a medical device. As previously studied ISO 14971:2019 is the international standard that must be followed when developing this type of software.

Software risk management can be defined as "a set of principles and practices aimed at identifying, analyzing and handling risk factors to improve the chances of achieving a successful project outcome and/or avoid project failure". As software projects can be very complex and have a large structure and dimension, there as recognized as high-risk projects. Thus, the development of a risk management process gains more relevance and helps in different levels, from avoiding disasters and rework to stimulating win-win situations (Bannerman, 2008).

The first step of the risk management process is the development of requirements related to the design input. Since this is a systematic activity, the risk management process is integrated into the design process, to identify and reduce risks, when necessary. The essential conclusion to be drawn when addressing risk management is that the software company must have a risk management procedure that facilitates the conduct of risk management during the product life cycle. This can be achieved through a template, that includes definitions of hazard severities and the probability of occurrence, as well as the risk acceptance criteria (Bartoo, 2003).

The essential steps related to risk management are: identifying the design inputs, that can vary from product features and requirements to design constraints; identifying the potential hazards, as well as their severity and occurrence probability; based on established criteria, evaluate the risk; according to the previous steps identify possible causes and risk control measures; apply the identified risk control measures. This is a process that must be maintained at the same time as the product life cycle, so it is expected that the identification of hazards situations is conducted by the entire team. Different items can be included in that list, from hardware or software failures to injuries that may be inflicted on patients (Bartoo, 2003).

2.5 Software development life cycle methodologies

2.5.1 Plan-driven approaches

Nowadays in such a dynamic and fast-paced software development industry, many challenges arise when it comes to choosing one software development life cycle, that addresses such essential phases in the work of a developer, such as planning, analysis, design, and implementation.

Over the years, some models of software development life cycle were created, namely waterfall, V-Model, and agile (Balaji & Murugaiyan, 2012).

The waterfall model is characterized to be very inflexibility, due to the inability to make changes in a previous phase once the next stage is completed, so there is not the possibility to go back and

revise the previous addresses work (Weisert, 2003). It is reported that during a project development, the requirements change around 25% and that 45% of the features are never used. (Moniruzzaman & Hossain, 2013).

Thus, due to these possibilities of changing the usage of a model like waterfall will not be efficient and the level of response to eventual upcoming changes will be much lower (Williams & Cockburn, 2003).

The V-Model is a modified version of the waterfall model that changes the paradigm when compared with the waterfall model because it allows the verification of previous steps before proceeding to the next one. Since it has a V-shaped structure, like shown in figure 1, it will enable that the developer and the tester work in simultaneously, so the development and testing stages are happening at the same time, involving the tester sooner in the process, allowing it to discover potential bugs or defects earlier (Balaji & Murugaiyan, 2012).

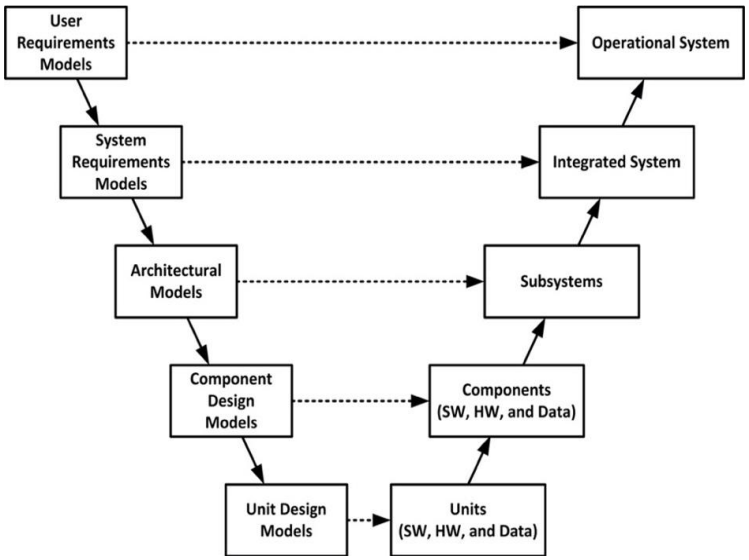


Figure 1 - V-Model (Balaji & Murugaiyan, 2012)

Both these models have clear disadvantages, mainly regarding the requirements phase, because in the waterfall model the requirements cannot change during the software developing process and, in the V-Model, if any change occurs during the process, the requirements documentation must be updated (Balaji & Murugaiyan, 2012).

2.5.2 Agile

This constant technology and business change arise a clear challenge in creating a clear set of requirements up front.

In response to these challenges in the software development process, a new approach arose, the agile approach. This approach is based on the understanding of the dynamic characteristic of software requirements and that those are driven by strong market forces (Moniruzzaman & Hossain, 2013).

So, as a response to the inability of previous approaches, like waterfall and V-Model, to handle rapidly changing environments, an agile methodology will be further analyzed to respond to the specific characteristics imposed by MDF's requirements.

The extreme programming method was one of the first methods being studied and is known as the starting point for the various agile methodologies' methods. Since there, a wide array of methods was introduced, such as Crystal Methods, Feature-Driven Development, among others (Abrahamsson, Salo, Ronkainen, & Warsta, 2017).

Extreme programming is a methodology with the principal goal of simplifying the development process and consists of five phases: Exploration, Planning, Iterations to Release, Productionizing, Maintenance, and Death. It allows close collaboration between the developers and the customer and it is based on quick releases (Abrahamsson et al., 2017).

Regarding Scrum, it is defined as a method that reintroduces the ideas of flexibility, adaptability, and productivity. Applying this approach helps to deal with unpredictable changes in environmental and technical variables, such as requirements, helping to improve existing engineering practices (Abrahamsson et al., 2017).

The agile movement began to gain significance in 2001 with the publication of the Manifesto for Agile Software Development, written for seventeen software practitioners (Beck et al., 2001). This important publication set twelve important principles that address important topics.

First of all, the highest priority is to satisfy the customers, and for that, a change in the paradigm was made. Through agile development, early and continuous deliverables can be made, making the software valuable.

In terms of requirements, agile is prepared to deal with changes, independently of the project stage. This way, the competitive advantage of the customer is never challenged.

Working with a sensitive industry, with restricted requirements and critical nature, such as healthcare, handling with requirements and eventual changes is a strong advantage that will also impact the software development process.

Another objective is to enhance and improve the relationship between the business and developers, and for that, they should work together daily. This focus on the people and their relationship is also very present in other principals from the manifesto, such as making the process of conveying information to and within a development team, more personal and in face-to-face conversations. This is believed to be the most efficient and effective method.

Also motivating, supporting, and trusting the team are essential points from the manifesto (Beck et al., 2001).

Although the integration of agile methods to deal with requirements and their consequential changes, managing requirements in large scale projects is complex and challenging, even more, when dealing with sensitive rules and restrictions (Bass, 2014).

Therefore, in companies with complex business needs and demanding time restrictions, many times the teams are developing having strict deadlines. This leads to increased pressure in all stages of development, eventually leading to not applying better software practices.

Addressing the topic of interest of this work, agile applicability in a strongly regulated industry has had several successful cases. This specific example showcases the case of Abbott's adoption of agile software development methods in a project that required submission for the U.S Food and Drug Administration (FDA). In this specific case, the company used AgileTek's custom methodology called Agile+. This custom methodology has the objective of improving agile's speed and cost-effectiveness, incorporated in a highly disciplined process, indicated for projects in large and regulated industries. In conclusion, the application of agile methodologies had lowered

the cost and shortened the duration of the project, had better and less prescriptive test cases, and reduction in defects. These two last points are of high importance to the quality assurance process, proving that agile methodologies can enhance it. Also, this case demonstrated that agile is beneficial when dealing with changing requirements, a problem faced in these long-duration projects (Rasmussen, Hughes, Jenks, & Skach, 2009).

2.5.2.1 Scrum

Schwaber and Sutherland (2013) defined the entire scrum framework and settled the necessary guidelines and concepts to use it. The authors describe scrum as "a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value" (Schwaber & Sutherland, 2013).

It is a framework that addresses the developing, delivering, and sustainability of complex products as large enterprise projects. The scrum framework gains particular relevance in this work due to its nature. According to the Guide, "Scrum is not a process, technique, or definitive method", which mean that it can be adaptable to each development environment, according to different needs and different processes and techniques can be seamlessly integrated within this framework.

Through the values of transparency, inspection and adaptation, the Guide enhances the high applicability of scrum in complex enterprise products while ensuring that the incremental releases can control its quality.

As previously studied in chapter 2.2, the IEC 62304 does not specify or name specific software engineering methods and techniques for medical device certification. Using scrum framework in these conditions can facilitate the integration of desired techniques, having in mind the desired outcome.

Above all, the scrum adaptability in complex and requirement-demanding projects has been determinant to select the usage of this methodology in this work scope.

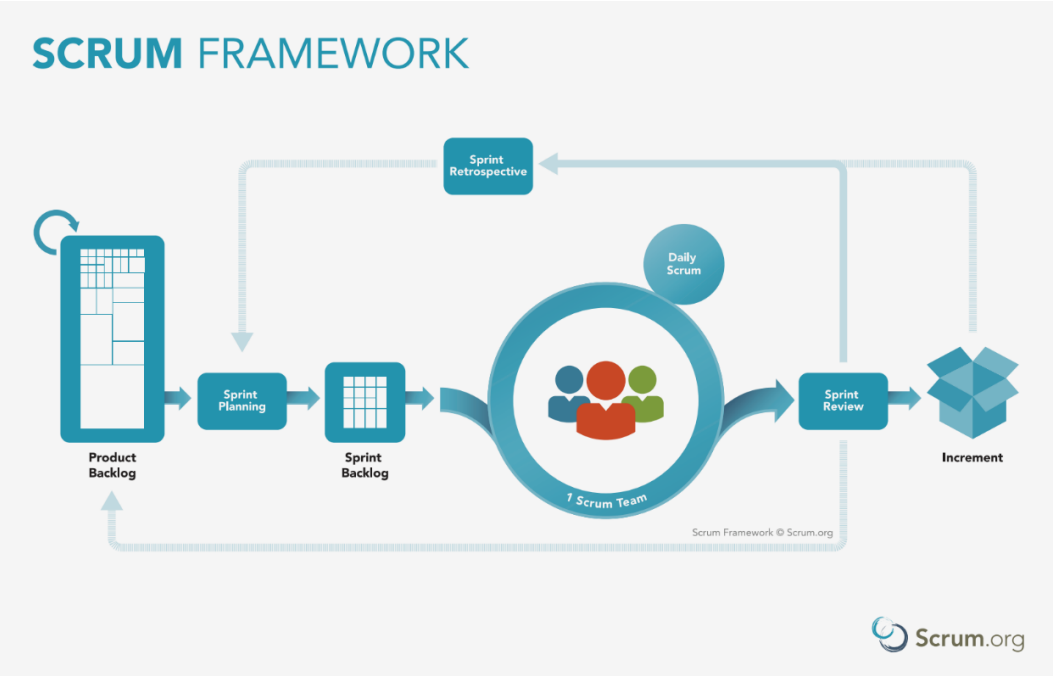


Figure 2 - Scrum Framework (Scrum.org, 2020)

Figure 2 represents the different scrum phases and activities. The product backlog includes all product-related functionalities requirements that are not being used in the current sprint. A backlog item can assume many different forms, like a defect, a customer requested change or a technology upgrade. Then in the sprint planning, through analysis of the backlog items, there are selected which items will enter the sprint and will contribute to the product increment. The sprint is an iterative cycle of work, where a set of development activities are conducted. Closing this cycle, in the sprint review, the incremental work developed during the sprint in reviewed and new backlog items can be introduced (Schwaber, 1997).

Scrum addresses the agility needed in complex projects by implementing these practices. In this methodology the software teams decide what software features are going to be developed in each sprint, resulting in an increment to the software, that can be shipped directly to the customer (G. Lee & Xia, 2010).

3 Problem Characterization

In this chapter, it is analyzed the problem in the light of the previously reviewed literature. To have a good understanding of what problems need to be addressed is crucial for understanding the developments made.

The problem can be generically divided into two main aspects. The first one is the introduction of a more agile approach towards software development methodologies, to improve the main work for the teams. The second aspect is the need to classify software as a medical device, respecting the imposed requirements by international standards.

With the objective of organizing the topics of the present work, a mind map was developed, present in appendix B. This mind map worked as a starting point, not only to develop the literature review but also to struct and organize the problem to be addressed and how the solution is capable of resolving that issue.

3.1 Company and project contextualization

The developed work was performed in an organizational context, at Glintt.

Glintt is a healthcare-focused company that has been focusing in software development, for hospitals, clinics and pharmacies. One of the company's main focus is to provide response to different clinical needs, from clinical trials to patient admission and prescription.

The context behind the development of this specific project marks the disengagement from software development processes and methodologies that are not suitable for a critical context. This project is born in the heart of a newly idealized project, where exists freedom and necessity to implement a suitable software development process. The main objective in applying this process is that, while not distorting the core functions of a developer, it is closely related to the critical nature of healthcare software and the medical device classification.

The main problem that works also as one of the main motivations for this work is a general process improvement, to improve the software quality as well as the Glintt's team satisfaction. Many of the current used processes and methodologies used at Glintt, besides being considered as not suitable for critical systems, are also generally, perceived as outdated comparing to a more agile approach.

Nevertheless, the implementation of agile methodologies in a safety-critical environment can be a problem in itself. According to Ge, Paige and McDermid (2010) the iterative nature of agile methodologies is their most important characteristic. Still, it can also be a problem when developing for this type of systems, since all the standard requirements must be implemented to ensure the safety of the systems. This represents a problem when adapting safety-critical development environments to strongly regulated standards and is one of the main focus of the proposed solution model.

3.2 Software as a Medical Device

Another topic of interest in directly connected with these strong regulated standards in safety critical environments.

The European regulation 2017/745 has expanded the definition of the term medical device, and created a vast set of requirements, supported by huge international standards, that must be adhered to classify a software as a medical device.

To classify a software as a medical device, MDF requirements must be met. This is where problems can start to arise, because complying with such restrict standards can represent a costly commitment for medical devices manufacturers. When first looking to the requirements imposed to achieve compliance, immediately is perceived that these requirements include a wide array of topics, including the process of development, quality testing and more. The more complete the classification of the device that wants to be achieved the more documentation and evidence of correct implementation is necessary, what can turn this process onerous, and too costly and time consuming (Quinn, 2017).

Another problem that arises is that, nowadays at Glintt, the process of documenting and creating evidence is not being performed in parallel with the rest of the development, which means that once the development is complete and exists the need to audit some software, all the work is condensed in a short period and driven by necessity, which can potentially lead to a lower quality in documentation. Shifting this perspective to include in the development process itself all the activities needed to achieve compliance facilitates every audit process in the future.

Analyzing the regulations and standards can also be a problem, since they are extensive and required a specific person in the project to perform that task, which is not directly related with any function existent nowadays at Glintt.

Since the company identified the need of creating a software that is compliant with the international standards, achieving a medical device classification, together with implementing agile practices, are the problematics that are addressed.

4 Methodology

4.1 Existing approaches and reasons for the choice of the adopted approach

For this project and proposed objectives, besides implementing agile methodologies in the development of a healthcare solution to classify it as a medical device, it is essential to collect data and information that will work has grounding foundations to support the software development process premises and consequently the implementation of agile methodologies.

Therefore, two possible approaches can be used in the development of this work: qualitative and quantitative paradigms. The quantitative paradigm is characterized by an empirical investigation, in which there is no room for human perception to take part and only exists one truth. This is made possible by the ability to study a given object without influencing it or being influenced by it (Sale, Lohfeld, & Brazil, 2002).

On the other hand, the qualitative paradigm is characterized by having multiple realities or multiple truths based on a socially constructed and ever-changing reality. Thus, in this paradigm, it exists a relationship between the researcher and participants that allows the development of a mutually created result (Sale et al., 2002).

After framing the two types of approaches that can be followed, it is important to conceptualize the process of research as a process of, gradually reduce the uncertainty about the phenomena or questions in study. As the investigation intensifies, the level of uncertainty can enlarge, not only about the problems but also if the right questions are being formulated and how they can be framed in a meaningful way and to the right set of personas. Qualitative methods are beneficial when the researcher is constructing or developing theories. It can also be used when creating a conceptual framework or generating hypothesis (Sofaer, 1999).

Analyzing the scope and characteristics of the project to be developed and having in consideration the final work to be presented, in the form of a life cycle model for the software development process, based on the scrum framework, qualitative research should be conducted. This qualitative research can be supported with several data collection methods. The data collection method used to develop this work is interviews, which will be explained in further detail in the next chapter.

4.2 Method used in project

After understanding the nature and the objectives of developing a qualitative research, it is needed to explore and identify a data collection method to gather the appropriate information. Several data collection methods can be used, such as observations, textual or visual analysis and interviews (individual or in group) (Gill, Stewart, Treasure, & Chadwick, 2008).

To obtain a deeper understanding of the subject in question and explore the detailed insights that can be obtained from the participants (Gill et al., 2008), interviews were the data collection method chosen.

The data for this study were collected between the 5th of June 2020 and the 14th of July 2020. A total of ten interviews were conducted, where, due to the pandemic situation, nine of these interviews were conducted through video chat using Microsoft Teams, and the remaining interview was done face-to-face.

Online interviews are seen as a viable alternative when performing qualitative research, allowing the researchers to choose from a varied communication array, and maintain the facility of talking

directly with the interviewees. This type of online interview can adapt the form of verbal or written communication and have a script or being just an interchange of ideas (Salmons, 2014). This type of interview helped not to miss the main goal and allowed some flexibility since many of the interviews were performed during working hours.

For these interviews, a previously defined script was followed, which can be found in appendix A. The main goal is to understand the interviewee's point of view, according to their professional experience.

Although the script for the interview was previously defined and written, containing a set of questions touching all the key points, the interviews that were conducted can be defined as a semi-structured interview. This type of interview is particularly relevant for this topic of interest because it allows starting from a key question defining the area of interest and then, diverge to capture some other idea or perspective in more detail by the interviewee that may have not been initially thought in the design of the interview (Gill et al., 2008).

Due to the multidisciplinary nature of the agile teams and the importance of the different roles in a software development process, distinct profiles where interviewed, to gather information according to different perspectives, backgrounds, and stages of the process. The four roles that were selected to interview were:

- Developers;
- Product Manager;
- Quality Assurance;
- Team Leaders.

In a scrum team, there are different responsibilities according to the different roles played by members. The objective of defining this set of diverse roles was to understand each interview perspective about the software development process and the usage of methodologies, in specific in a healthcare environment, fitting their daily responsibilities and own previous experiences. Taking this into consideration two of the interviewees are playing a senior role, with years of different professional experiences in other companies and other business areas. These experiences were particularly important to understand their perception of the unique characteristics of software development in the healthcare industry.

Touching all these areas of the software development process gave a holistic view and understanding, helping to realize where are present the main aspects that must be addressed while trying to classify a SaMD and what contribution the application of agile methodologies can have.

All the interviewees were previously contacted to explain the objectives of the session and the interviews were scheduled afterwards. Before beginning the study a contextualization was made, to explain the objectives, depict the subjects to be addressed, and also to offer some guidance to the interviewee (Foddy & Foddy, 1994). Also, due to the large amount of data generated in the interviews, these were audio-recorded, and all the participants signed an informed consent, ensuring also the anonymity and confidentiality of the interview.

Table 4 shows the different responsibilities of the interviewees.

Table 4 - Interviewees responsibilities

Name	Responsibility
Developer 1	Developer
Developer 2	Developer
Developer 3	Developer
Developer 4	Developer
Product Manager 1	Product Manager
Quality Assurance 1	Quality Assurance
Quality Assurance 2	Quality Assurance
Team Leader 1	Team Leader
Team Leader 2	Team Leader
Team Leader 3	Team Leader

The interviewees represent members of five different teams in Glintt. Developer 1 and Team Leader 1 belong to a core team, responsible for managing, give support, and develop new features in a wide variety of products. Developer 2, Developer 3, and Team Leader 2 belong to a team that is specifically performing a digital transformation from a core, legacy product to a new HTML version. Developer 4, Quality Assurance, 1 and Team Leader 3 are all members of a recently formed team, that is currently trying to work with new technologies and methodologies disruptively. Although these 3 members have worked for several years at Glintt in different projects, during the interviews it was asked to present their perspective on this new project, to illustrate contrasting realities. Product Manager 1 has also worked in several projects and the vision presented mirrors all these experiences. Quality Assurance 2 is not allocated to a specific project, being part of the general quality team. It is important as well to understand this perspective, as the quality team only intervenes in the last stage of the development process, by contrast with the current experience that Quality Assurance is having.

The interview script was defined with different objectives in mind:

- Understand the software development process currently in use by the teams
- The strengths and weaknesses of the software development process currently in use by the teams
- Understand the perception of the teams using agile methodologies
- Impacts of using agile methodologies in the software development process
- Specific characteristics of developing healthcare software
- How agile methodologies can be integrated into a healthcare software development environment

Firstly, and before starting to understand the methodologies the first primary objective was to understand the software development process because it lays the basis and major rules to be followed. Since the methodology used can influence the process and not the other way around, a

good understanding of the process is vital and helps to map the current advantages and disadvantages.

After, the methodologies topic is addressed. Here, a very interesting point is approached since many of Glintt's teams do not use an agile methodology, identifying some problems and obstacles. These problems and how to overcome them will be further developed in the results chapter of the current document.

Lastly, and addressing such a specific software area some questions regarding the perception of this business area and what care should be taken when developing for this industry, having as well in mind how these agile methodologies can integrate themselves in this environment while improving the general quality of the final delivery.

4.3 Data Analysis

All the data generated was transcribed from the audio recordings. Then, the transcribed data was cross-referenced with the notes taken during the interview, allowing a process of self-reflection into the more meaningful data. This transcribed data was organized, in an Excel file, to facilitate the coding process. This coding process is crucial to categorize the data, making it easier to analyze it further. Each code can refer to some large amount of data or can be just applied to specific words or short sentences. The main idea is that each code is correlated with an idea or concept (Ranney et al., 2015).

This process allows to better understand the topic of interest according to the interviewee's perspective since different issues, similarities and differences are captured and coded (Sutton & Austin, 2015).

5 Results

Having the previously explained objectives in mind helped conduct the interviews. A better perception about current in use methodologies was gained, contrasting with different ideas and different conceptions about using agile methodologies, and what are their advantages and disadvantages. Additionally, it is very important to understand how the critical nature of this kind of healthcare software can affect the development and what is the role of agile methodologies in this kind of fast-paced critical business area.

5.1 Results of the data collection and analysis

Table 5 represents the main categories that were identified during the interviews, serving as a starting point to describe in more detail each topic. These categories represent the totality of questions, and all the interviewees have responded.

Table 5 - General coding

Codes	Sources (n=10)
Software development process currently in use by the teams	10
Strengths and weaknesses of the software development process currently in use by the teams	10
Perception of the teams using agile methodologies	10
Impact of using agile methodologies in the software development process	10
Specific characteristics of developing healthcare software	10
How agile methodologies can be integrated into a healthcare software development environment	10

5.2 Software development process currently in use by the teams

This section analyses the software development process currently in use by the teams interviewed and what methodologies are used to support that process. Analyzing table 6, four sources refer to use waterfall methodology, identified as the main methodology used, while three sources use this methodology with some scrum influences.

Three sources refer to use scrum methodology, but only one team uses it with consolidated experience and with an outsourced team. The other two sources are less experienced in using scrum.

The main goal of this section is to focus on the software development process currently in use by the teams and how methodologies can influence this process.

Table 6 - Software development process currently in use by the teams

Codes	Sources (n=10)
Definition of the process	8
Waterfall	4
Monthly planning	2
Activities plan in Jira	2
Scrum	3
Risk Management	1
Grooming activities	2
Waterfall with scrum influences	3
Acceptance criteria	2
Product Backlog	1
Influenced by external factors	1
GIT Flow practices	4
Implementation of GIT control version system	1
Code Review	1
System of branches	2
Quality Assurance	2
Integration of the QA team in the entire process	1
Relationship with other teams	2
Happy Path and Sad Path	1
Prone to error due to the need for testing in two environments	1
Automatization of the process	2

In the section “Definition of the process” of table 6 are coded some activities inherent to the software development process and is represented how these activities relate themselves with the methodologies in use. It is showed that only two activities of the process are exclusively used by the scrum teams, the “Risk Management” and “Grooming activities”. In contrast, the other process activities are being used in the waterfall methodology. Even the teams using a waterfall methodology are starting to use some activities defined by the scrum methodology, such as “Product Backlog”, where the tasks assigned move through different states, such as “To Do” or “In Progress”, helping organize the development.

In one team, the weight of external factors, such as time or budget constraints, is felt, and this is when a good risk management document, developed collaboratively by the entire team, is much

needed. Risk management is one of the aspects that is going to be approached and seemingly integrated into the scrum proposed model, filling this gap.

Regarding technical characteristics of the software development process, changing to a GIT version control system, helped to have more significant control over the code that is integrated into the final development version, by contrast with the previous team foundation version control system. Also, it was implemented a system of Code Review in the process, improving the quality of the code. These changes significantly improve the code quality and the processes for safeguarding the final delivery.

Team Leader 2 states that some technical characteristic of the software development process, together with the usage of some agile inputs are being beneficial for the improvement of the process.

"Recently, in my team, there were applied some major changes that, combined with agile methodologies, improved the entire process."

Team Leader 2

"The change to GIT helped in implementing the culture of code review within the team and helped to improve the general quality of the code."

Team Leader 2

Concerning quality assurance aspects, interviewing two members of two different teams presented two very distinct visions regarding the process. Substantially, the main difference lies in how the team is integrated into the software development process and their relationship with the other teams. One of the most important aspects when classifying a SaMD is the software quality and how it fulfils the acceptance criteria. Therefore, the role of the quality assurance team is very important and plays a leading role in the entire process.

The integration of a scrum methodology, besides other quality validity processes, further analyzed in chapter 6, can improve the general impact that the quality assurance team has in the software development process.

Also, there are being implemented some automation processes, facilitating some stages of the software development process that can become repetitive and time-consuming. These automation processes are also ensuring more code validity stages and help to decrease potential human failures.

Overall, this focus on the process and some of the most important phases, namely the most crucial in classifying a SaMD, will be further discussed in chapter 6, where will be integrated with a scrum methodology.

5.3 Strengths and weaknesses of the software development process currently in use by the teams

In this section are depicted the strengths and weaknesses of the software development process addressed in the previous section.

Table 7 represents the codes identified and its notorious a more significant number of weaknesses than strengths recognized by the interviewees, leaving room for improvements.

Table 7 - Strengths and weaknesses of the software development currently in use by the teams

Codes	Sources (n=10)
Strengths	6
The developer is consulted regarding times of development	1
Code is always reviewed	2
Improvements in the quality assurance process	1
Agility	1
Weaknesses	10
Waterfall	8
Lack of kick-off meeting	2
Mid project alterations, like the addition of more tasks	2
Possible errors are found out too late in the process	2
Wrong time estimations	3
Strict and not suited to fast-paced environments	1
Quality	3
Lack of unitary testing	1
QA Team is not integrated into the development process	2
Code integration problems	1
Lack of practical experience when implementing scrum	1

Begging the analysis, a very interesting dichotomy was approached.

Most of Glintt's teams use a waterfall methodology, as demonstrated in section 5.2, considered by many of the interviews outdated and it is perceived as an obstacle in the software development process.

This path towards agile methodologies is slowly being implemented, namely in a recently formed team that has identified most of the strengths of the process. This team is fully implementing a scrum methodology. It serves as proof of its applicability and usefulness in healthcare fast-paced environments, namely by improving the quality assurance process and by taking advantage of the incremental deliveries, to get client's feedback.

The waterfall methodology can create some obstacles and difficulties, that a shift in the paradigm, towards agile methodologies, can improve. The interviewee's experience proves that this process is rigorous and that its structure is not framed with a fast-paced environment, such as healthcare development, as Developer 1 and Team Leader 1 states.

"The main methodology used is waterfall, which can lead to possible errors that may arise during the developments that are only discovered in the last stage, causing a lower quality in the software and a considerable period to fix those errors."

Developer 1

"I think that the current process is too strict and takes too much time, for example, when some specification of software is defined by the Product Manager it is very difficult to do some changes to that specification, which can cause some challenges to the developers."

Team Leader 1

"A waterfall methodology is not very suited to today's market, since there is a shortage of agility, much needed in nowadays fast-paced industries."

Team Leader 1

There are also difficulties when defining development. In the teams using waterfall methodologies, exists the concept of the kick-off meeting. This meeting is where the development in question is presented and times are assigned to the developers. The main problem is that the kick-off meetings only exists in some developments and that many of the developments are altered mid-development, with the addition of more tasks.

Since the time for the development is defined in the kick-off meeting, no time is planned for additional tasks, leading to incorrect time estimates. Alongside this problem, developers are not present in the kick-off meetings and the team leader defines the time estimate. As a result, quite often, the time for the development is misaligned with the developer estimative.

5.4 Perception of the teams using agile methodologies

In the last sections it was analyzed some of the strengths and weaknesses of using methodologies in the software development process, with a strong focus on understanding the differences between a waterfall methodology or an agile methodology.

In this sector the principal purpose is to explore some specific objectives and what are the team's perception of using agile methodologies to fulfill different objectives.

Therefore, there were identified three major areas: Value and quality in the final delivery, requirements interpretation and documentation. These areas are perceived as critical both in improving the software development process and on this will lead to a better and clearer classification of the software as a medical device. In each of these the interviewees have identified some benefits.

Table 8 - Perception of the teams using agile methodologies

Codes	Sources (n=10)
Value and quality in the final delivery	10
Inclusion of tests improve the final quality	2
More control in the entire software development process	4
Better management of time, resources and quality	1
Iterative process helps in the inclusion of the client along the process	2
Requirements interpretation	8
Acceptance criteria is important to narrow developers focus	4
Jira centralization	1
Better written and defined user stories	2
Quicker customer feedback to improve the next acceptance criteria	1
Documentation	8
Very important point in the software development process	3
Code maintenance and traceability	1
Methodology helps to create a process for documenting	4
Develop just the right and more relevant documentation	1
Methodology helps to define better the entire process	1
Automation	1
Good code practices	1

The improved value and more controlled quality of the final software delivered to the customer were mentioned by all the ten sources, as demonstrated in table 8. Applying an agile methodology can be beneficial for the inclusion of several types of code tests, which will, inherently, safeguard the final quality of the software. Together with a more controlled development process, it is demonstrated a unanimous feedback that the final product can reach a higher quality, specially by Quality Assurance 1.

"Without question the inclusion of quality aspects in the methodology helps to improve the final quality of the delivery."

Quality Assurance 1

Another important aspect is the interpretation of the requirements. They assume an important role in the development process, since is the starting point for the developer work. The centralization

of information in Jira, together with better written and defined user stories help, in a large scale, in narrowing developer's focus to the code development itself. Team Leader 1 corroborates this point in particularly, saying that:

"It is very important because it helps you to focus only in what is required by the specification. You can think better in what it is needed to develop and perform a more modular interpretation."

Team Leader 1

The documentation is also a crucial factor in the development process and here the methodology also plays a fundamental role, because it can help to create a process for developing documentation, much needed for code maintenance and traceability. However, it is essential not to lose the focus and developed just the right amount of documentation to not waist unnecessary time and resources.

"Process over documentation. Develop the necessary documentation to achieve system comprehension, but without causing entropy."

Team Leader 1

5.5 Impact of using agile methodologies in the software development process

After detailing the currently in use software development process, how methodologies and their strengths and weaknesses support it, the focus was on understanding the specific usage of agile methodologies.

So, in the previous section it was addressed different objectives and what type of response these methodologies can give. This section focusses on depicting the strengths and weaknesses of using agile methodologies and how they can leverage the software development process.

Table 9 shows that more sources identified strengths than weaknesses, proving that agile is a step forward and it is perceived as an improvement, moving away from more plan driven methodologies.

Table 9 - Impact of using agile methodologies in the software development process

Codes	Sources (n=10)
Strengths	8
Strong definition of the process and delivery deadlines	3
Process and team organization	3
Quick delivery and iterative process	3
Client feedback	2
Entropy reduction	1
Easier to react to unforeseen events	1
Weaknesses	4
Applying a methodology can be more time consuming	1
High responsibility in each team member role	1
Need to respect the methodology guidelines	1
If the entry point of the process fails, the entire process fails	1

Definition of the process and delivery deadlines, team organization and quick and iterative processes were identified more often as the significant strengths of agile methodologies. The client and how its expectations are managed during the time that a project is being developed, are also points where the usage of an agile methodology has a significant impact since their incremental nature allow the client to be more involved and aware of the project status, as well as provide constant feedback. It also allows to implement some user testing, and according to the results change and improve ahead. All these strengths are identified by Team Leader 3.

"When using agile methodologies, it is easier to foresee unexpected events, since the iterative deliveries allow that the client can participate in the product evolution. The user testing allows to iterate on the result and improve according to the feedback."

Team Leader 3

Fewer sources have pointed out weaknesses, for the most part, to do with the application of the methodology, the role of each team member, and the need to respect the methodologies guidelines. In a nutshell, these are all related and can be overcome with experience. It was also mentioned the high importance and responsibility of the product manager, since it is the main entry point when gathering business requirements and translate them to acceptance criteria. Since the process is very dependent on these acceptance criteria if this entry point fails, the entire process fails.

5.6 Specific characteristics of developing healthcare software

This section approach all the specificities and care measures to have in account when developing healthcare software. It maps the perception of the interviewees and what it is considered important, comparing, for example, with different business areas. Table 10 presents the information regarding these specific characteristics.

Table 10 - Specific characteristics of developing healthcare software

Codes	Sources (n=10)
Design	2
Must have a very appealing and straightforward design	2
Quality	8
Must have great quality and enough code test coverage	8
Critical business area	4
High learning curve in terms of associated concepts	1
Importance of agile methodologies	2
Technical	3
Must ensure data privacy policies	2
Databases can reach a huge amount of data	1
Need to connect to external API's	1

Two of the most important aspects identified are related to the quality of the software. It is mandatory to have excellent quality and enough code test coverage to verify and ensure quality for the final client. This is the more important aspect to highlight regarding this section, emphasized by Team Leader 3 and Developer 3.

"Since this type of software is to assist the patient, it has to be very carefully developed and tested. We must try to have the maximum testing guarantee and also to have in mind how we can facilitate the life of the healthcare providers."

Team Leader 3

"We have an increased responsibility when developing, that many people do not take seriously. It is needed special attention to the quality of the software."

Developer 3

Also, some sources identify this business area as a critical one, strengthening once more the importance of the quality and the necessary care when developing. This criticality inherent to this type of software is highlighted by Product Manager 1, as well as the legal implications involved.

"There are more legal implications involved in this type of software. It must ensure data privacy policies and not cause clinical risk situations. For this, they can achieve a very high complexity and be supported by huge databases. Is a more critical software, with high clinical risk and several legal implications."

Product Manager 1

This type of software can also have a high learning curve, as stated by Developer 4, and it is highlighted the importance of agile methodologies as well, namely when delivering incremental value.

"It is a very critical software, with a high learning curve, in terms of concepts. I think that each business area has its risk. In an area as healthcare, it is very important an agile methodology, to incrementally deliver value."

Developer 4

5.7 How agile methodologies can be integrated into a healthcare software development environment

This last section analyzes how agile methodologies can be integrated into this specific software development area.

The general idea of all the respondents is that agile methodologies can adapt to any business area, due to its agility and how they respond to the current market needs. This proves that, in an area previously categorized as critical, agile provides a capable and robust response, as highlighted by Team Leader 3.

Table 11 - How agile methodologies can be integrated into a healthcare software development environment

Codes	*Sources (n=10)*
Agile methodologies adapt to any business area	8
Importance of agile	4
More frequent and market-oriented releases	2
Smaller but more focused teams	1
Risk management documentation	2
Allow defining a strong process	2
Quality assurance	1
Ensure that no errors get to the final client	1

Analyzing the table 11, it can be interpreted as a summary of all the previously addressed topics. It is perceived the importance of agile and quality assurance.

"These methodologies are suitable for a healthcare environment because it allows the define an agile and strong process to ensure quality."

Team Leader 3

6 Proposed solution

In chapter two, it was detailed how the new European regulations applied changes regarding the scope of the medical device, as a wider variety of software can achieve this classification, according to the intended class. These changes, together with the international standards, had a direct impact on the software development process and its life cycle and how this is defined and maintained. Also, the introduction of agile methodologies, namely scrum, is changing the paradigm of software development, increasing flexibility, and providing a more substantial and faster response to the healthcare software requirements.

Chapter five has validated the relevance of using an agile methodology together with a robust process, namely in terms of quality assurance and code verification activities. Applying these changes results in overall improved software quality, while never neglecting the critical nature of healthcare development. The interview results have confirmed that agile methodologies can be integrated into a healthcare software development environment. Interviews also showed that a change from waterfall methodologies to agile methodologies and agile-based activities could improve the process as well as the value and quality of the final delivery, by integrating tests and have a better requirement interpretation.

At the light of the requirements defined in the IEC 62304:2006, represented in table 3, the objective of this chapter is to define a life cycle model for the software development process, using scrum principles, as this methodology allows the definition of a process that better suits the team necessities and objectives. For this purpose, are defined specific processes and development activities which, in addition to scrum principles can comply a software as a medical device.

Figure 3 represents the proposed life cycle model for the software development process. The documents, activities, processes and team represented in red are not addressed in the traditional scrum definition and represent additions to it. The documents, activities, processes and teams represented in black are contemplated in the scrum guide. The main goal is that all the additions integrated into the scrum model do not cause entropy or misrepresentation the scrum values, as well as do not interfere with the normal course of a sprint. Thus, the main goal behind implementing all these strategies is to create conditions that without distorting the flexibility provided by agile methodologies create a software development environment that allows compliance with the IEC 62304.

Implementing the Product Vision stage and the documents that are developed there is closely related with part 5 “Software development process” of the IEC 62304; developing an Identity Card for representing the requirements, based on epics and user stories is related with part 5.2 “Requirement analysis” of the IEC 62304; developing a software architecture document complies with the requirement 5.3 “Architectural design” of the IEC 62304 while performing a pre-grooming activity helps in responding with part 5.7 “System testing” and part 6 “Software maintenance process”. The change of paradigm towards the Quality Assurance team, which is not initially defined by the scrum guide, is performed also based on the conclusions obtained from the results of the interviews. Since critical safety systems must have a strong focus on quality and the IEC 62304 also reinforces it, the Quality Assurance team and consequential activities play a crucial role in the proposed life cycle model. Risk Management is an activity that accompanies the life cycle since it is highly iterative as allows the system to comply with part 7 “Software risk management process” of the IEC 62304.

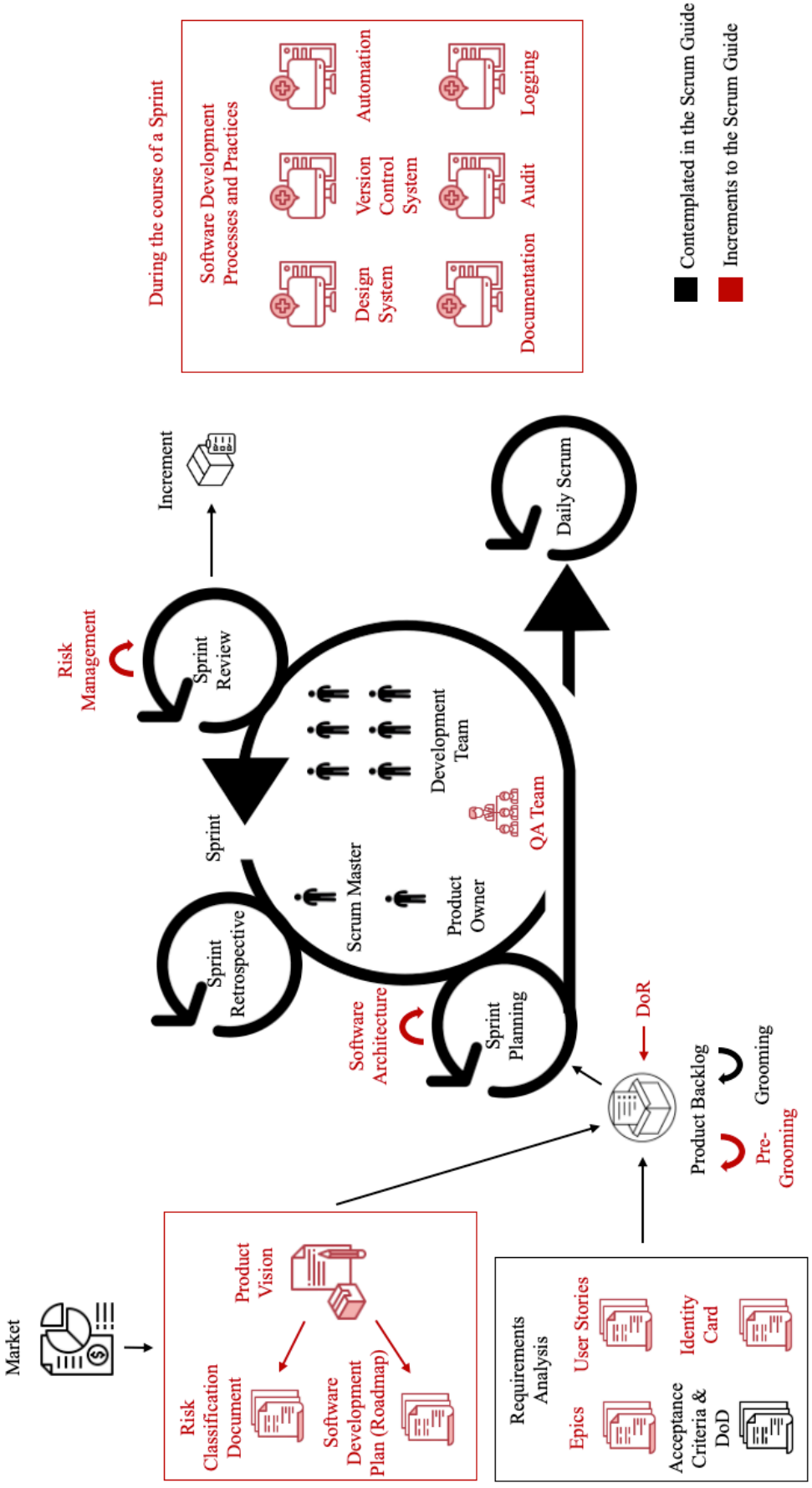


Figure 3 - Proposed life cycle model, using scrum principles

The software development processes and practices identified in the rectangle on the right side of the image are transversal to sprints and occur on every sprint. For implementing all these processes, a previous work of configuration and implementation is needed before the project starts, intending to reduce the amount of work and time invested every time that some of those need to be triggered.

The implementation of a design system aims to comply with the point 5.4 “Detailed design” of the IEC 62304; version control system and automation processes are essential addressing topics of part 5 “Software development process”, part 6 “Software maintenance process” and part 8 “Software configuration management process” of the IEC 62304. Documenting and keeping a traceability record is also an important transversal activity that influences the majority of IEC 62304 requirements. Audit and logging are two specific activities that, implemented together, aim to comply with part 9 “Software Problem Resolution Process” of the IEC 62304.

The outcomes of the interviews performed allowed to define the requirement analysis process better, in the light of team members experiences. In chapter 5.4 are obtained important conclusions such as “Acceptance criteria is important to narrow developers focus” or “Better written and defined user stories”, which were fundamental in understanding how to improve the requirement analysis process. The inclusion of the Quality Assurance team in the process as a member of the scrum team was a key finding in chapter 5.3 as the quality assurance member interviewed has identified it as one of the problems that have more impact in the software development process currently in use by teams. The implementation of a version control system, the rationales used to document the system and the implementation of automation processes, were also influenced by the interview results. According to the teams, implementing all these processes enhance the existing software development process, as seen in the previous chapters 5.4 and 5.5.

Besides, the implementation of all the agile activities described in the scrum guide (represented in black in figure 3) is also validated by the teams as suitable and essential for a healthcare software development environment.

6.1 Scrum Methodology

Firstly, it is detailed how the scrum methodology is included in the process and what are its benefits. The way scrum organizes the teams, its ceremonies and what artifacts are generated are approached in this first chapter, as well as their impact and how it helps the software to comply with the IEC 62304:2006. All the fundamentals of these activities are based in the Scrum Guide written by Schwaber & Sutherland (2013).

- Scrum Team - Scrum teams have the characteristics of being self-organized and cross-functional. In this model, besides defining the usual members of the scrum team (scrum master, product owner and development team), is added the quality assurance team. Also, as the scrum teams deliver the work iteratively and incrementally, they can maximize the opportunities for receiving feedback and quickly solve any problem.
- Scrum Master - The scrum master is responsible for applying scrum as defined in the Scrum Guide, helping all the team understand scrum theory, practices, rules, and values. The Scrum Master is a facilitator who ensures that the Scrum Team is provided with an environment conducive to complete the project successfully.
- Product Owner - The primary responsibility of the product owner is to maximize the value of the product produced by the development team. It is also responsible for articulating customer requirements and maintaining business justification for the project. One of the main activities of the product owner is to create and manage the product backlog.

- Development Team - The development team is constituted by the developers that transform the product backlog items in potentially releasable features, to be integrated into the software application.

6.2 Scrum ceremonies

The scrum framework also suggests the realization of scrum events. Scrum events are prescribed events that have the objective of creating regularity in terms of meetings and only encompasses events that are defined in scrum. Each event in scrum is an opportunity to analyze the work done or the work to be developed and adapt something necessary, always having a critical transparency and inspection in mind. The proposed model, represented in the figure 3, applies all the scrum ceremonies.

- Sprint - The main scrum event is the sprint. It consists of a time-space where a potentially releasable product increment is created. The time duration of the sprints can vary, but once a development of a product is started, each sprint of that development must have the same period.
- Daily Scrum - The daily scrum is a 15-minute reunion held each day of the period of the sprint. In this reunion each member of the development team exposes the work addressed the previous day, the work that is going to develop in the current day and if it had any impediment that prevents it from reaching the sprint goal.
- Sprint Planning - In the sprint planning the work to be developed in that sprint is defined, through a collaborative work of the entire scrum team. It is the responsibility of the scrum master to schedule this event and make sure that each one of the participants understands their purpose. At the end of the sprint planning the development team should have a clear vision on how they will organize them-selves for accomplish the sprint goal and consequentially, develop the defined increment. Also, this model defines the creation of a software architecture, which is defined before the first sprint of the project and can be discussed in this event.
- Sprint Review - The sprint review takes place at the end of the sprint to analyze and adapt, if necessary, the product backlog. During the sprint review a critical reflection is conducted, where the scrum team and the stakeholders discuss the work that was developed during the sprint. Since it is an informal meeting, the presentation of the increment, in form of work produced, has the objective to generate feedback and foster collaboration of all the intervenient. As a result of the sprint review, a revised product backlog is generated that defines the next items to be addressed in the next sprint. In every sprint review, this life cycle model proposes also the revision of the risk management document, where future risk can be defined.
- Sprint Retrospective - The last scrum event is the sprint retrospective. It takes place after the sprint review and has the purpose of analyzing the previous sprint overall performance. Improvements for the next sprints can be found, through a collaborative discussion about what things went well and what things went worse.

6.3 Product Vision

Product vision is an activity included in this model, that has an extreme importance, giving the need to comply with IEC 62304.

This activity takes place long before the development and the definition of the process starts. It is conducted by the product owner and essentially, in the product vision phase, the main purpose of the product is defined as well as the intention for the product. Ultimately, it is a vision for the future, containing the ambitions for the product and describing what problems the product tries to resolve.

Alongside, the main software architectural pieces are defined, according to the macro requirements. Later on, these pieces are going to be integrated into the software architecture document. In this model, the product vision has the responsibility to define two essential documents.

The first one defines the required risk classification for the software as medical device. This can be classified has class A, B or C according to its risk to the patient. This must be defined and justified in a document, resultant from the product vision. According to this classification, the software should respond to different requirements of the IEC 62304, as can be seen in appendix C.

The other document is inherently connected with the point 5.1 “Development planning” of the IEC 62304. The product owner must create a development planning document. According to the standard, this document should contain information regarding several items of the development process, namely: system design; planning of development standards, methods and tools; software integration planning and integration tests; software verification planning; risk management planning; documentation planning.

This development planning document sets the base for the core start of the project and many of its points are going to be addressed and included in the model proposed in this chapter.

6.4 Requirements Analysis

The IEC 62304 has one point concerning requirements analysis, the point 5.2 “Requirement Analysis”, where defines the creation of a document reflecting the software requirements, as well as systems to control their updates and verification processes. Since requirements analysis, in the software development process, is also a critical activity, this model addresses special activities and processes, integrated into the scrum methodology to handle requirements definition, control and verification.

Firstly, a requirement addresses a business need conveyed by the customer. For achieving this goal, there are performed the four main activities defined by requirements engineering: elicitation, documentation, validation and management. So, every user need is translated into documented and validated requirements. After this, the requirements are integrated into an agile environment and are addressed with a greater granularity.

The scrum guide does not define epics and user stories as techniques to write requirements. Nevertheless, these two methods are widely used in agile environments and going to be used in this proposed model.

6.4.1 Epics

An epic is a large body of work that can be divided into smaller specific tasks, that is based on the needs of end-users. An epic can be a high-level description of what the end-user wants and what value this customer need conveys. An epic is a high-level requirement.

The primary value of epics is the possibility to reflect the most important goals of end-users of the product.

An epic is created based on the previously defined requirements, where one requirement can be translated into one or more epics.

6.4.2 User Stories

To achieve an even greater granularity, epics can be divided into user stories. Stories are one of the core components in an agile approach to software development. User stories are defined for providing a user-centered perspective and are written in a non-technical language, providing context for the entire team.

The previously defined requirements convey a functionality, based on a business need. The main difference for a user story, besides its granularity is that a story focuses on the experience, and on what the user wants to achieve when performing a functionality.

User stories in an agile environment are usually the principal piece of work, and represented through a template, perceptible to all. This template is:

"As a [who wants to accomplish some goal]

I want to [what wants to be achieved]

So that [why the user wants to achieve that goal]"

The user stories are kept simple, only with the above description. Further detail, in the form of acceptance criteria, should then be added in the pre-grooming stage (chapter 6.5.3).

6.4.3 Acceptance Criteria & Definition of Done

Defining the template for a user story focus on what the user wants to achieve. However, can be a little vague, leaving room for indecisions. Acceptance criteria and definition of done (DoD) are two concepts that help to define the full context of a user story and clearly defining its objectives.

Acceptance criteria are a set of conditions used to confirm when a specific user story is completed. These criteria provide a deeper and better understanding of the user story since they provide key information on how user story perform. Acceptance criteria is used to define boundaries and to reach consensus, meaning that, through acceptance criteria the development team understand exactly what functionalities that user story must meet and what the client desires. Acceptance criteria also represent the business rules for that client need. Another important aspect is that acceptance criteria as a strong correlation with the quality process, as these criteria serve as a starting point for defining test cases. This relation is further studied in chapter 6.7.2.1.

Additionally, DoD is a list of requirements that a user story must fulfill to be considered completed by the scrum team, promoting the quality of the developed work. Some points that must be present in DoD for any user story, part of a system that aims to be considered a medical device, are:

- Code developed according to the desired functionality;
- The objective of the user story is completed;
- Acceptance criteria are met;
- Code reviewed;
- Unit, automated, functional and regression tests are completed;
- Project builds without errors;

- Project deployed in a test environment;
- The product backlog is updated;
- The documentation is updated.

The principal difference between the two lies in the fact that, while DoD is common for all user stories, acceptance criteria are different and specific for each user story.

For a user story to be completed, both DoD and acceptance criteria must be met.

At the light of IEC 62304 point 5.5 “Unit implementation and verification”, defining acceptance criteria to verify each software unit implementation is a mandatory point. Applying the previously defined acceptance criteria defines that a specific user story is completed.

6.4.4 Identity card of a software component

Another addition to this development life cycle model is the development of a document called identify card. Each software component has its identity card, working basically as an information aggregator, centralizing all the information regarding to that specific component.

This identify card is based on a software requirement specification document (SRS). In its definition an SRS describes what the software will do and what are the expectations regarding to how the system will perform. Besides, it describes the functionalities that the product must give response, in both business and user needs.

The identity card used in this model is compound by several topics:

- Objective
- Uses Cases
- Requirements
- Success Metrics
- User Interaction and Design
- User Story Workflow
- Test cases
- Technical Requirements
- Risk assessment matrix (if appliable)

Having this centralized access to information helps to organize the entire process and creates a single source of truth for each software component, which gets special emphasis when the international standard for medical devices imposes a system of traceability.

Besides the platform used to achieve traceability (Jira, addressed in chapter 6.7.6.2), this identity card creates a possibility of navigation between different points of the system, that, otherwise will be very hard to achieve, improving traceability through different artifacts and information.

The identity card works as the requirements specification document, needed to comply with the point 5.2 “Requirement Analysis” of the IEC 62304.

6.5 Agile artifacts and activities

In this chapter are covered some agile artifacts and activities that help to strengthen the process. Product backlog, grooming and burn-down chart are addressed in the scrum guide, while definition of ready, pre-grooming activity and the risk assessment process are additions in this model.

Nevertheless, the agile mentality is always present, and the objective is to integrate them in the proposed model, having in mind the requirements imposed by the IEC 62304.

6.5.1 Product Backlog

The product backlog is used to gather all the items known that a product needs, orderly, to optimize the value of the work of the development team. It works as the single source of requirements, and lists all features, requirements, bugs or defects that are iteratively being identified in the product development. Its conception is aligned with the agility provided by the scrum methodology, and so, the product backlog is in constant evolution, mirroring, at different stages the product needs.

6.5.2 Definition of Ready

Definition of Ready (DoR) is another artifact that is not covered by the scrum guide, but it is included in this development model and essentially creates another layer for verifying quality before the development starts.

Similar to DoD, DoR defines a list of requirements that a product backlog item must have before the team implementing it in the next sprint. These requirements are addressed in the pre-grooming activity, addressed in the next section.

6.5.3 Pre-grooming

This activity is an adding to the process, specially thought for this development life cycle model. As the name says it occurs before the grooming take place.

The pre-grooming essentially is a collaborative approach to define acceptance criteria. The product owner, a quality member and the tech lead are present, and together discuss the acceptance criteria to a requirement previously defined by the product owner. Together, this small team applies the DoR in order to determine if a user story is ready for grooming or not.

The main advantage of having a multidisciplinary team doing this assessment is to integrate different visions and to assure that the business vision of the product owner is wholly aligned with technical requirements. Furthermore, by having a quality team member present in the pre-grooming, this element makes sure that the defined acceptance criteria can be translated into test cases, ensuring the feasibility of the tests. By guaranteeing this, the point 5.7 “System testing” of the IEC, system testing, start to gain strength.

Also, the quality works closely with both the product and the development areas, improving the process that was identified through the interviews.

6.5.4 Grooming

Product backlog grooming, also known as product backlog refinement, is an activity where all the team reviews the backlog items. In the first idealization of the scrum guide backlog grooming was not considered an official element of scrum, but from 2011 on, the scrum guide addresses it. This activity is integrated in the life cycle and is scheduled together with the other scrum activities such as sprint planning or sprint review.

Product backlog grooming occurs one day before sprint review and has the goal to review all the backlog items, to prepare them for the next sprint planning. In this collaborative process of reviewing the backlog items, there can be added some details to user stories and are made time

estimates for each pending task. At this moment, the pre-grooming activity already took place, which means that every user story is ready to be groomed, time estimated and prioritized.

6.5.5 Burn-down chart

A burn-down chart is a tool used in this development life cycle model that has the objective of monitoring the team progress towards its goal. It represents the work left to do in the sprint time versus the remaining time. The main advantage of using this type of graphs in an agile environment is that they can provide information whether planning is being carried out, and, according to that information, the team can readjust the planned workload. The tasks that will not be able to be addressed during the sprint time are moved back to the product backlog.

6.6 Risk Assessment

Part 7 “Software risk management process” of the IEC 62304 is entirely dedicated to the software risk management process. Since risk management is of extreme importance when addressing critical systems like healthcare development, this activity must not be neglected. Thus, the proposed model includes a template for performing risk assessment, as shown in table 12.

This risk management assessment helps in the first instance to identify if a risk is of type project, technical or business. In terms of project risk, it defines the risk relating to compliance with the project plan, which can possible lead to a delayed delivery or other internal problems, such as increased costs or human resources. Technical risks are related with threats that may affect the software development process or quality. Moreover, business risks are related to scope of the project and its viability in the market. From that point it is identified risk severity, likelihood and impact. According to the previous points, mitigation actions and correction actions are defined to solve that hazardous situation as soon as practicable.

Also, an important aspect defined in the template is the traceability maintained between the correction action and a potential functionality implemented to correct it. By providing a link to navigate directly to that functionality, is kept a visible record.

Further development on system traceability is addressed in chapter 6.7.6.2.

Table 12 - Risk assessment template

Risk Type	Risk Description	Risk Severity	Risk Likelihood	Impact	Mitigations	Warnings	Correction
Project | Technical | Business	Small description of the risk	Acceptable | Tolerable | Undesirable | Intolerable	Improbable | Possible | Probable	Low | Medium | High | Extreme	Actions to mitigate the risk	What can go wrong	Actions to resolve the risk | link to functionality

Given the agile nature of the model and the criticality of defining and maintaining a risk assessment process, the initial risk matrix is defined by the entire team before initiating the first sprint of the project. Then, in every sprint review the risk matrix is revised, and the previously identified hazardous situations are reviewed and checked if the mitigations and corrections were successfully

applied. Then, if needed, new risks are identified and added to the matrix, making it highly iterative and collaborative.

6.7 Definition of the software development process

6.7.1 Architecture

In an early stage of development, in a collaborative effort by the entire team, an architectural document is created, assuring that it contains all the necessary components for the software.

Sustained by principles of modularity, scalability, security and traceability the architectural document contains all the software pieces that are part of the solution: visual components representing the frontend of the software and all the data layers compounded by the databases and the server, representing the backend of the software. Also, the modularity between the components facilitates the process of risk management.

The connections between all the components and the way they communicate with internal or external services are represented in a diagram. Besides this, it is written a document containing all the technical details of the software components, as well as their purpose.

The idealization of this document was based on the initial identified requirements. Of course, and since this activity is integrated into an iterative and incremental process, the document is itself in constant discussion and evolution. The initial version set the base for the beginning of the development, but as requirements grown and eventually change, the architectural document changes too, being reviewed at each sprint planning until a final version is defined.

All these steps are following point 5.3 “Architectural design” of the IEC 62304, proving its validity.

6.7.2 Quality Assurance

Quality assurance (QA) is a critical activity in any type of software, but specially in this work and looking to the IEC 62304, it assumes an undeniable dimension. The general outcome of the conducted interviews has validated the importance of a well-defined quality assurance process as well as some changes in the existing process and the way that the quality team is allocated to a project and its perceived by the other team members.

Thus, this model purpose a different perspective regarding quality and has the main objective of integrating, seamlessly, different quality assurance activities into an agile process.

6.7.2.1 Behavior driven development (BDD)

For the scope of this project and with the goal in mind of achieving a good and transparent relation between acceptance criteria and the tests performed by quality assurance, a software development technique is used, called behavior driven development (BDD).

BDD is strongly adaptable to an agile environment since its main focus is to foster collaboration between team members and narrow the gap between the business side and the development side. For achieving this goal, using a BDD approach is closely related with the pre-grooming activity, described in detail in chapter 6.5.3. By joining collaboratively, the product owner, the tech lead and the quality assurance member, tests are conceptualized in an early stage of the development

and its ensured that all the business requirements are being met. As so, BDD enhances the usage of agile methodologies at the same time that focus strongly in achieving automatic testing systems.

All the tests written in a BDD environment are written in a domain specific language, which helps in describing business need, without need to enter in technical implementation details. This can be achieved by writing the tests in a language format called Gherkin, which is closely related with user stories. When writing test cases in Gherkin, a template similar to user stories is used:

"Given [initial context of the system]

When [describe an event or action]

Then [expected outcome or result]"

Using this template, helps to translate the business need, according to their acceptance criteria and use case, into a test case. Using this declarative style when writing tests describe the behavior of the user when wanting to achieve an expected outcome, improving the readability of the feature.

An important premise to take in consideration also when writing tests is the usage of happy path and sad path. Happy path focus on describing the functionality when the software reaches the desired output and has a positive result. In contrast, sad path ends prematurely and focus on what can go wrong in the testing scenario.

6.7.2.2 Types of tests

When developing critical systems where each error can have a potential huge impact testing becomes more important than ever. Besides it the IEC 62304, in the points 5.6 "Integration and integration testing" and 5.7 "System testing", demands that at least tests at level of integration, system and regression are implemented and evidences of its implementation are shown. In this model, beyond these three types of tests component and acceptance tests are implemented, following the most common test model.

All these types of tests are related with a development phase. For testing the requirements created, acceptance tests are created; for performing a functional analysis of the entire system, system tests are created; for testing the technical design and ensuring that all component is visually correct, integration tests are created and finally to test the developed code, component tests are created.

All these types of tests are automatically triggered and generate a detailed report, creating evidence of its execution.

6.7.2.3 Definition of the quality assurance process

First, the proposed model changes the paradigm of how the quality team is integrated in the scrum team and process. So, the quality team is now an active integrant of the development process and is present in all stages, from requirement specification to design conceptualization. Quality activities are present each time a development is made, rather than just being present at the end.

Since this is integrated into an iterative agile process, the different stages where the quality team have a special contribution are going to be approached from a sprint perspective:

- Before starting the sprint – As previously explained in the chapter 6.5.3 in the beginning of the sprint exists a pre-grooming activity. Here, in a joint effort between quality, product and development the business need is depicted and deconstructed into requirements, which in turn have a set of acceptance criteria. These acceptance criteria are then translated to

Gherkin, the language used to write the tests. All the scenarios generated from the QA team are based on the premise of happy path and sad path, where are tested both the cases where a positive result is achieved and the cases where are some error or a deviation in the workflow occurs. In this pre-grooming the product owner defines the happy path, based on business requirements and the QA team defines the sad path. This completes the process before the user story jumps to the active sprint.

- During the sprint – At this stage, the QA element is responsible for programming automatic end-to-end tests. In terms of testing the software, parallel to end-to-end tests, unit and regression tests are being implemented by the development team. Whit this type of tests scenarios covering the entire workflow of the software are tested. The main objective of automatic tests is to replace the deprecated practice of manual testing a functionality. Automating the process makes it much less time consuming and less prone to eventual error. Every time that a member of the development team integrates some code in the development branch, the automatic tests are triggered automatically. Then, if the automatic test run successfully means that the functionality can be integrated with the rest of the solution. If not, when a bug is detected, an issue is created in the product backlog, of the type "defect", and is associated with the user story in question. To correct the bug found during the sprint the team must analyze it and assess the cost of correcting it, during the sprint and making sure that this not impact the final sprint goal. If the bug is decided not to correct at the moment, the created defect will be transferred to the backlog, to be addressed in the next sprint. This process is complying with the point 6.3 "Implementation of modifications" of the IEC 62304.
- End of sprint – At the end of the sprint the QA element is responsible for verifying if the definition of done for the developed user stories are met. If a certain story does not meet all the DoD that story is not considered complete and is transferred back to the backlog. Then, with all the validated increments at the end of the sprint a new release is created. This new release is a new distribution of a final software version, representing the culmination of various iterations.
- During release – At this stage, and since a new release is created and deployed, the objective of the QA element is to execute the same tests performed during the sprint, but in a release environment. Executing this new set of tests allows verifying if all the user stories of this release were successfully integrated with the previous versions of the software. Also, this last stage tests the system in a release environment, which is fundamental for complying with the point 5.8 "Release" of the IEC 62304.

Since all these steps are performed previously to the software is deployed in the client, it is ensured sufficient tests coverage to provide quality guarantees. It is also important to note that every time that an automatic test runs it generates a full report on the Jira page. This report is presented in the page of the associated user story and details the overall execution status, what test cases have passed and what test cases have failed.

6.7.3 Design System

With the objective of document and follow design patterns, to respond to the requirements imposed by point 5.4 "Detailed design" of the IEC 62304, a design system is an important aspect of this model. The design system implements a system of reusable user interface (UI) components, to create a coherent user interface across the entire software. Ultimately, this process creates a visual identity, unique to each software.

The process of building these interfaces is also a collaborative and, in the same line of most points approached in this model, an iterative process. This iteration is characterized by building a UI in the first instance. Then that interface is reviewed internally by the team, where is checked if the interface is consistent with the desired requirements and functionalities. If so, the interface is tested, where further improvements can be implemented. Since this design system is stored in a storybook, it is possible to visually document the interface and access to its source code. By using this storybook is possible isolate different visual components and develop them individually. This way is easier for the development team to copy and paste the code of each interface and quickly integrate it into the project. Then the last stage is distributing the interface to be reused in different contexts across the project.

Under this design system is used a component library. This library stores all the parts that compound the software and allow the designers to improve efficiency when designing interfaces. Usually, a component library follows atomic design, which is a methodology used to create web pages. This methodology introduces the concept of atoms, molecules, organisms, templates, and pages. The atoms represent the smaller component in a web page, like a search box or a button. Then the design aggregates more and more elements, reaching the final composition, which is the entire page (Catakli, 2018).

After achieving a visual identity, through the component library, there are created mockups, which are visual representations of a specific page or functionality. These mockups are linked with a user story and it must be guaranteed consistency between the user story objectives and acceptance criteria and the designed functionalities of the page.

In conclusion, the usage of a design system enables ease of use for the user with a functional innovation in terms of technology, complying with all the business rules and requirements.

6.7.4 Version control system

As reviewed in the literature a version control system is a mandatory activity in the software development process (Loeliger & McCullough, 2012).

For the scope of this project, the IEC 62304 requires the usage of verification activities for the different software units, and specifically code verification. The activities that are integrated into this chapter of the proposed model (git flow, pull request and code review) are specially designed to aim that requirement. Additionally, in the part 8 “Software configuration management process” of the IEC 62304, it is required to have a change control and a configuration status accounting. Since this version control system allows the definition and storage of different software versions, there is ensured compliance with those requirements.

6.7.4.1 Git flow

In this model, beyond the implementation of a version control system that can be applied with agile methodologies, responding to it iterative nature, some git flow activities are implemented. The focus of integrating these activities in the process is, as seen in previous points, not to misrepresent the methodology, but increasing its applicability in a critical environment. In a nutshell, git flow is a branching model created for git that can foster collaboration between team members, as explained in chapter 2.3.2.

This model uses git flow to control the development process. The rationale behind it is to create a new branch for each user story that is being developed. This way, it is possible that several user

stories are being developed at the same time, for different developers, without integration issues. After the user story is completed the developed code is integrated from the new branch to the "develop" branch, which will contain all the developed code at the moment. When this happens the automatic testing process, explained in the chapter 6.7.2.3 is triggered, assuring that the integrated feature is bugs-free and approved by the quality assurance process.

The last stage of the git flow process is to integrate the code from the "develop" branch to the "release" branch. At this moment, the code that enters this branch is verified, having gone through different types of tests. In the sprint review, the increment presented by the team represents the code that was integrated into the "release" branch. This code verification process is an important point when complying with part 5.8 "Release" of the IEC 62304. Figure 3 (Frauenholtz, 2016) addresses the code verification process workflow, showing when to create a pull request and when code review is performed.

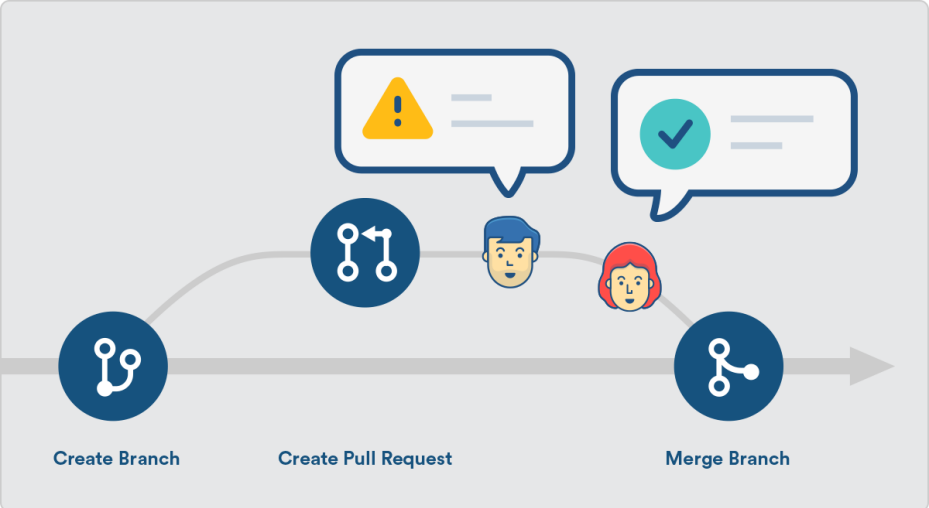


Figure 4 – Git flow process (Frauenholtz, 2016)

6.7.4.2 Pull Request & Code Review

Every time some code is integrated into the "release" branch, a pull request (PR) is created. This way, all the team members are notified that the developer wants to integrate a new code. The created pull request contains a brief description of the implemented change, as well as the correspondent code. Using this tool creates an additional layer of code verification and allows the team to perform code review. Through the pull request, the other member of the team can analyze and make suggestions to the developed code. After the code is reviewed, the pull request can be accepted or declined.

Also, this model imposes some rules to approve a pull request. At least two members of the team must approve the PR and one of them must be part of the default reviewers to all pull requests. Furthermore, the description of the PR must be clear and perceptible, referring to any documentation that has been influenced. Meeting these two rules, the PR is accepted and then the code is integrated in the "release" branch.

6.7.5 Automation

In this chapter, are going to be approached automation techniques that are integrated into the new model. Automation techniques can be a powerful advantage to improve the entire development process, since they allow a greater level of scalability in terms of tests and code verification, without the need to increase complexity or team size. The implemented automation techniques are of special emphasis when complying with parts 5.6 “Integration and integration testing”, 5.7 “System testing” and 5.8 “Release” of the IEC 62304. The first automation process visible is in tests and code verification, previously explained in chapter 6.7.2.3.

Another software development practice that is integrated into this proposed model is continuous integration and delivery. Continuous integration, commonly known as CI, is strongly connected with tests automation. CI allows that every time a developer introduce new code to the main branch a process of building the application and running the integration tests is triggered, ensuring that all new modifications do not conflict with previous system versions.

Continuous delivery, also known as CD, is an automated release process, meaning that the application is ready to deploy new versions at the customer, by just clicking a button. Implementing CD guarantees that the code is always in a stable state, and besides running integration tests, it also saves the necessary configurations required for the code to be deployed in a production environment.

Implementing CI and CD also ensures that a changelog file is created where a log of all versions is maintained, with information regarding all the modifications and configurations implemented since last version.

6.7.6 Documentation

Documentation is an essential point in the software development life cycle. Although the Agile manifesto states that it should be produced “working software over comprehensive documentation”, this does not mean that documentation should not be considered. Analyzing the IEC 62304, many requirements are related to creating and maintaining several types of documentation, from the development process itself to risk management documentation.

Creating comprehensive documentation also have a set of advantages like supporting internal team decisions and serving for audit purposes, and mandatory aspect for the scope of this model.

Therefore, documentation is being developed in every stage of the process, and, according to agile principles it is being created and maintained in a collaboratively perspective.

6.7.6.1 Information Repository

For storing all the documentation and make it accessible, an information repository is used in this model: Confluence. Confluence uses a hierarchical tree structure to present all the pages, making it easier for the team to reach any page and navigate between them. Also, some rules to write documentation were defined by the team to maintain the coherence between different pages and documents.

6.7.6.2 Traceability

In software engineering, traceability refers to the ability to trace different items in the development life cycle, to keep visible records of what is being developed. Traceability can be essential when

developing critical software and is a vital aspect addressed by the IEC 62304, especially in terms of risk management. Using a good traceability system can contribute to a better understanding of the entire system, while highlighting different aspects of it. In this life cycle model, the used tool to achieve traceability is Jira, an issue tracking management system.

Jira is used together with Confluence and git, allowing to integrate both documentation and code into a single system. Besides, it is also suitable to incorporate in an agile environment because it empowers the usage of scrum by creating scrum boards with product backlog and sprint control.

One evidence of the traceability present in this model is the identity card of a software component (chapter 6.4.4). Only by consulting this document is possible to see the requirements associated with the software component, their acceptance criteria and test cases for every condition. All this data is connected with the correspondent Jira page. Another relation that is kept is between the requirement and user stories. Once the requirement page is opened it is possible to verify all the related user stories and navigate through them.

Above all, implementing these traceability systems allow a more straightforward response when change impact is analyzed and also helps in verifying if a certain functionality complies with the defined requirements.

6.8 Problem resolution process

The software problem resolution process is addressed in point 9 ”Software Problem Resolution Process” of the IEC 62304, where items like preparing problem reports and investigating the problem are covered. To have a mechanics suitable to predict and control issues that can arise after the software is deployed into the customer, a system monitoring process is another addition to this proposed model.

This process of system monitoring consists of two main aspects: logging and audit. Implementing a logging system allow the team to keep records and monitor eventual applications crashes, bug that may appear, failure of servers or data loose. All these types of errors are stored in a specific tool for this purpose, allowing a quicker and better team response to this type of situation. An audit system is similar to logging and has the same objective. Still, the critical difference is that implementing an audit system stores software events and information about what, when and who triggered that specific event. Some of these events include user login and logout or configuration events, for instance.

All these event records are stored in an organized way and presented to the team in a visual monitoring tool. From these records, it is possible to create new items into product backlog, aiming to address the problem. The new items inserted are prioritized by the team and fixed, accordingly to the prioritization and the remaining sprint time. Creating a mechanism to handle process resolution is facilitated by the iterative nature of agile environments, and, therefore, is another activity that can be integrated over the normal course of software development.

Summarizing, this chapter defines a model for software development life cycle, supported by the core scrum activities and principles that aim to create evidence and processes capable of giving response of the requirements imposed by the IEC 62304 for a software to be classified as a medical device. All stages of development are addressed, from the market-based product vision to the final iterative increment developed at each sprint.

7 Conclusion and future research

The main objective in developing this work included studying the integration of agile software development methodologies into a healthcare solution, classifying it as a medical device.

The seamless creation of evidence, in the form of documentation, software activities and auxiliary processes, was integrated along agile principles and methodologies, creating a model capable of responding to a strict and regulated environment.

The first and primary driver when initially conceptualizing this work was the goal of classifying a software as a medical device, having in consideration its safety critically nature and impact in many lives, from patients to healthcare professionals.

It was essential an in-depth study of European regulations and international standards that govern the software development practices that should be followed and understand how these could be compatible with existing software development methodologies. Furthermore, another essential step on perceiving how these two problematics can be linked was to study the impact that methodologies are having in software development, and specifically in safety critical software. Reviewing this literature paved the way to respond to RQ1, where it was concluded that agile methodology is more capable in responding to change and in dealing with changing nature of requirements. With the goal in mind to use an agile methodology and modulating it to the requirements imposed by international standards, Scrum was further studied and used as principal methodology to define the process.

To further advance and taking into account that this work was developed in an organizational context, this study encompassed the understanding of the perceptions of development teams in using software development methodologies. To achieve this goal, interviews with different team members with different roles were performed, addressing several topics. First, mapping the processes that are currently used by the teams and what are its strengths and weaknesses was mandatory. It was concluded that most of the teams and processes are not following an agile path, which can be disadvantageous. This conclusion strengthened that adopting an agile methodology is considered an improvement and is better suited to the business area. Thus, to understand how to construct agile processes around the international standard needed to classify a software as a medical device, another objective of the interviews was to map and describe what is the team's perception on adopting agile methodologies, what strengths and weaknesses these methodologies bring into developing software and how these methodologies can be integrated in a healthcare environment.

Performing a qualitative study with in-depth interviews, alongside with the developed model have responded to RQ2. An important conclusion retrieved from the interview process was that the usage of a methodology helps in the definition of the process, and it was this mindset that was adopted when developing the solution.

Starting by defining how agile methodology Scrum will help in organizing the team and the main structure of work, allowed to understand that some software development auxiliary activities and processes could be seamlessly integrated, and together with agile practices allow compliance with the requirements for a software as medical device.

Overall, the development of this work resulted in an agile based model that can be implemented for achieving software compliance and classify it as a medical device. This model includes activities aligned with the scrum vision for software development, like the pre-grooming stage in the life cycle. Also, the product vision and requirement analysis activities are vital in understanding

the business needs and translating them into workable items that are integrated in an agile process, taking into consideration the risk classification for the software as medical device. Changing the quality assurance responsibilities and procedures enhanced the general software quality and verification activities. Further, there is included a risk management process as well as other software development processes that are developed and maintained during a sprint. In these transversal processes are included the implementation of a version control system, the development of documentation and traceability systems and the implementation of audit and logging systems to prevent and resolve software problems. All these strategies are implemented together with existing concepts of scrum, like sprint, scrum events and people. The definition of this life cycle development model assures the seamless compliance with the requirements imposed by the IEC 62304 for classifying software as a medical device.

Although this work has been developed in a relatively long period, the process of creating and placing healthcare software in the market is very long and complicated. Thus, the implementation of the defined model could not be practically tested and verified across the entire software life cycle. There are already some activities and processes of the proposed model that are being implemented in an organizational context, but without carrying out an audit to the software and documentation is not possible to prove its complete compliance with the international standards.

Also, future work can be introduced by studying in detail what are the requirements imposed by regulatory authorities when classifying software as a medical device. The proposed model responds to a macro level to the international standards requirements. Still, a lack of detail can remain, especially when trying to comply with medical devices of class B or C. Also, the international standard in question, the IEC 62304 can be a little vague and open to interpretation and discussion, leading to different approaches and visions of solutions. Thus, the needed level of detail required by regulatory authorities could not be achieved by this model.

The research performed in this work, can work as a foundation for future research, especially in analyzing in detail all the subclasses of the international standard IEC 62304 and verify if the proposed model is capable of giving response to each one of them. Also, a practical implementation of the proposed model is needed, mirroring the defined team, member's responsibilities and processes. This practical implementation must be carried out throughout the entire process of analyzing the market in terms of business opportunities to create a software that will work as a medical device, and the whole life cycle of development. Them, some measures in terms of costs and project roadmap compliance should be analyzed to perform changes to the model and prove if it is beneficial in an organizational context. Ultimately, the software should be subject to an audit, to assess it compliance and response to the risk classes A, B or C.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

Balaji, S., & Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management, 2*(1), 26-30.

Bannerman, P. L. (2008). Risk and risk management in software projects: A reassessment. *Journal of Systems and Software, 81*(12), 2118-2133.

Bartoo, G. (2003). Risk management [medical devices]. *IEEE Engineering in Medicine and Biology Magazine*.

Bass, J. M. (2014). *Scrum master activities: process tailoring in large enterprise projects.* Paper presented at the 2014 IEEE 9th International Conference on Global Software Engineering.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Jeffries, R. (2001). Manifesto for agile software development.

Catakli, T. (2018). Why Should You Use a React Component Library for your Project? Retrieved from https://medium.com/@timurcatakli/why-should-you-use-a-react-component-library-for-your-project-aa530a05e038

Chacon, S., & Straub, B. (2014). *Pro git*: Springer Nature.

Commission, E. (2009). *Implementation of Directive 2007/47/EC amending Directives 90/385/EEC, 93/42/EEC and 98/8/EC*. Brussels

Statement regarding Use IEC 62304:2006 "Medical
device software -- Software life cycle processes" (2015).

Fillmore, R. (2019). “Is your software a medical device?”. Retrieved from https://www.raps.org/news-and-articles/news-articles/2019/3/is-your-software-a-medical-device

Foddy, W., & Foddy, W. H. (1994). *Constructing questions for interviews and questionnaires: Theory and practice in social research*: Cambridge university press.

Frauenholtz, A. (2016). 5 elements of a perfect pull request. Retrieved from https://www.atlassian.com/blog/bitbucket/5-pull-request-must-haves

Fu, K. (2011). Trustworthy medical device software. *Public Health Effectiveness of the FDA, 510*, 102.

Galin, D. (2004). *Software quality assurance: from theory to implementation*: Pearson Education India.

Ge, X., Paige, R. F., & McDermid, J. A. (2010). *An Iterative Approach for Development of Safety-Critical Software and Safety Arguments*. Paper presented at the Agile Conference, Orlando.

Geremia, F. (2018). Quality aspects for medical devices, quality system and certification process. *Microchemical Journal, 136*, 300-306.

Gill, P., Stewart, K., Treasure, E., & Chadwick, B. (2008). Methods of data collection in qualitative research: interviews and focus groups. *British dental journal, 204*(6), 291-295.

GitHub. About pull requests. Retrieved from https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

Software as a Medical Device (SaMD): Key Definitions, (2013).

Hrgarek, N. (2012). *Certification and regulatory challenges in medical device software development*. Paper presented at the 2012 4th International Workshop on Software Engineering in Health Care (SEHC).

Huhn, M., & Zechner, A. (2010). *Arguing for software quality in an IEC 62304 compliant development process.* Paper presented at the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation.

Humphrey, W. S. (1988). *The software engineering process: definition and scope.* Paper presented at the Proceedings of the 4th international software process workshop on Representing and enacting the software process.

Lee, G., & Xia, W. (2010). Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility. *MIS quarterly, 34*(1), 87-114.

Lee, I., Pappas, G. J., Cleaveland, R., Hatcliff, J., Krogh, B. H., Lee, P., . . . Sha, L. (2006). High-confidence medical device software and systems. *Computer, 39*.

Loeliger, J., & McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development*: " O'Reilly Media, Inc.".

Matharu, G. S., Mishra, A., Singh, H., & Upadhyay, P. (2015). Empirical study of agile software development methodologies: A comparative analysis. *ACM SIGSOFT Software Engineering Notes, 40*(1), 1-6.

Migliore, A. (2017). On the new regulation of medical devices in Europe. In: Taylor & Francis.

Mnkandla, E., & Dwolatzky, B. (2006). *Defining agile software quality assurance.* Paper presented at the 2006 International Conference on Software Engineering Advances (ICSEA'06).

Moniruzzaman, A., & Hossain, D. S. A. (2013). Comparative study on agile software development methodologies. *arXiv preprint arXiv:1307.3356*.

Parliament, E. (2017). *Regulation (EU) 2017/745 of the European Parliament and of the council, on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and*

Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC Official Journal of the European Union Retrieved from https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32017R0745&from=EN

Parliament, E. (2020). *Regulation (EU) 2020/561 of the European Paliament and of the council of 23 April 2020,*

amending Regulation (EU) 2017/745 on medical devices, as regards the dates of application of certain of its provisions Official Journal of the European Union Retrieved from https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32020R0561&from=EN

Quinn, P. (2017). The EU commission's risky choice for a non-risk based strategy on assessment of medical devices. *Computer Law & Security Review, 33*(3), 361-370.

Ranney, M. L., Meisel, Z. F., Choo, E. K., Garro, A. C., Sasson, C., & Morrow Guthrie, K. (2015). Interview-based qualitative research in emergency care part II: Data collection, analysis and results reporting. *Academic Emergency Medicine, 22*(9), 1103-1112.

Rasmussen, R., Hughes, T., Jenks, J. R., & Skach, J. (2009). *Adopting Agile in an FDA Regulated Environment*. Chicago, IL, USA.

Sale, J. E., Lohfeld, L. H., & Brazil, K. (2002). Revisiting the quantitative-qualitative debate: Implications for mixed-methods research. *Quality and quantity, 36*(1), 43-53.

Salmons, J. (2014). *Qualitative online interviews: Strategies, design, and skills*: Sage Publications.

Schwaber, K. (1997). Scrum development process. In *Business object design and implementation* (pp. 117-134): Springer.

Schwaber, K., & Sutherland, J. (2013). The scrum guide The definitive guide to scrum: The rules of the game. *SCRUM. org, Jul-2013*.

Scrum.org. (2020). The Scrum Framework Poster. In. Scrum.org: Scrum.org.

Sofaer, S. (1999). Qualitative methods: what are they and why use them? *Health services research, 34*(5 Pt 2), 1101.

Sorenson, C., & Drummond, M. (2014). Improving Medical Device Regulation: The United States and Europe in Perspective. *The Milbank Quarterly, 92*(1), 114-150. doi:10.1111/1468-0009.12043

Standardization, I. O. f. (2006). IEC 62304:2006 Medical device software — Software life cycle processes. In.

Standardization, I. O. f. (2015). ISO 9001:2015 Quality management systems — Requirements . 29.

Standardization, I. O. f. (2016). Medical devices — Quality management systems — Requirements for regulatory purposes.

Standardization, I. O. f. (2019). Medical devices — Application of risk management to medical devices.

Sutton, J., & Austin, Z. (2015). Qualitative research: Data collection, analysis, and management. *The Canadian journal of hospital pharmacy, 68*(3), 226.

Weisert, C. (2003). There's no such thing as the Waterfall Approach!(and there never was)'. *Information Disciplines, Inc*.

Williams, L., & Cockburn, A. (2003). Agile software development: it's about feedback and change. *IEEE Computer, 36*(6), 39-43.

Zamith, M., & Gonçalves, G. (2018). Towards an Agile Development Model for Certifiable Medical Device Software.

APPENDIX A: Interview Script

Este questionário insere-se no contexto de desenvolvimento da tese no Mestrado de Engenharia de Serviços e Gestão. Tem como objetivo perceber a utilização de metodologias no processo de desenvolvimento de software e como as mesmas se enquadram com a natureza crítica do software na área da saúde.

Dados Socio-demográficos:

a. Idade, género
b. Experiência profissional

Entrevista:

- Qual o teu papel na empresa?

- Em que equipa(s) estás inserido?

- Quais as tuas responsabilidades diárias?

- A nível de processo de desenvolvimento de software, como descreves o processo que tu / a tua equipa utilizam?

    - O que corre bem ou mal neste processo?

- Que metodologias conheces que possam ser utilizadas no processo de desenvolvimento de software?

- Utilizas alguma metodologia?

    - Qual?

- Trabalhando numa área como o software em saúde, qual é a tua perceção do mesmo?

    - Encontras alguma distinção, em comparação com outra área de negócio?

- Na tua opinião, quais os pontos fortes e fracos na utilização de metodologias no processo de desenvolvimento de software?

- Como é que estas metodologias se adequam ao software em saúde?

- Como é que estas metodologias se adequam no relacionamento com outras equipas?

- Como é que estas metodologias ajudam a criar valor e qualidade no entregável final para o cliente?

- Em que medida a utilização de uma metodologia é importante para reconhecer e interpretar os requerimentos pedidos?

-Consideras importante a criação de documentação, durante o processo de desenvolvimento?

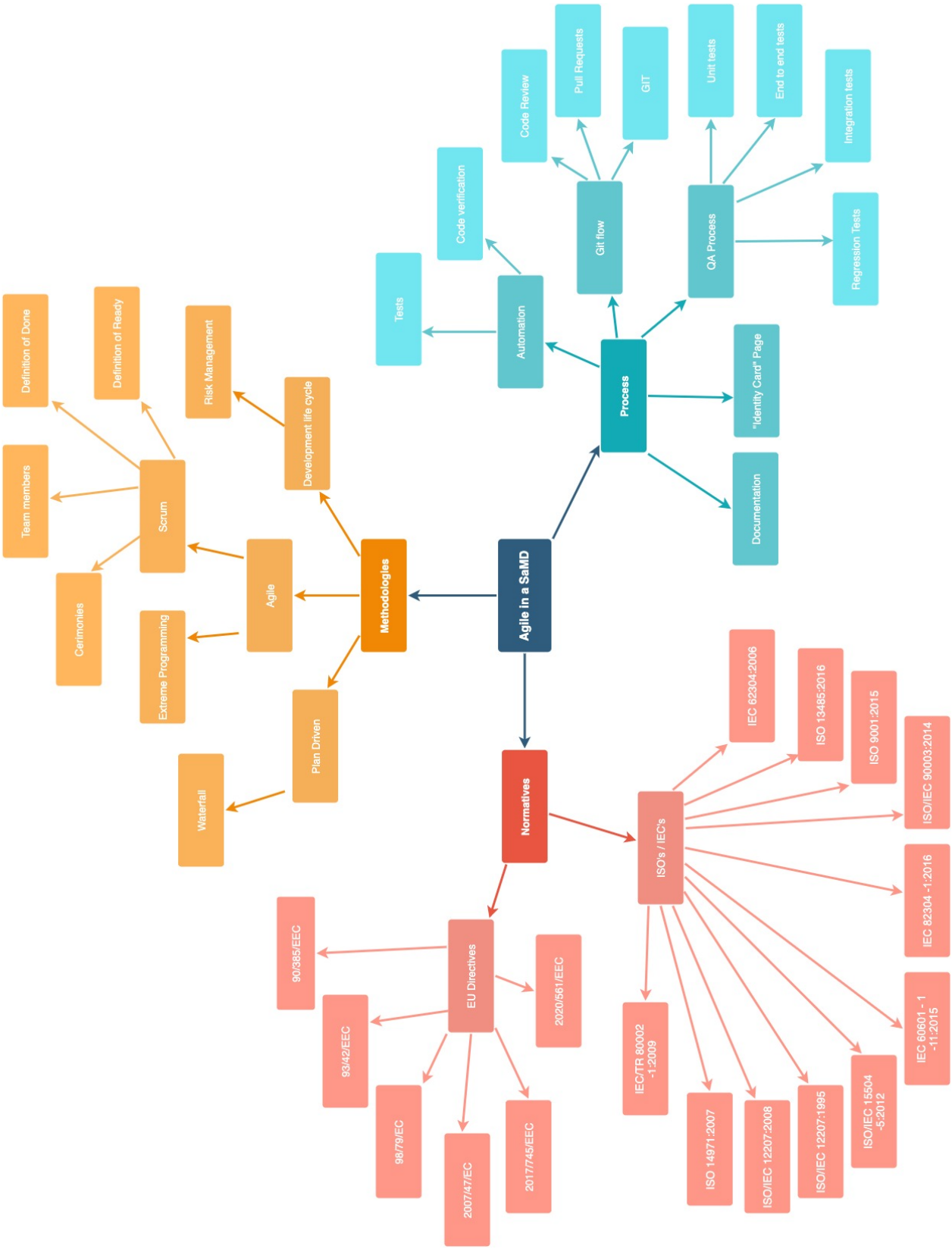
- Que tipo de documentação é criada, no processo de desenvolvimento atual?

    - Consideras esta documentação relevante?

    - Em que plataforma de informação é que esta documentação é criada?

-Em que medida a utilização de uma metodologia de desenvolvimento de software impacta a criação de documentação?

APPENDIX B: Mind Map



APPENDIX C: Summary of Software Documentation according to IEC 62304

Software Documentation		Class A	Class B	Class C
Main Class	**Subclasses**			
Software development process	5.1.1, 5.1.2, 5.1.3, 5.1.6, 5.1.7, 5.1.8, 5.1.9	X	X	X
	5.1.5, 5.1.10, 5.1.11		X	X
	5.1.4			X
Software requirements specification	5.2.1, 5.2.2, 5.2.4, 5.2.5, 5.2.6	X	X	X
	5.2.3		X	X
Software architecture	5.3.1, 5.3.2, 5.3.3, 5.3.4, 5.3.6		X	X
	5.3.5			X
Software detailed design	5.4.1		X	X
	5.4.2, 5.4.3, 5.4.4			X
Software unit implementation and verification	5.5.1	X	X	X
	5.5.2, 5.5.3, 5.5.5		X	X
	5.5.4			X
Software integration and integration testing	All		X	X
Software system testing	All		X	X
Software release	5.8.4	X	X	X
	5.8.1, 5.8.2, 5.8.3, 5.8.5, 5.8.6, 5.8.7, 5.8.8		X	X
Software maintenance process	6.1, 6.2.1, 6.2.2, 6.2.4, 6.2.5	X	X	X
	6.2.3		X	X
Software risk management process	7.1, 7.2, 7.3, 7.4.2, 7.4.3		X	X
	7.4.1	X	X	X