José Manuel Alarcón Roldán

Establishing Guidelines for Medical Device Software Development Using Agile - Case: Start-up's Infant Apnoea Monitor

Helsinki Metropolia University of Applied Sciences

Master's Degree

Health Business Management

Master's Thesis

29 December 2015

Helsinki Metropolia University of Applied Sciences

Author Title Number of Pages Date	José Manuel Alarcón Roldán Establishing Guidelines for Medical Device Software Development Using Agile: Start-up's Infant Apnoea Monitor 54 pages + 1 appendix 29 December 2015
Degree	Master's Degree
Degree Programme	Health Business Management
Instructor	Thomas Rohweder, Dr (Econ), Principal Lecturer

Software has become a prominent part of modern medical devices. In order to ensure safety of patients and users of medical devices, health authorities around the world have produced a number of regulations that control the development, manufacturing and sales of medical devices. Software which is part of a medical device must meet the same safety and quality requirements as the device itself.

In Europe, Directive 2007/47/EC regulates the development and manufacturing of medical devices. International standardization organizations have produced harmonized standards such as IEC 62304 – medical device software – software life cycle processes to assist the manufacturers of medical devices in obtaining regulatory approvals.

In recent years a new way to develop software known as Agile has emerged. Agile methods are based on an iterative and evolutionary software development life cycle. Although regulators do not mandate what software life cycle should be used, most of the regulations and standards assume a linear life cycle, such as waterfall.

The Agile practices emerge from a common set of values and principles, such as quality of the software, productivity of the development teams and customer satisfaction. In this thesis we discuss how these values align with those of health authorities and regulators around the world.

In this thesis we Introduce the Agile SW development practices in the context of a medical device company. We will analyze the European Medical Device Directives and international standards. We then propose a set of guidelines for the development of medical device software based on Agile practices while complying with the international standards.

Keywords	Agile, Medical Device Software, MDD, Software Development Life cycle, IEC 62304

Contents

Appendices

1 Introduction

A medical device is any instrument or apparatus used to diagnose, prevent or treat a medical condition. Medical devices vary in complexity, from a simple bandage to life sustaining equipment or sophisticated imaging devices such as MRI scans (Council Directive 2007/47/EC).

In order to minimize the risks that the usage of medical devices may impose to human health, manufacturing and merchandizing of medical devices is regulated by the competent authorities (e.g., the Food and Drug Administration in the US or the European Commission in Europe).

The number of medical devices functions performed by software is increasing greatly. It is not uncommon to find nowadays medical devices that are implemented purely by software. The safety regulations that apply to traditional medical devices apply also to the medical device software. Harmonized standards and guidelines exist to help manufactures of medical device software to comply with the requirements of these regulations.

During the last years a new way to develop software has appear known as Agile (Agile Alliance, 2013). It started as a niche concept on small areas of the software industry but it has quickly gained traction and it is nowadays used in several software development contexts. For Agile to be widely used in the medical device software industry it has to be adapted to fit the needs of this unique context.

In this thesis we will propose guidelines for the software development process for a new devices manufacturer willing to enter the medical device market. These guidelines will be based on Agile methodologies while complying with the medical device safety regulations.

1.1 Case context

Circular Devices Oy (hereafter “the case company”) is as recently established company that develops consumer electronics products with a focus on protection of the

environment, responsible sourcing of raw materials and production. They are developing PuzzlePhone, a modular, upgradeable and repairable smartphone. Alongside with PuzzlePhone, they are working on a vision based infant apnoea monitor intended to help on the prevention of Sudden Infant Death Syndrome (SIDS).

SIDS is defined as the sudden death of an infant less than 1 year of age that cannot be explained after a thorough investigation is conducted, including a complete autopsy, examination of the death scene, and a review of the clinical history. SIDS is the leading cause of death in otherwise healthy infants 1 to 12 months old (National Centre for Chronic Disease Prevention and Health Promotion, 2015).

The vision based infant apnoea monitor being developed uses machine vision algorithms to detect the baby's respiratory movements and monitor his respiratory frequency. If there's no breath for a set number of seconds (a condition called apnoea) it produces an alarm to alert parents or other caregivers.

1.2 Business problem, objective and intended outcome of this thesis

Any device which primary function is, as defined by its manufacturer, to prevent, diagnose or treat a medical condition is considered as a medical device. Therefore, it must comply with the relevant medical device regulations before it can be sold (MEDDEV 2. 4/1 Rev. 9 2010).

The infant apnoea monitor being developed by the case company is being advertised as a device that helps in the prevention of SIDS. Therefore, it must fulfil the requirements set by the competent authorities for medical devices. These apply both to the hardware and to the software components of the device.

The objective of this thesis is to produce a set of guidelines that the case company should follow when developing its medical device software. We will study the European Medical Device Directives (MDD's) that regulate the development of medical devices and analyse the international standards and industry guidelines concerning medical device software.

The intended outcome of this thesis is a set of guidelines covering all phases of the software creation process (including development, maintenance, configuration and risk

management) based on Agile that fulfils the safety and performance requirements set for medical device software. We will focus on the European market although most of what is being proposed here would be easily adapted to comply with the requirements from other regions regulators, such as the FDA in the US.

1.3 Research design

As we discussed earlier in this chapter, this thesis address a real business problem faced by the case company: the infant monitor being developed will be sold and advertised as a device that helps in the prevention of SIDS. Any device which primary function is to treat or prevent a medical condition is considered a medical device and hence it must comply with the safety and quality requirements set by regulators. These requirements apply to all the components of the medical device, including its software.

The objective of this thesis is to propose a set of guidelines for the software development process based on Agile methodologies while complying with the safety and quality requirements for medical devices. For the production of these guidelines, we will first study in depth the legal framework and standards that apply to medical device software. These guidelines will be shaped as well by the *best practices* identified during a literature review of published case studies from companies that apply Agile when developing medical device software as well as from insights obtained from the industry experts being interviewed.

The figure below represents the different steps of the research design used in this thesis:

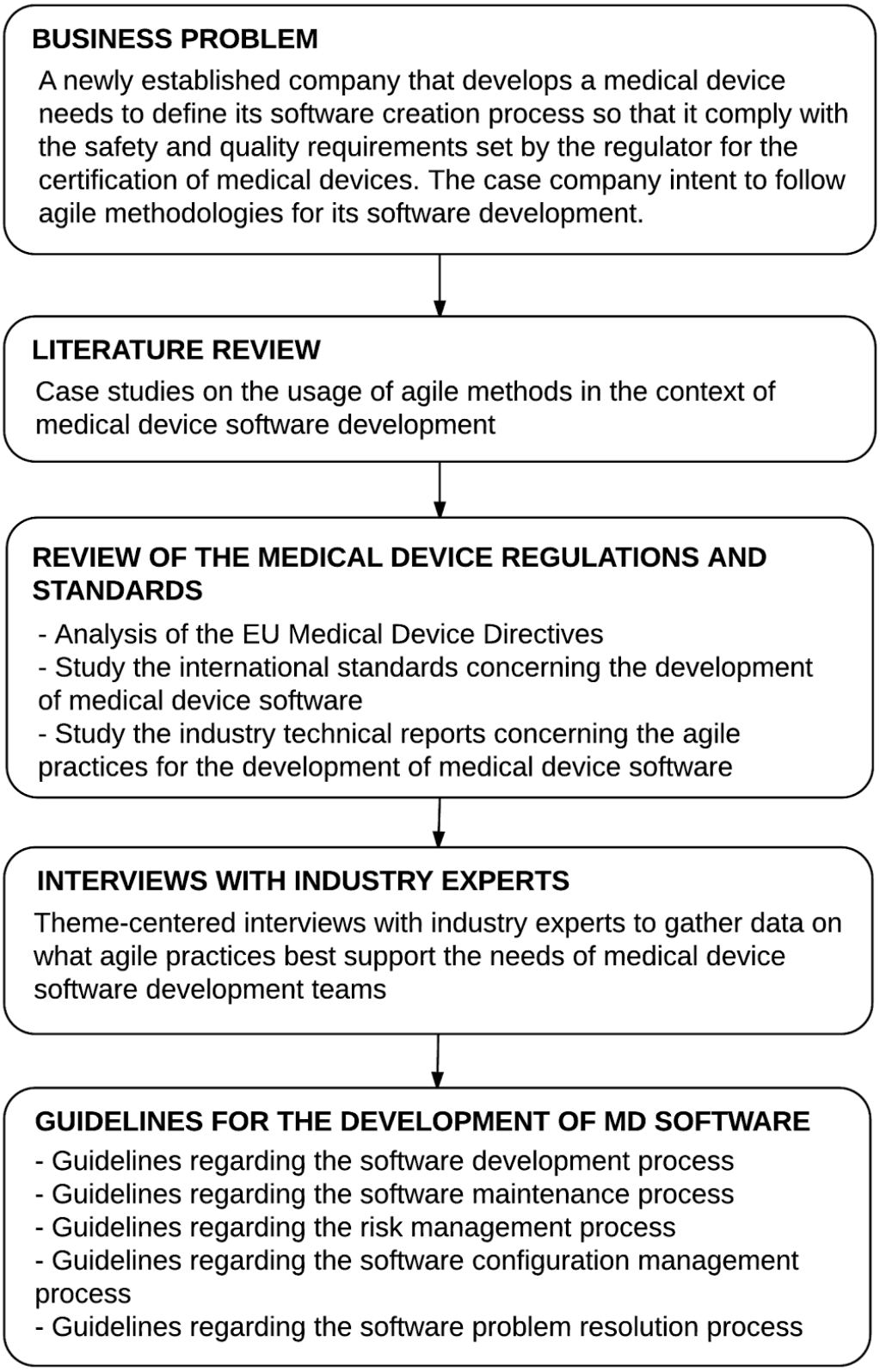

Figure 1.1 - Research design of this thesis

This study begins with the definition of the business problem of the case company as previously described. We will then introduce the Agile software development concepts and perform a literature review of some case studies in which Agile has been previously introduced in medical device software companies. We will study the relevant European Medical Device Directives, the corresponding international standards used to assess compliance as well as industry guidelines that assist medical device manufactures on the implementation of these standards. Some industry experts will be interviewed to understand how they apply Agile practices at their organizations, what Agile practices enhance safety and quality when developing medical device software and what hurdles they found when adopting Agile practices. After that, a software development process proposal will be presented. The purpose of this proposal is to serve as a guideline for the case company for the development of its own medical device software in a way that it benefits from the productivity and software quality improvements achieved when using Agile methods while demonstrating compliance with the medical device regulations.

2 Agile SW Development Practices in the Context of Medical Device Software Development

2.1 Agile vs. plan driven development life cycles

Software development teams are under pressure to develop and deliver software faster not only meeting their customer's requirements but exceeding them. Traditional plan-driven, document intensive, software life cycles such as Waterfall or V-Model, where planning of the software requirements is being done up front, followed by architectural design, module design, implementation, testing, system integration and validation, which each of these steps being done following a sequential order, make it very difficult and costly to accommodate to changes during the latest phases of the development process.

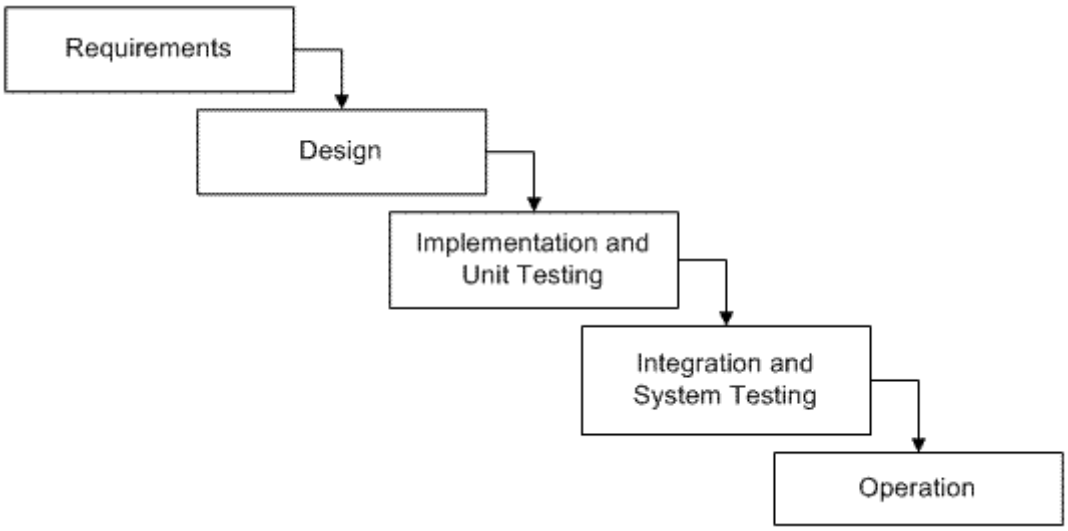

Figure 2.1 - Waterfall life cycle model

In a sequential development life cycle, each activity is performed in a certain, fixed order. The next one is not started before the previous one is completed (e.g., requirements definition precedes architectural design; hence the requirements need to be fully defined before the design of the architecture can start, and so on).

Iterative software development appeared to offer a solution by aiding faster development of systems. In an iterative life cycle, these activities are performed several times during the development process. In an evolutionary life cycle, a system is developed in a number of consecutive builds. An evolutionary strategy is similar to an

incremental strategy in that after each iteration, an increment of the system is being delivered, but differs in acknowledging that the user needs are not fully understood and all requirements cannot be defined up front. In this strategy, user and system requirements are partially defined at the early stages of the development. Then they are refined in each successive build. Agile is a subset of these iterative techniques (Agile Alliance, 2013).

The flow diagram below illustrates the evolutionary approach as described by AAMI in their TIR45:2012:

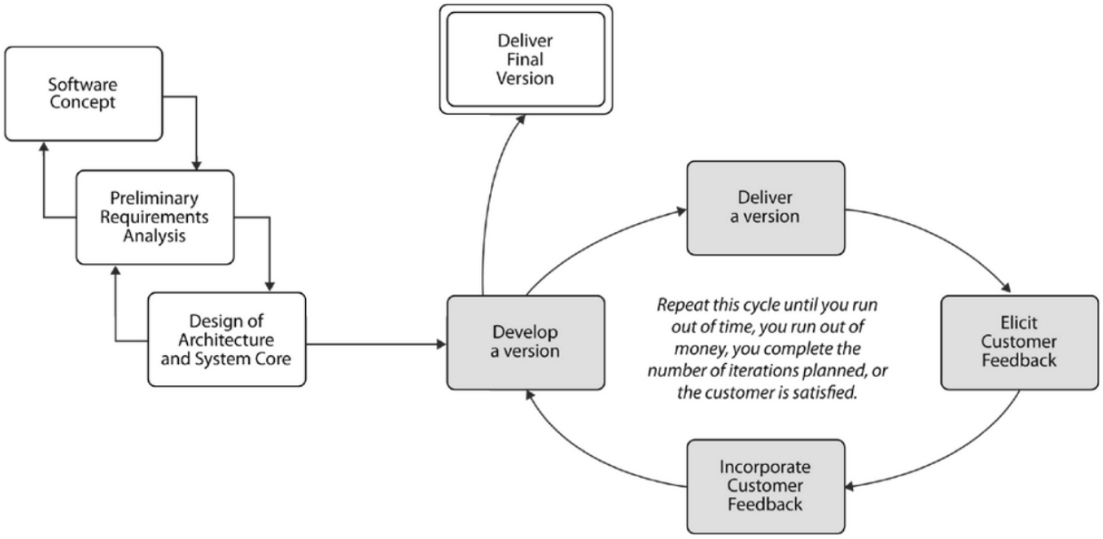

Figure 2.2 - Evolutionary life cycle model

The evolutionary approach puts a bigger emphasis on customer feedback, which is an essential component of Agile. In this model, there is a preliminary requirement analysis followed by an initial, high level design and architecture definition which is “just enough” to get started with the version build cycle. This cycle is being repeated until the customer is satisfied or we run out of time or money (whichever occurs first).

Agile was developed in response to quality and efficiency concerns present in existing software development methodologies. Agile bring benefits to software development teams in the following areas (Association for the Advancement of Medical Instrumentation, 2012):

- Continuous focus on safety, risk management and delivering customer value through backlog prioritization, planning practices and customer feedback

- Continuous assessment of quality through continuous integration and testing
- Continuous improvement of the software development process through retrospectives and team accountability
- Continuous focus on “getting to done” and satisfying quality management stakeholders through the regular completion of activities and deliveries

It is apparent that any organization, including those developing software in a highly regulated and controlled environment such as a medical device context, can benefit from these improvements. Because Agile initially grew from the IT industry where human safety and risk management were not of primary importance, there are some concerns about Agile compatibility with the regulated world of medical device software development. Fortunately, as we shall discuss later in this chapter, the fundamental nature of Agile is to be adaptable to the context in which it is being utilized, encouraging its practices to be adapted to this particular context in ways that they are compatible with the needs of the safety-critical medical device software development world.

Agile methodologies (Scrum, Extreme Programming, Kanban, etc.) are made of Agile practices, such as sprint planning, daily stand ups, product backlog, pair programing and so forth (Mc Hugh et al., 2012).

Agile methodologies such as Scrum and XP are becoming increasingly popular in traditional software development projects. Some case studies exist where Agile practices have being used when developing software for safety critical applications (Drobka et al., 2004), (Grenning, 2001). These cases studies show a significant improvement in the productivity of the team and in the quality of the software being developed when compared to plan-driven software development projects. However, Agile methods present certain challenges for highly regulated industries, such as the production of documentation and verification of the traceability from end product to customer requirement.

2.2 Agile values, principles and practices

Agile software development is a set of values, principles and practices that self-organizing teams use to quickly and regularly deliver software that provide value to their customers. It follows an incremental and evolutionary life cycle. Agile puts a bigger emphasis on collaboration between the development team, the customer and other stakeholders. Agile recognizes the need to adapt its practices to the context in which software is being developed.

Agile software development utilizes a completely different paradigm to that of traditional software development processes. This new way of developing software is rooted in a set of values, principles and practices that guide Agile teams when performing their work.

Values are abstract, universal concepts that all members of the team believe provide value to the team as a whole: it is not how any member of the team behaves what matters, but how individuals behave as part of a team. It is important that the team adapts its behaviour to its values. Values don't provide concrete advice on what to do in software development. Values held by the team shape its practices in ways that benefit the team needs.

Practices are the common procedures and the routines that the team follows in a regular basis when developing software. Practices are the visible artefacts of the way the team works. Practices are evidence of values. Just as values bring purpose to practices, practices bring accountability to values. Practices are clearly stated. They are defined to address a real life software development problem (for example, lack of communication, poor quality of the code, lack of code reviews, etc.). Practices are highly situated and should be considered in the particular context of the team.

Principles bridge the gap between values and practices. Principles are a set of domain-specific guidelines for finding practices in harmony with the values of Agile. While the statement of the practices is intended to be clear and objective, understanding how to apply the practices in a given context might not be obvious. These principles give a better idea of what the practice is intended to accomplish (Beck 2004).

2.2.1 Manifesto for Agile Software Development

The Agile values are captured by the Agile Manifesto (Beck et al. 2001)

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

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

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

The Agile Manifesto is a clear, powerful and provocative statement of the values that drive Agile software development. The values of this manifesto can seem to be contradictory to those of the regulated world of medical device software. However, when interpreted correctly, they enhance the value of quality management systems required by regulations and standards: The goals of Agile are to provide benefits in the quality of the product, improve the product's effectiveness and increase the productivity and predictability of the development team.

The last phrase of the Agile Manifesto "*while there is value in the items on the right, we value the items on the left more*" provides a vital qualification of the Agile values. Many of those that oppose to the use of Agile on safety critical software development inferred that the word "over" here means "instead of", leading to the incorrect conclusion that Agile practices required no documentation, no processes and so on. Or that regulations required things that are not "Agile". Thoughtful consideration of this phrase is essential to the proper application of Agile when developing medical device software in order to find the proper balance of what is valued.

Individuals and interactions over processes and tools

This value recognizes that skilled people working well together will produce good software and that processes and tools are useful but insufficient. Effective processes and tools will help a good team perform even better. But no amount of processes and tools will help a poor team perform well.

Software development is a creative process; hence developers and designers have to be allowed to think, to create and to solve problems. Good software can't be produced by following processes and check-lists alone.

A criticism to this value is that it might lead to a lack of discipline. That could be true if this value would be taken to the extreme. However, if applied with balance it can lead to an even better discipline. Formal processes and support tools bring discipline to a software development process when they codify behaviors that the team consider important. This kind of discipline is useful for bringing consistency to the way the team work together. However, this can lead to negative output if the team doesn't feel responsible of the processes, leading to processes that are misunderstood or ignored.

Working software over comprehensive documentation

This value recognizes that working software is the ultimate deliverable, the best indicator of progress and the best way to assess whether the needs of the customer are being satisfied. Documentation is necessary and useful but it is of little value if it doesn't come together with working software. Documentation, when clear and sufficient, helps the team in developing a good software product. It also helps external stakeholders evaluating the completeness and quality of the software that has been delivered.

Agile can't be used as an excuse to not producing the needed documentation. Agile values and specially the principle of "mutual benefit" has to be applied to ensure that valuable documentation that benefits both the team and external stakeholders is produced and that wasteful documentation is eliminated.

Customer collaboration over contract negotiation

This value recognizes that collaboration with a customer who is engaged in the development process is more likely to deliver software that meets the customer's needs. Contracts alone cannot guarantee that the team will produce software that fulfills the customer's requirements. However, clear contracts are useful to establish a common understanding and to set reasonable expectations. Inflexible contracts can hinder innovation, which is essential when developing a new software product.

In medical devices software development it can be challenging to determine who the customer is. Medical software can be used by multiple users and it is typically part of a wider system. Moreover, the software definition will evolve over time. It is important to establish a customer role in the development team and mechanism that support emergent product definition.

Responding to change over following a plan

This value recognizes that change is inevitable in new product development and should be embraced as a good and useful thing. Plans alone are not enough to ensure that a good product will be delivered.

In Agile software development, planning occurs at different levels during the product development life cycle, from a coarse-grained plan to a detailed one as the features are being implemented. This sequential planning allows the team to incorporate feedback and make adjustments as software is being developed; minimizing waste in form of very detailed up-front plans in case radical and unexpected changes occur that impact greatly the software being developed.

It is important to define planning mechanisms and to establish feedback mechanisms to demonstrate effective control of changes.

2.2.2 Agile principles

The principles behind the Agile manifesto are:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

- Business people and developers must work together daily throughout the project.

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

- Working software is the primary measure of progress.

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

- Continuous attention to technical excellence and good design enhances agility.

- Simplicity--the art of maximizing the amount of work not done--is essential.

- The best architectures, requirements, and designs emerge from self-organizing teams.

- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

In the next section we will discuss how these principles align with the special needs of medical device software development.

2.2.3 Agile principles in the context of medical device software

There are many publications that describe the principles of Agile software development in the context of traditional, non-regulated software development. Here we will introduce a list of principles that are relevant in the context of medical device software development, as described by the Association for the Advancement of Medical Instrumentation in their TIR45:2012 *Guidance on the use of AGILE practices in the development of medical device software* (Association for the Advancement of Medical Instrumentation, 2012):

- Apply incremental and evolutionary lifecycle concepts
- Define what **DONE** means
- Deliver customer value, with the highest priority features first, through collaboration between the team and the customer
- Accept that customer needs and requirements are likely to change throughout the project and accommodate to this change in an effective and efficient way
- Manage project risks through increased visibility and team accountability
- Empower the teams to become self-organized and to manage the daily tasks
- Seek for technical excellence in the software through high-quality designs and verification practices
- Reflect and adapt the process at regular intervals to constantly improve quality and efficiency of the work
- Satisfy business stakeholders, both internal and external

2.2.4 Agile practices in the context of medical device software

The fundamental nature of Agile is to be adaptable to the context in which it is being applied. The following summarizes the Agile practices relevant to the development of

medical device software gathered from various Agile methodologies (Scrum, XP, etc.) They have been grouped according to the software development phase that they apply:

- **Product definition:** product vision/vision statement, product backlog, epic/stories/use cases/personas, backlog management, planning poker
- **Product implementation:** emergent/evolutionary design, refactoring, architecture spikes, coding guidelines/standards, test driven development (TDD), continuous integration, daily builds/automated builds, frequent delivery/frequent releases, unit testing, story testing/executable requirements, and test automation
- **Project execution:** time boxing/fixed increments length, release planning, increment planning, daily planning, definition of done, and velocity/burn down/burnup charts
- **Team organization:** self-organizing teams, team roles, team size, scrum of scrums and sustainable pace
- **Team collaboration:** stop-the-line/information radiators, co-location, pairing, reflections/retrospectives, and collective ownership

2.3 Summary of the conceptual framework

Agile software development is based on a set of values, principles and practices that self-organizing teams use to quickly and regularly deliver software that provide value to their customers, in an evolutionary and incremental manner.

Values are abstract, universal concepts that all members of the team believe provide value to the team. It is important that the team adapts its behaviour to its values. Values held by the team shape its practices in ways that benefit the team needs.

Practices are the common procedures and the routines that the team follows in a regular basis when developing software. Practices are evidence of values. Practices are clearly stated. They are defined to address a real life software development problem. Practices are situated and should be considered in the particular context of the team.

Principles bridge the gap between values and practices. Principles are a set of domain-specific guidelines for finding practices in harmony with the values of Agile. These principles offer a better idea of what the practice is intended to accomplish.

The table below summarizes the values, principles and practices that define the ways of working of an Agile software development team in the context of medical device software development:

Table 2.1 - Agile values, principles and practices

Values	**Principles**	**Practices**
Individuals and interactions over processes and tools	**Incremental and evolutionary lifecycle**	**Product definition:** product vision/vision statement, product backlog, epic/stories/use cases/personas, backlog management, planning poker.
Working software over comprehensive documentation	Definition of **DONE**	**Product implementation:** emergent/evolutionary design, refactoring, architecture spikes, coding guidelines/standards, test driven development (TDD), continuous integration, daily builds/automated builds, frequent delivery/frequent releases, unit testing, story testing/executable requirements, and test automation
Customer collaboration over contract negotiation	**Deliver customer value**, with the highest priority features first, through customer collaboration	**Project execution:** time boxing/fixed increments length, release planning, increment planning, daily planning, definition of done, and velocity/burn down/burnup charts
Responding to change over following a plan	**Embrace change** of customer needs throughout the project	**Team organization:** self-organizing teams, team roles, team size, scrum of scrums and sustainable pace
	Manage project risks through **increased visibility and team accountability**	**Team collaboration:** stop-the-line/information radiators, co-location, pairing, reflections/retrospectives, and collective ownership
	Empower the teams to become self-organized and to manage the daily	

	tasks	
	Seek for **technical excellence** in the software through high-quality designs and verification practices	
	Reflect and adapt the process at regular intervals to constantly improve quality and efficiency of the work	
	Satisfy business stakeholders, both internal and external	

3 Analysis of the Medical Device Directives and Standards

3.1 Definition of Medical Device

A medical device is defined as an instrument, apparatus, implant, in vitro reagent, or similar or related article that is used to diagnose, prevent, or treat disease or other conditions, and does not achieve its purposes through chemical action within or on the body (which would make it a drug) (U.S. Food and Drug Administration, 2014).

Medical devices can vary greatly in complexity ranging from a simple bandage or dressing to a pacemaker, cochlear implant or a life sustaining medical ventilator.

In Europe, Directive 2007/47/EC defines a medical device as: “Any instrument, apparatus, appliance, software, material or other article, whether used alone or in combination, together with any accessories, including the software intended by its manufacturer to be used specifically for diagnostic and/or therapeutic purposes and necessary for its proper application, intended by the manufacturer to be used for human beings for the purpose of:

- Diagnosis, prevention, monitoring, treatment, or alleviation of disease
- Diagnosis, monitoring, treatment, alleviation of, or compensation for an injury or handicap
- Investigation, replacement, or modification of the anatomy or of a physiological process
- Control of conception

This includes devices that do not achieve their principal intended action in or on the human body by pharmacological, immunological, or metabolic means—but may be assisted in their function by such means” (Council Directive 2007/47/EC).

This definition includes explicitly the term “software”, *whether used alone or in combination*. This is an important remark, since now the definition includes not only the embedded software which is part of a medical device, but also standalone software

that runs on off-the-shelf computer hardware such as personal computers, smartphones, tablets and even *smart watches*.

The manufacturing and commercialization of medical devices is a regulated activity. The manufacturer of medical devices must comply with the requirements set by the regulatory agencies of the geographical areas where it intend to market its devices (the FDA in U.S., European Commission in Europe, etc.).

Medical devices must not endanger the safety or health of the patient or the caregivers. The manufacturer of a medical device must ensure:

- Safety
- Suitability for the intended use
- Performance and reliability

In the following section we will introduce the legal framework that regulates the manufacturing and commercialization of medical device in Europe, its safety classification and the certification process.

3.2 EU MD Directives

In Europe the following directives define the rules that regulate the safety and performance of medical devices:

- Council Directive 90/385/EEC regarding active implantable medical devices
- Council Directive 93/42/EEC concerning medical devices
- Council Directive 98/79/EC regarding in vitro diagnostic medical devices

These three main directives have been supplemented over time by several modifying and implementing directives, including the last technical revision brought about by Directive 2007/47 EC.

They aim at ensuring a high level of protection of human health and safety and the good functioning of the Single Market by eliminating technical barriers to trade and dispel the consequent uncertainty for economic operators.

The government of each Member State must appoint a competent authority responsible for medical devices. The competent authority (CA) is a body with authority to act on behalf of the member state to ensure that member state government transposes requirements of Medical Device Directives into national law and applies them. The CA reports to the minister of health in the member state. The CA in one Member State has no jurisdiction in any other member state, but exchanges information and tries to reach common positions.

The European Commission provides a set of guidelines to help medical device manufactures in the implementation of these directives (European Commission 2015) The guidelines aim at promoting a common approach by manufacturers and Notified Bodies involved in the conformity assessment procedures according to the relevant annexes of the directives, and by the Competent Authorities charged with safeguarding Public Health.

3.2.1 Classification of medical devices

The classification of medical devices in the European Union is outlined in Annex IX of the Council Directive 93/42/EEC. There are basically four safety classes, ranging from low risk to high risk.

- Class I (including Is & Im)
- Class IIa
- Class IIb
- Class III

The purpose of establishing a classification scale is to ensure an adequate level of supervision and validation based on the potential risk that patients or caregivers could be exposed when using the device, since it wouldn’t be feasible or economically viable to impose the strictest controls to each and every medical device.

The authorization of medical devices is guaranteed by a Declaration of Conformity. This declaration is issued by the manufacturer itself, but for products in Class Is, Im, IIa, IIb or III, it must be verified by a Certificate of Conformity issued by a Notified Body. A Notified Body is a public or private organisation that has been accredited to validate the compliance of the device to the European directive. Medical devices that pertain to class I (on condition they do not require sterilization or do not measure a function) can be sold purely by self-certification.

The European classification depends on rules based on the following:

- The medical device's duration of body contact
- Invasive character
- Use of an energy source
- Effect on the central circulation or nervous system
- Diagnostic impact
- Incorporation of a medicinal product

The European Commission has published a guidance document to help medical devices manufacturers and notified bodies in the application of the classification rules defined by the Council directive 93/42/EEC. This document provides advice when using the classification rules and illustrates them with practical examples.

When interpreting the classification rules manufactures should bear in mind the following:

- It is the intended purpose that determines the class of the device and not its particular technical characteristics
- It is the Intended use and not the accidental use of the device that determines its class
- It is the Intended purpose assigned by the manufacturer to the device that determines it class and not the class assigned to other similar products

- A device that is part of a system may be classed as a device on its own right rather than classifying the system as a whole
- Accessories of medical devices are classified in their own right separately from the device that they are used with
- General purpose devices that are used in combination with medical devices are not considered medical devices

Certified medical devices should have the CE mark on the packaging, insert leaflets and other marketing material or documentation. The packaging should also show harmonised pictograms and EN standardised logos to indicate essential features such as instructions for use, expiry date, manufacturer, sterile, don't reuse, etc.

3.3 Medical Device Software

As previously stated, the legislation that regulates the manufacturing and commercializing of medical devices in Europe apply as well to the software components of medical devices. In particular, Directive 2007/47/EC which amended the definition of the term “medical device” used in Directives 90/385/EEC and /93/42/EEC. Recital 6 of Directive 2007/47/EC, states that “it is necessary to clarify that software in its own right, when specifically intended by the manufacturer to be used for one or more of the medical purposes set out in the definition of a medical device, is a medical device. Standalone software for general purposes when used in a healthcare setting is not a medical device.”

In the following section we will highlight the motivation behind medical device software regulations and the objectives that competent authorities seek to achieve with said regulations.

3.3.1 Regulatory goals, values, principles and practices

Regulatory agencies are responsible for protecting the health of patients, caregivers and others by assuring the safety, security and efficacy of medical devices.

Regulatory goals are to provide benefits on the following areas:

- **Quality**: correctness of the product, reliability, free of defects
- **Safety**: identification and mitigation of safety risks to patients and users
- **Effectiveness**: delivering value to the patient and users, satisfying customers' needs

The regulatory values and principles can be derived from the regulations, standards and guidelines. The following are some principles that are relevant when developing medical device software:

- **Established and controlled design and development processes** are essential when producing high quality software. The level of control and the amount of effort devoted to it should be decided based on the intended use and the level of safety associated with the software being developed
- **Software verification and validation** must be conducted throughout the development of the software regardless the chosen life cycle model
- **Quality systems** must be put in put in place to ensure that the software meets the customer requirements with the best level of correctness and free of defects. Quality rely on management control, professional work and professional training
- **Documentation** is necessary to demonstrate compliance to regulations, to facilitate the maintenance, investigation of software problems and to evaluate software for those devices requiring regulatory approval
- **Planning** is essential to ensure a high quality and an efficient implementation of the software product. Risk management activities should be conducted as part of the software development activities

The software development practices that are encouraged by the regulatory perspective can be grouped in categories based on the problem that they attempt to solve:

- **Patient safety and risk management:** identification of risk scenarios, risk mitigation, verification to assess the remaining risks in a software product

- **Quality planning:** management controls, planning of the software development environment, team development and training
- **Documentation:** document controls, configuration management and change controls
- **Software definition:** user requirements, design requirements and definition of system and software interfaces
- **Software design:** architecture design, detailed design, implementation and design change management
- **Verification and validation:** design reviews, peer reviews, unit, integration and system testing and user validation

Regulations and standards do not usually specify how these activities must be performed. This allows manufacturers to design their own practices in a way that suits their needs best and aligns with their own ways of working and culture while maintaining conformity with regulatory principles.

Once we have outlined the motivations and objectives behind the directives that regulate medical devices, we will next introduce the standards and technical reports that help manufacturers of medical device to comply with those regulations.

3.4 Standards, technical reports and guidance documents

As discussed in section 4.2, manufactures of medical devices must comply with the relevant EU directives for their devices. A set of harmonized standards exists to assists manufacturers with achieving compliance. Although it is not compulsory to follow these standards, vendors who do so will find easier to demonstrate compliance.

Guidance and technical reports help medical device manufactures and regulatory authorities in assessing the classification of medical devices. They provide guidance in the application of standard practices during the medical device development process.

3.4.1 Standards applicable to medical device software

There are a number of harmonized standards, each of them setting requirements for compliance in a given area (development process, risk management, quality management, etc.). These are the most relevant harmonized standards for medical device software development:

- IEC 62304:2006, Medical Device Software - Software life cycle processes
- ISO 14971:2007, Medical Devices - Application of risk management to medical devices
- ISO 13485:2003, Quality management systems - Requirements for regulatory purposes
- IEC 62366-1:2015 Medical Devices - Part 1: Application of usability engineering to medical devices

The most relevant standard for the development of medical device software is IEC 62304:2006. This standard defines the requirements for the software life cycle processes during the software development and maintenance phases.

This standard defines a framework of life cycle processes for the safe development and maintenance of medical device software. The standard identifies processes, activities and tasks that have to be carried out during the development and maintenance phases in order to demonstrate compliance. Each life cycle process is divided in a set of activities and these are further decomposed into individual tasks.

The standard assumes that medical device software is being developed and maintained within a quality management system and a risk management system. For risk management, the standard IEC 62304 requires the manufacture of medical device software to comply with ISO 14791.

The standard requires that a software development plan must be documented and followed. However, the standard does not impose the use of any particular life cycle model. Manufactures of medical device software are free to choose and adapt any life cycle model that better suits their needs and current practices as long as the

processes, activities and tasks defined by the standard are being performed and documented within their life cycle.

The standard identifies two additional processes considered essential for developing safe medical device software: software configuration management process and software problem resolution process.

The figure below shows the software development process activities as described by the standard:

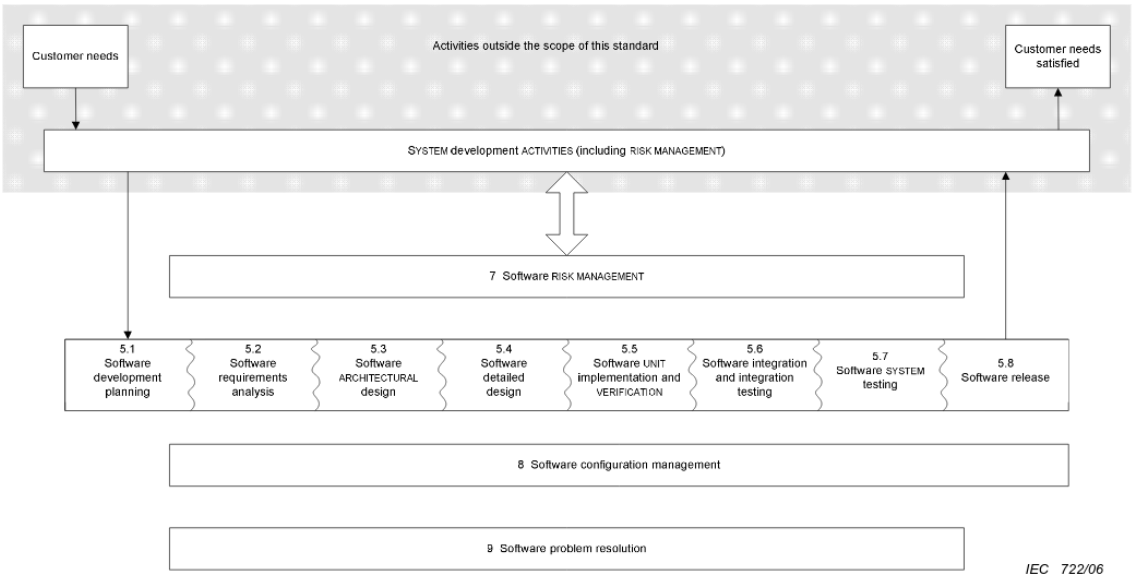

Figure 3.1 - Overview of SW development processes and activities as defined in IEC 62304

The standard considers the maintenance of the medical device software as important as its development.

The figure below describes the software maintenance processes and activities:

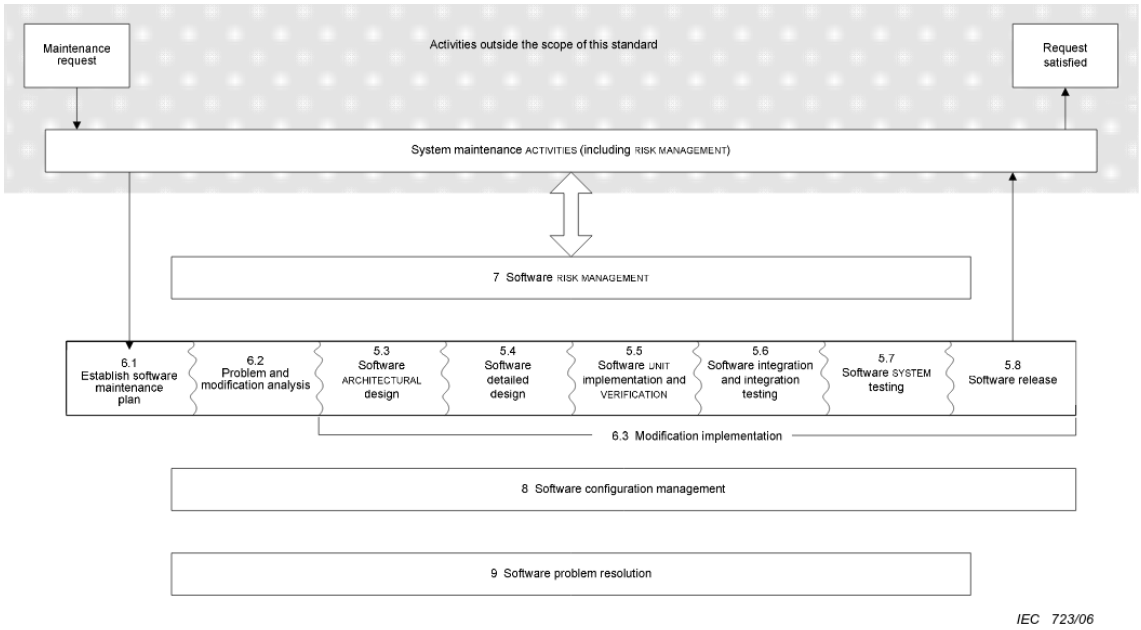

Figure 3.2 - Overview of SW maintenance processes and activities as defined in IEC 62304

A controlled and strict maintenance process needs to be in place in order to avoid incidents due to incorrect servicing or maintenance of the device, including software updates and upgrades.

3.4.2 Guidance documents

Various organizations including regulators, notified bodies and industry associations produce guidance documents to help manufacturers of medical devices with the interpretation of the regulations and the application of the standards. The technical information reports and guidance documents relevant for the development of medical device software include the following:

- AAMI TIR45:2012, Guidance on the use of AGILE practices in the development of medical device software
- MEDDEV 2.4/1 Rev.9 June 2010, Medical Devices - Classification of medical devices
- MEDDEV 2.1/6 January 2012, Medical Devices - Qualification and Classification of standalone software
- NB-MED/2.2/Rec4 Software and Medical Devices

- IEC/TR 80002-1:2009, Technical Report-Medical device software-part 1: Guidance on the application of ISO 14971 to medical device software

- General Principles of Software Validation; Final Guidance for Industry and FDA Staff

For the purpose of this thesis, the most relevant document is the technical information report AAMI TIR45:2012 produced by the Association for the Advancement of Medical Instrumentation.

This technical information report provides recommendations for complying with international standards and with medical devices regulations when using Agile practices to develop medical device software. It also provides guidance to adapt Agile practices to fit the unique needs of medical device software development.

This technical information report describes first the Agile perspective, explaining its goals, values, principles and practices. It then goes on to describe the goals, values, principles and practices that define the regulated world of medical device software development. Most of the Agile values and principles align very well with those of the regulatory perspective while others might provide some challenges. This document also provides medical device software manufacturers with recommendations on how to align the values and principles of both perspectives in a supportive manner.

3.5 Summary

Medical software development process is regulated by the competent authorities of the areas in which the software product will be marketed and taken into use. Standards and guidelines exist to assist development teams in the certification process.

The standard IEC 62304:2006 provides a conceptual framework that covers the complete software development and maintenance phases. It identifies the processes, activities and tasks that are carried out when developing medical software; it describes the sequence and dependencies between activities and it identifies the milestones at which deliveries are verified. The standard requires that a software development plan exists and it is being followed. It does not mandate the usage of any concrete life cycle model, implying that development teams should adapt the one that best fits their needs

and the existing practices of their organizations (e.g., regarding the usage of quality and risk management systems).

The table below summarizes the requirements set by the standard IEC 62304:2006 grouped by categories:

Table 3.1 - Software development process requirements as specified by IEC 62304:2006

Quality management	Demonstrated by compliance with ISO 13485
Risk management	Demonstrated by compliance with ISO 14971
SW development process	• Development and maintenance planning • Requirement analysis • Architectural design • Detailed design • Unit implementation and verification • Integration and integration testing • System testing • Releasing • Risk management process • Configuration management • Problem resolution • Change management

The technical information report TIR45:2012 provides recommendations for complying with international standards (i.e., IEC 62304:2006) and with medical devices regulations when using Agile practices to develop medical device software. It also provides guidance to adapt Agile practices to fit the unique needs of medical device software development teams.

4 Guidelines for the Development of Medical Device SW

4.1 Introduction

As discussed in the previous chapter, the standard IEC 62304:2006 mandates that certain activities have to be performed and documented during the software development and maintenance phases. It does not mandate, however, in which sequence these activities should be performed.

The standard IEC 62304, Section 5.1.1, requires that the medical device software manufacture choose and define their *software development life cycle*. Although the structure of the standard is heavily based on a Waterfall model, it does not impose the use of any particular life cycle model.

In this chapter we will propose a number of guidelines for the development of medical device software following an Agile approach. We will provide guidelines for the software development process, the software maintenance process, the software risk management process, the software configuration management process and the problem resolution process. These guidelines are based on the Agile practices we introduced in Chapter 3 and the recommendations provided by the Association for the Advancement of Medical Instrumentation in their TIR45:2012.

We will propose an evolutionary and iterative development life cycle. We will discuss how the development activities are defined in ways that the proposed processes comply with the requirements set by the medical devices regulations and relevant harmonized standards.

The processes and activities proposed here are expected to be used by the case company software engineering team as guidance when developing and maintaining their medical device software. The Agile principle *reflect and adapt* must be observed in order to adapt the process at regular intervals to constantly improve the quality and efficiency of the work.

4.1.1 Vision based baby monitor

The device being developed by the case company *Circular Devices* is a vision based baby monitor that detects the baby's breathing movements and triggers an alarm if the movements stop. It is a stand-alone, dedicated device. Its hardware consists on an embedded microprocessor, a dual camera, infrared light emitting diodes used for illumination under dark operation conditions (night vision) and various input and output devices to control its operation and to produce acoustical and visual alarm signals.

Software is a fundamental part of the medical device being developed. Software is responsible for detecting and monitoring the baby's breathing movements using machine vision algorithms, measuring his respiratory frequency and producing an alarm if there's no breath for a set number of seconds, a condition known as *apnoea*.

The diagram below shows a high level overview of the software architecture and the main components of the system:

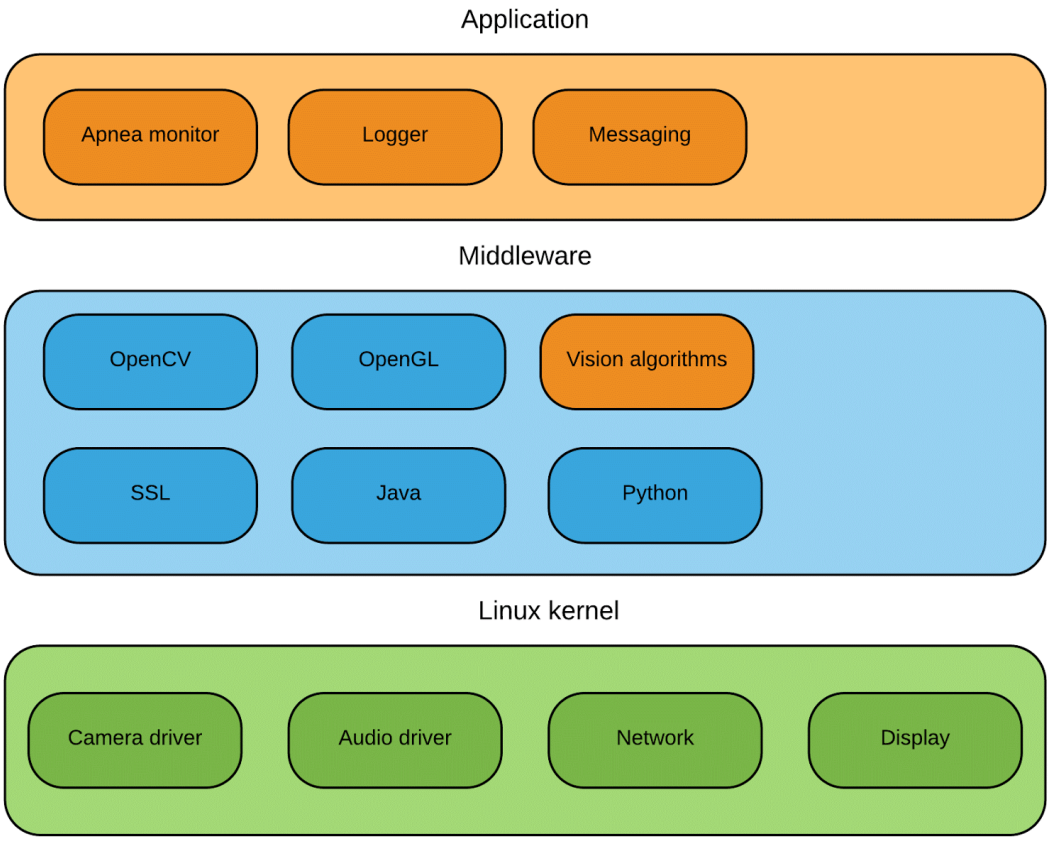

Figure 4.1: SW architecture diagram

At the bottom of the SW stack we find the Core OS layer which main components are the Linux kernel, the C library, camera drivers, audio drivers, graphics and display drivers and network stack. Most of these components are being developed by the open source community or by the respective device manufactures (e.g., camera drivers). The case company does not actively develop any of these components but it might submit to the upstream projects bug fixes or modifications made to them. For these components, the case company apply the SOUP maintenance plans and risk management process described later in this chapter.

The next layer in the software stack is the Middleware layer. This layer is comprised of software libraries that provide certain specialized functionality, such as the OpenCV computer vision library, OpenGL graphic acceleration and various virtual machines and interpreters such as Java and Python. Most of these components are either open source or developed by their respective manufactures (e.g., Java is being developed by Oracle). The case company adds extra functionality at this layer by implementing its own machine vision algorithms. For the components being developed outside of the case company the SOUP processes apply. For those components being actively developed by the case company, the full development, maintenance and risk management processes described later in this chapter need to be followed.

The top layer in the software stack is the Application layer. This layer is made of the high level applications that implement the full functionality of the device (e.g., the apnoea monitor). There are some other applications such as logging to keep track of any occurrences of apnoea as well as a messaging application that can be used to send alarms to other devices such as mobile phones or tablets. Any helper applications used for tracing and debugging during the development process belong to this layer as well. These applications are fully developed by the case company. The full development, maintenance and risk management processes apply to the components in this layer.

4.1.2 Software safety classification of the baby monitor

According to the Recommendation NB-MED/2.2/Rec4 *Software and Medical Devices* (Co-ordination of Notified Bodies Medical Devices 2001), depending on the intended use by the manufacturer, medical device software can be

a) *a medical device or an accessory to a medical device, which must be CE-Marked, or*

b) *a component and integral part of a medical device, which cannot be CE marked in its own right, but which is covered by the conformity assessment of the medical device of which it forms a part or*

c) *None of the above and therefore not covered by the Medical Devices Directives.*

In our case, the software being developed falls under the second category since it is a component and integral part of the baby monitor, which is a medical device.

The case company intends to place the vision based baby monitor in the market as an *infant or child apnoea monitor*, which according to the Annex IX of the Council Directive 93/42/EEC and MEDDEV 2. 4/1 Rev. 9 would be classified as a Class II medical device.

The standard IEC 62304:2006 requires that the manufacturer assigns to each software system a software safety class (A, B or C) according to the possible effects on the patient, operator or other people resulting from a hazard to which the software system can contribute.

- Class A: No injury or damage to health is possible
- Class B: Non-serious injury is possible
- Class C: Death or serious injury is possible

The standard will require stricter risk management activities and higher levels of documentation for software systems with a higher safety class.

4.2 Guidelines for the software development process

A software development process defines the activities that are to be performed when developing a software product. The standard IEC 62304 requires manufacturers of medical device software to define and follow a development process when developing

their software products. In this section we will outline the proposed software development process based on Agile practices.

4.2.1 Software development plan

A software development plan must be created and updated regularly. This plan must include a reference to the software development life cycle model being used. The software development plan shall describe:

- The processes used when developing the medical device software
- The deliverables of each activity and task
- Traceability between system requirements, software requirements, system tests and risk control measures implemented in software
- Software configuration and change management
- Software problem resolution process

This plan must be kept up to date as the development proceeds. It must include references to system design and system development plans. The software development plan must include integration and testing plans. It shall cover as well software verification, risk management and documentation planning. Certain supporting items, such as development tools, chosen software settings, etc. which might impact the way the medical device software works shall be included in the development plan as well.

In a sequential life cycle, such as waterfall or V-model, this planning will be naturally performed at the beginning of the process, before the software development begins. In an evolutionary life cycle model such as the one being proposed, planning activities are being performed at different layers of abstraction throughout the whole project lifetime, as we will discuss in the next sections.

4.2.2 Software development life cycle model

The software development life cycle being proposed is based on an emergent and evolutionary model. It makes use of proven Agile techniques and practices and adapts them to the particular needs of a medical device software development context.

Figure 5.1 provides a high level view of the proposed life cycle[OBJ] (Schwaber and Beedle, 2002):

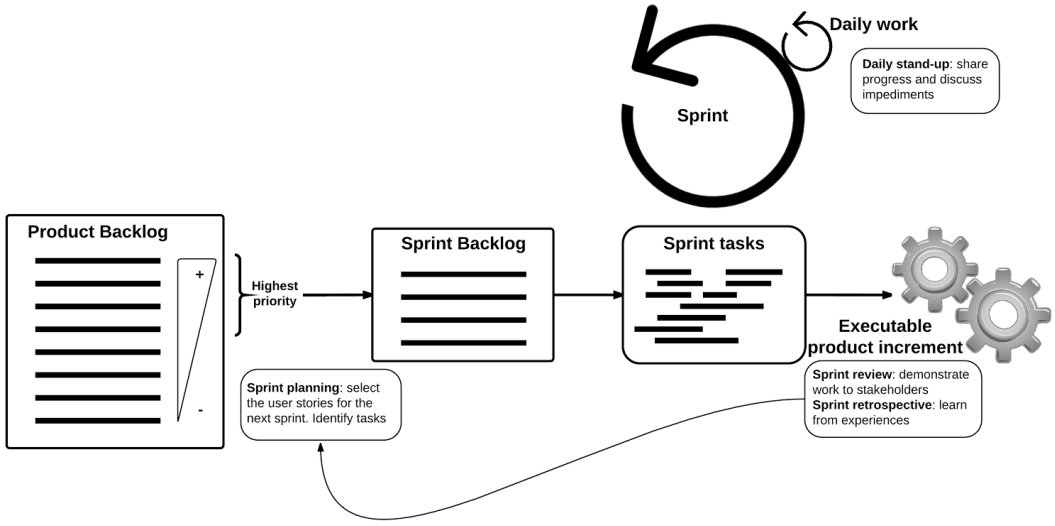

Figure 4.1 - SW development life cycle model based on Scrum

A “product backlog” consisting of “user stories” is created and its items are then arranged based on their priority. The “product owner” owns the backlog and he is responsible of updating and prioritizing it. At the beginning of each development iteration or “sprint”, the development team will go through the product backlog and will pick up the highest priority user stories to be worked on next. These stories will form the “sprint backlog”. The sprint backlog will be then further decomposed on tasks that developers can process during the sprint. These tasks include software design, implementation, integration, testing, verification, documentation, etc. Each day there is a “daily stand-up” meeting where the team members discuss their progress, identify any impediments and choose their next work tasks. At the end of the sprint, a “sprint review” is carried out where the team demonstrates the stories that have been completed during the sprint to the product owner and any other interested stakeholder (i.e., customer or marketing representative). The sprint review demonstrations are a

mechanism to gather continuous feedback from users. They serve as a tool for continuous requirements *verification and validation* by reducing the risk of incorrect requirements definition during the early phases of product development. The product owner accepts or reject these stories based on their completeness against the team's "definition of done". Accepted stories are marked as completed while rejected ones are taken back to the product backlog for re-prioritization. The team performs a "sprint retrospective" meeting where they learn about how the team worked during the sprint and those areas that require improvement are identified.

Process flow

The figure below describes the timeline of the development activities and the abstraction layers of the proposed software development process:

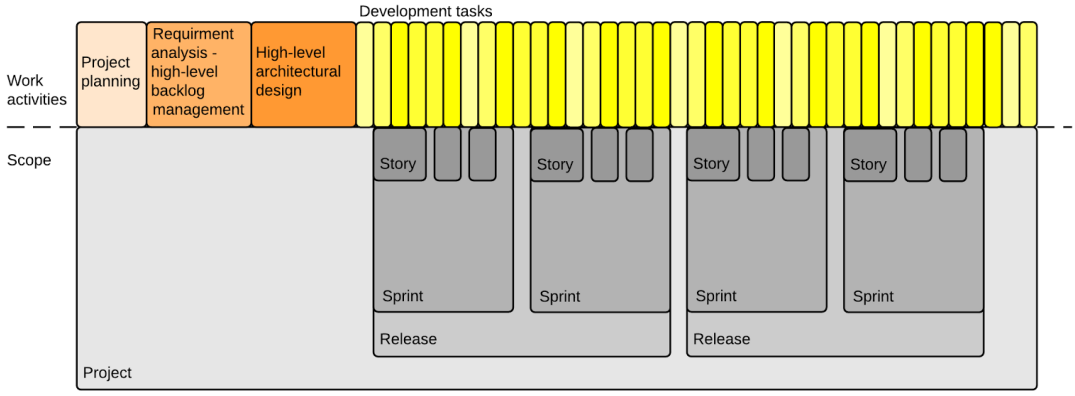

Figure 4.2 - development process timeline

The product development starts with the conception phase in which key product features are being identified. These features are then written in the form of user stories. User stories capture software features from an end-user perspective. The user story describes the type of user, what they want and why. A user story helps to create a simplified description of a requirement (Rouse, 2015). Following a preliminary analysis, these user stories are then prioritized in the product backlog. After that the first iteration or sprint can commence. In this first iteration the work consists mainly on a high level design of architecture and system core. A number of iterations are then performed until the system is deemed completed. These iterations consist of software architecture, detailed design, implementation, testing, integration, verification and documentation

activities and tasks. The output of these iterations are then grouped together to form a “release”, where further verification, validation, testing and integration is being performed.

4.2.3 Software development activities

The standard IEC 62304 defines a software development process model with a well-established set of activities necessary to develop medical device software. The figure below illustrates how these activities are being performed when following an Agile life cycle such as the one being proposed:

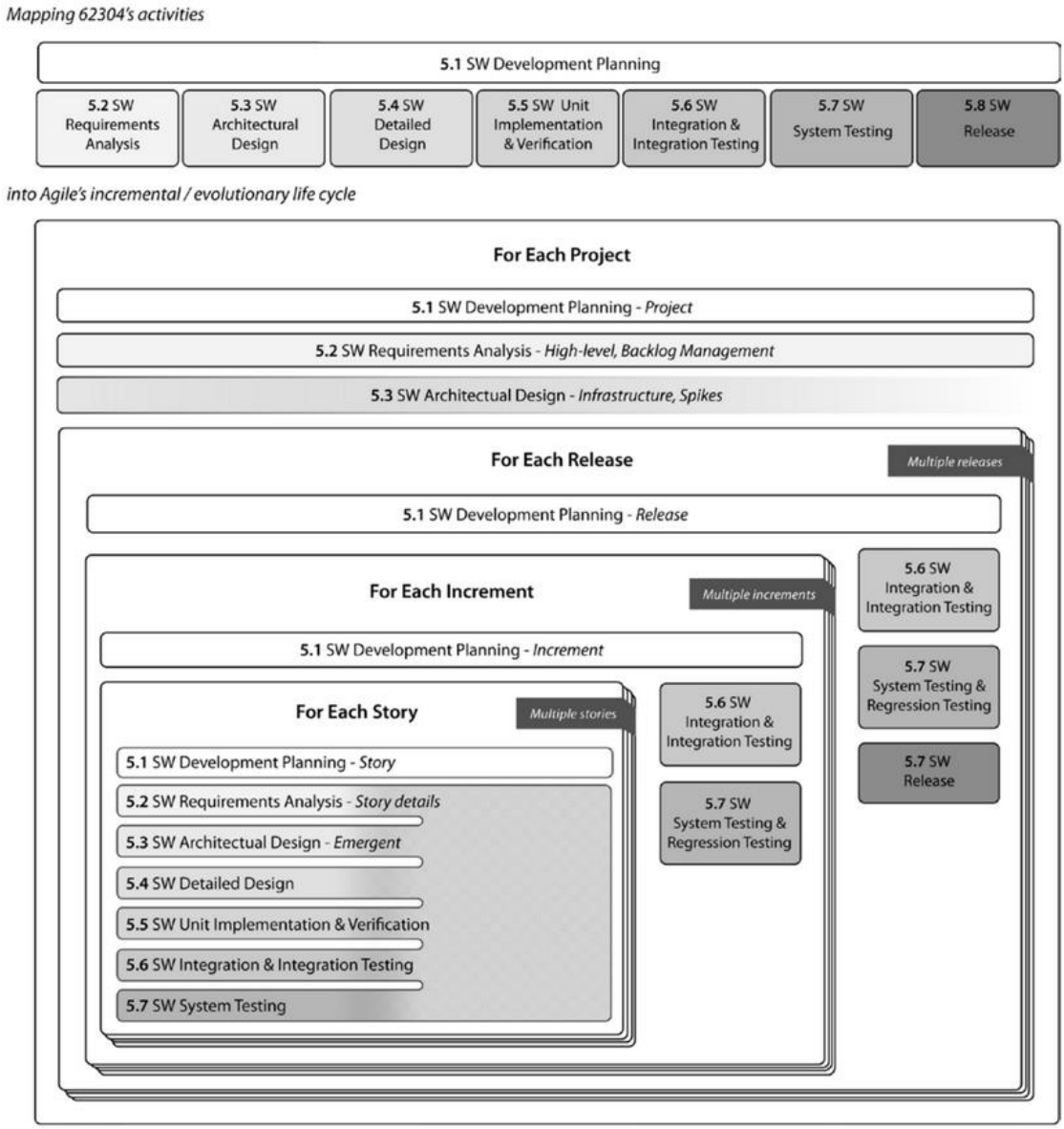

Figure 4.3 - Mapping IEC 62304's activities into Agile life cycle as defined by AAMI TIR45

At the top of the figure are the activities defined by IEC 62304. The bottom part of the figure shows how those activities are executed in an incremental and evolutionary life cycle model. When following such development life cycle, these activities are performed several times at different levels of abstraction.

In the proposed life cycle, we find four layers of abstraction: the **project layer**, which consists of the whole set of activities needed to deliver a finished software product; The **release layer** consists of the activities needed to create a usable product, wherever meant to be used by the end user or for internal purposes only; The **increment layer** consists of the activities needed to create a set of useful functionality, although not necessarily a complete product; the **story layer** consists of the activities needed to create a small piece of functionality. The time span of these abstraction layers goes from the whole lifetime of the software product (project layer), three to six months for a release, two to four weeks for an increment and one to three days for one story.

Next we will describe in more details how the software development activities defined by IEC 62304 are being performed at the different layers of abstraction in our development life cycle:

Software development planning

In the proposed development life cycle, software development planning is done at multiple layers. At the **project layer**, planning consists on the high-level activities of project management such as project scoping, forming the team, allocating resources and organizing the project into releases. At the **release layer**, planning addresses the mid-level activities of project management, such as scoping the release, scheduling the development activities within the release, defining integration points of the software and the rest of the system and organizing the release into increments. At the **increment layer**, planning addresses the low level activities of project management, such as scheduling and planning the low level activities, defining low level integrations points with other subsystems and organizing the increment into stories. At the **story layer**, planning addresses detailed team and individual planning, such as allocating tasks to individual team members, planning the execution of daily activities and tracking progress.

Feedback occurs within and between all layers of planning: during the daily stand-up meeting, developers share with the rest of the team what has been accomplished since the last daily meeting and bring up any issues that might be blocking their progress; during the sprint retrospective, the team reflect on how they are doing and find ways to improve.

Software Requirements analysis

Software requirements analysis occurs at two layers. At the **project layer**, software design inputs and specifications are processed and prioritized forming the initial product backlog which consists of high level definitions of the software requirements. At the **story layer**, detailed specifications of the software are created.

Software Architectural design

Software architectural design is being done at two layers of abstraction. At the **project layer**, a high level architectural work happens (either by allocating a complete increment to establish the architectural infrastructure as illustrated in figure 5.2 or by defining smaller development tasks dedicated to architectural work not related to any specific story). At the **story layer** detailed architectural design emerges by following Agile principles such as *simple design* and *refactoring*.

Software Detailed design, unit implementation and verification

Software detail design and software unit implementation and verification occur at the **story layer**. Following the Agile concepts of *simple design* and *refactoring*, detailed designs emerges as code is being produced. By following a *test-driven development* methodology, unit tests are written and executed at the same time than the code is being developed.

Software Integration and integration testing

Software integration and integration testing occurs at three layers of abstraction. At the **story layer**, software is being integrated into the rest as the system as code is being developed by using a *continuous integration* system. By following *test-driven development* methods, the integration is tested as the pieces are being put together. At the **increment layer**, test-driven development is used to integrate and test individual

stories as they are completed to create a complete set of related functionality. At the **release layer**, the integration of the complete sets of functionality provided by the different increments forming the release is then tested.

Software System testing

Software system testing occurs at three layers of abstraction. At the **story layer**, a test or other verification methods is created as the software requirement is defined. This test might be executed at the story layer or later at the increment or release layers when other required elements of the system are being integrated. At the **increment layer**, tests defined by the stories are executed or new ones are created to verify requirements related to subsystem integrations. At the **release layer** existing tests are executed once again or new ones are created to verify system level integration requirements.

Software Release

Software release activities are performed at the release layer.

4.3 Guidelines for the software maintenance process

The Standard IEC 62304 requires a software maintenance process to be established in addition to the software development process. The software maintenance process needs to be followed when performing software maintenance activities such as fixing defects, implementing change requests or when adding new software functionality (software upgrades) after the software product has been released to end users.

This section describes the software maintenance process in the proposed Agile software development life cycle.

4.3.1 Software maintenance plan

A software maintenance plan has to be established for conducting the software maintenance activities described in section 4.4.1, figure 4.2. This maintenance plan should include the following:

- Procedures for receiving, documenting, evaluation, resolving and tracking feedback arising after the release of the medical device software
- Criteria to determining whether the feedback is considered to be a problem
- Use of the software risk management process
- Use of the software problem resolution process
- Use of the software configuration management process
- Process to evaluate and implement upgrades, bug fixes, patches and obsolescence of SOUP

4.3.2 Problem and modification analysis

As part of the software maintenance process, any software defect or change request has to be analysed following a problem resolution process as described in section 5.6.

4.3.3 Modification implementation

After a software defect has been analysed and it has been found that it needs fixing or a change request has been approved for implementation, we shall follow the same Agile software development process as described in section 5.2. Software fixes or change requests items are added to the product backlog, with a clear marking (such as [BUG:XXX] or [CR:YYY] prefix in their title) to indicate that they originate from a defect report of change request. They are then prioritized together with the rest of items in the backlog. From here on the development activities don't differ substantially whether they correspond to new functionality or defect fixing and change request (other than in the later cases the fix has to be sufficiently documented in the defect tracking or change management tools).

4.4 Guidelines for the risk management process

The standard ISO 14971 defines the application of risk management to medical devices. The case company develops medical device software that belongs to risk class A or B.

When developing medical device software, the manufacturer must identify the software items and potential causes that could contribute to a hazardous situation, including:

- Incorrect or incomplete specification of functionality
- Software defects in the identified software item functionality
- Failure or unexpected result from SOUP
- Hardware failures
- Reasonably foreseeable misuse

If a failure of SOUP used in the medical device software is a potential cause to a software item contributing to a hazardous situation, any publicly available anomaly list should be reviewed to identify any possible sequence of events that could lead to a hazardous situation.

A *risk management file* must be maintained listing all potential causes of the software item contributing to hazardous situations. The sequence of events that lead to hazardous situations must be documented on this file as well.

4.5 Guidelines for the software configuration management process

The standard IEC 62304 requires a software configuration management process to be established when developing medical device software. Configuration items include developer tools, compilers, IDE, operating systems, testing tools, integration tools, Software of Unknown provenance (SOUP) and so forth.

4.5.1 Identification of configuration items

A mechanism to uniquely identify configuration items and their versions must be established. SOUP items are identified by a name, their manufacturer and a unique identifier such as version number or release date.

A *system configuration* list must be created that lists all the configuration items and their respective versions for a given software product release.

4.5.2 Change control

Configuration items must be changed only in response to an approved change request. The changes must be implemented as specified in the CR following the development or maintenance processes including appropriate verifications tasks. It is important to be able to trace any changes to configuration items.

4.5.3 Configuration status accounting

The history of changes to configuration items must be kept. It should be possible to retrieve the exact version of each configuration items for a given software product release. The consistent use of software version management and continuous integration tools can greatly simplify this task.

4.6 Guidelines for the software problem resolution process

A software defect tracking tool has to be configured and taken into used to track feedback originating both internally and from end users. The use of such tool serves as well as a mean to documenting the problem resolution analysis and fixing processes. By using unique identifiers (such as the bug or CR id's), It also helps in tracking the sections of the source code being modified in order to address the defect or change request and to trace it during the rest of the development activities until it is finally made available to internal or end users as part of a software release.

Each software problem report must include the following:

- **Type** (example: corrective, preventive, or adaptive to new environment)
- **Scope** (example: size of change, number of device models affected, supported accessories accepted, resources involved, time to change)
- **Critically** (example: effect on performance, safety or security)

The software maintenance process must be followed when implementing defects fixing activities as described in section 5.3.3

The figure below illustrates the software problem resolution process flow:

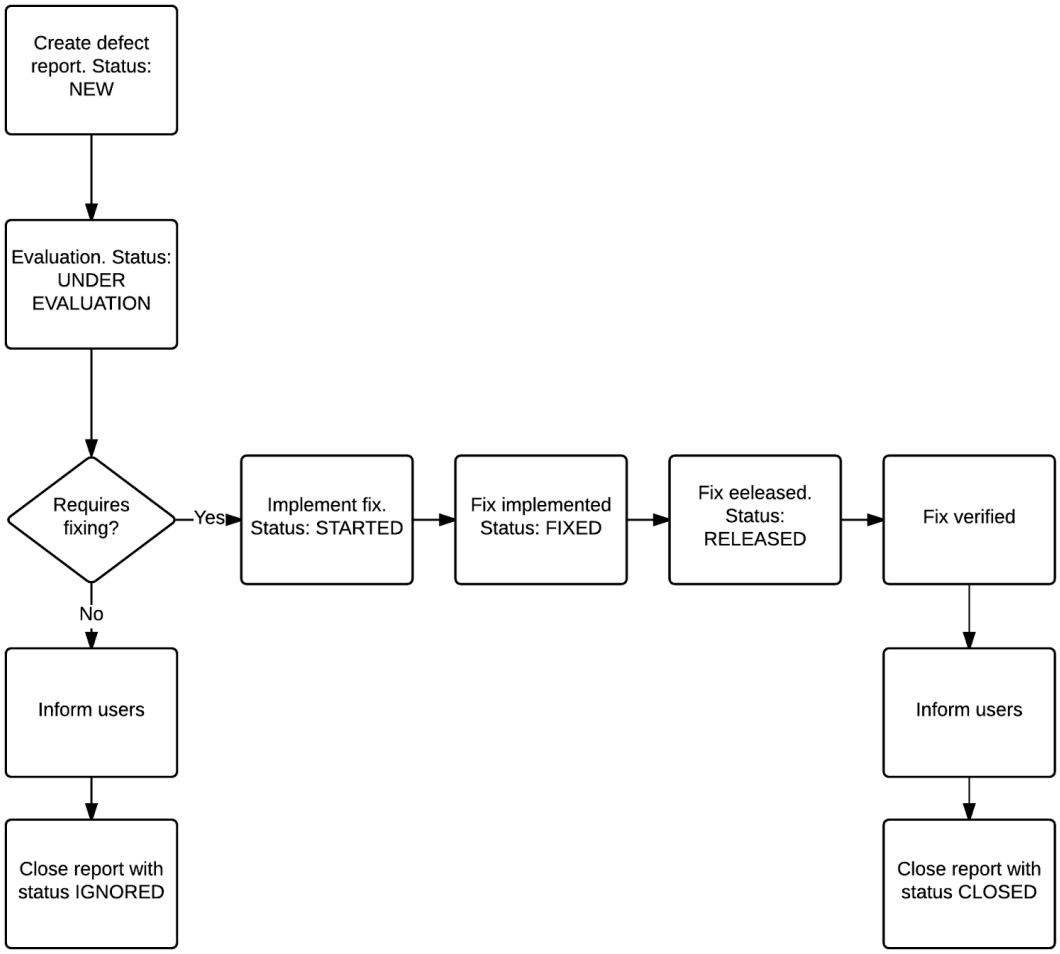

Figure 4.4 - Problem resolution workflow

Some of the key principles of Agile software development are *improvement*, *reflexion*, *quality* and *safety*. Hence an integral part of the problem resolution process consists on reviewing the problems trends, understand what the root causes behind these problems are and adapt the software development processes and practices in order to minimize defects and improve quality and safety. This reflexion occurs both at the **story layer** during the daily stand-up meetings and at the **increment layer** during the team's retrospective meeting.

Regulators, end users of the software product and any other affected parties must be informed of the existence of the problem. Once a corrected version of the software has

been released it must be made available to existing users with instructions on how to apply the upgrade.

4.7 Summary of the guidelines for the creation of medical device software

Creating medical device software consists not only on developing the software as such but also the maintenance of the software during the product life cycle, configuration management, risk management and problem resolution process once the product has been placed into the market. In order to comply with the EU directives regulating the development and manufacturing of medical devices and being granted certification for the final product, certain activities have to be performed and documented during all these phases of the medical device software creation.

The mind map below summarizes the guidelines being proposed:

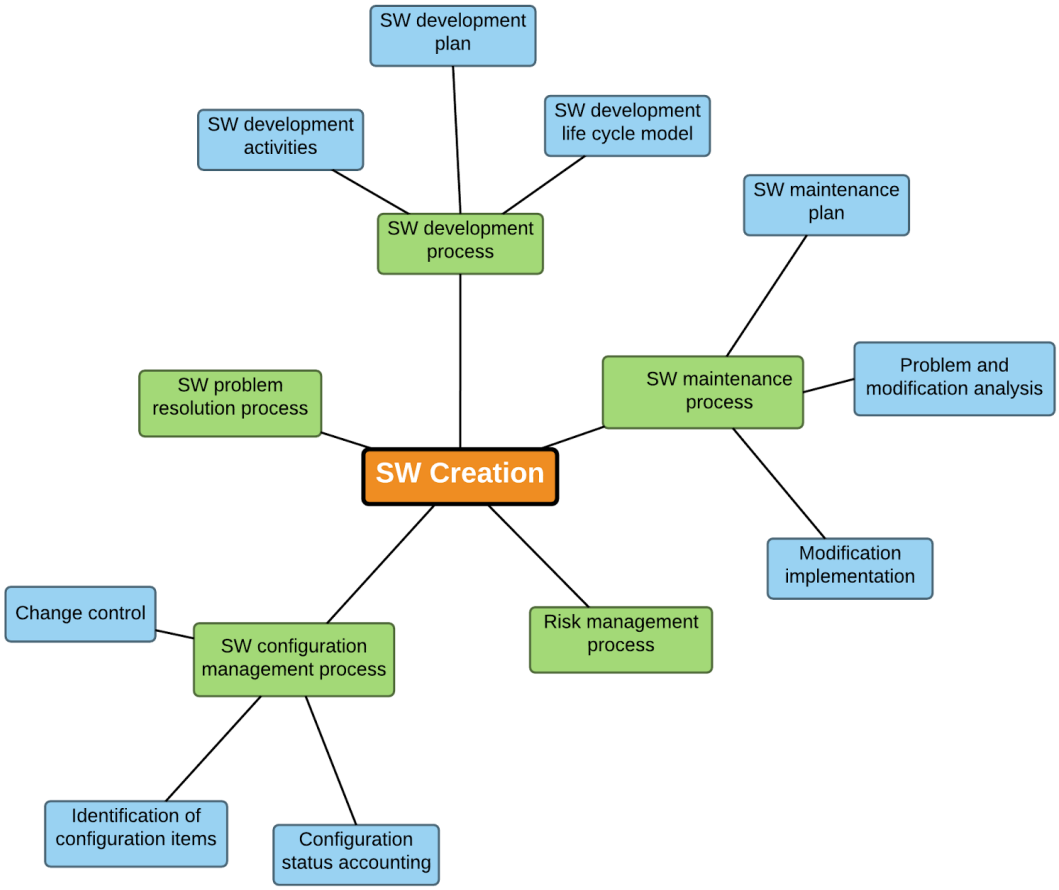

Figure 4.5 - Guidelines for the medical device software creation

In this chapter we have outlined a set of guidelines that the case company should follow when creating medical device software to ensure compliancy with the existing medical device regulations. These guidelines are based on Agile practices that enhance the safety, quality and performance of the software being created. These guidelines have been presented grouped according to the phase of the software life cycle that they apply. The software development activities that have to be performed to comply with the standard IEC 62304:2006 have been mapped to the relevant development phases in an evolutionary and iterative life cycle such as the one being proposed here.

5 Conclusions

This chapter summarizes the work done in this thesis. We will describe the limitations of this thesis and outline the next steps the case company would need to complete before it can place into the market the medical device under development. We will conclude with an evaluation of this thesis by comparing it outcome against the objective described in Chapter 2. We will as well discuss about the reliability and validity of this thesis.

5.1 Summary of the thesis

The case company, Circular Devices Oy, is developing a vision based infant apnoea monitor that is intended to be used for the prevention of the Sudden Infant Death Syndrome (SIDS). As discussed in Chapter 4, any device which primary function is, as defined by its manufacturer, to prevent, diagnose or treat a medical condition is considered as a medical device. Therefore, it must comply with the relevant medical device regulations before it can be sold (MEDDEV 2. 4/1 Rev. 9 2010). These regulations apply to all the components of the medical device, including its software. In Europe, Directive 93/42/EEC defines the rules that regulated the safety and performance of medical devices. Directive 2007/47 EC introduced the last technical revisions to Directive 93/42/EEC.

International organizations such as the International Standardization Organization (ISO) and the International Electrotechnical Commission (IEC) have published a number of harmonized standards to help manufacturers of medical devices to comply with the international regulations. The standard *IEC 62304:2006 Medical device software -- Software life cycle processes* defines the requirements for the software life cycle processes during the software development and maintenance phases. The standard identifies processes, activities and tasks that have to be carried out during the development and maintenance phases in order to demonstrate compliance.

Although the standard IEC 62304:2006 does not impose the use of a particular life cycle, its layout and organization is heavily influenced by a waterfall, sequential software development life cycle. The Association for the Advancement of Medical Instrumentation (AAMI) have published a document titled *Guidance on the use of Agile*

practices in the development of medical device software to address the challenges and particularities faced by manufacturers of medical device software when following Agile methods in the medical device environment.

The case company needs to define its own software development practices in a way that they comply with the European directives regulating the development of medical devices by following the international standards mentioned above. This is the business problem that this thesis sought to solve.

In this work we followed the research steps described in Chapter 2:

First we reviewed a number of case studies that discuss the benefits and challenges faced by established medical device manufacturers when introducing Agile methodologies into their software development teams. One interesting observation was that most of the challenges the authors described had to do with the changes Agile introduced into their current ways of working and corporate cultures rather than having difficulties demonstrating compliance to the standards during audits. The benefits of Agile methods in terms of software quality, productivity and team motivation were apparent. We outlined the Agile values, principles and practices in the context of medical device software.

After that, we analysed in depth the medical device software standards and industry guidelines on the use of Agile. We then carried out semi-structured interviews with industry experts to get insights on how their organizations have introduced Agile methodologies for the development of medical device software.

We concluded this work by proposing an evolutive and iterative software development life cycle and proposing a set of Agile software development practices that the case company can use as guidance when developing the medical software for its infant apnoea monitor.

5.2 Limitations and future research

This thesis presents a number of limitations and opportunities for future research. First of all, there are geographical limitations: this thesis focuses on the requirements set by regulators for the commercialization of medical devices in Europe. However, the

international standards and industry guidelines used in this theses (such as IEC 62304:2006 and AAMI TIR45:2012) are widely recognized by regulators elsewhere in the world (i.e., FDA in the United States). As a future research, we should study the regulations for medical devices in the other geographical areas where they case company intent to sell the device. We should ensure that our software development practices fulfil those requirements and adapt them when necessary.

There are also limitations in scope: this thesis focuses on the software part of a new medical device, concretely on the software development process used when developing medical device software. In order to obtain certification for the new medical device, the complete device has to fulfil the quality and safety requirements set by regulators for medical devices. Quality and risk management processes need to be defined and follow during the development of all parts of the medical device. A future research would include the development of a set of *work instructions* for the development and assembly of the hardware components of this medical device.

Focusing on the software development process being proposed in this thesis, the natural next steps would be to perform a number of development iterations or sprints to validate and improve it. As we discussed in Chapter 5, the Agile principle *reflect and adapt* must be observed in order to adapt the process at regular intervals to constantly improve the quality and efficiency of the work.

5.3 Evaluation of the thesis

5.3.1 Objective vs outcome

The objective of this thesis was to propose a set of software development guidelines based on Agile practices that the case company can follow when developing medical device software in order to comply with the EU Medical Device Directives. After analysing in depth the medical device regulations, the relevant harmonized standards and industry guidelines alongside with the insights gained from semi-structured interviews with industry experts, we proposed a number of Agile software development practices for each of the phases of the software creation process addressed by said standards (i.e., software development plan, life cycle model, software development activities, software maintenance process, risk management, software configuration and problem resolution).

As mentioned earlier when discussing the limitations of this thesis, it would have been beneficial to perform a number of iterations to better refine and adapt the practices being proposed here to the actual environment and needs of the case company.

5.3.2 Reliability and validity

In order to ensure the trustworthiness of this research, we will apply the model proposed by Guba (1981). In this model, the author of qualitative inquiries must consider the following four criteria when conducting their studies: credibility, transferability, dependability and confirmability (Shenton, 2004).

Credibility

In a qualitative study, credibility means that the findings of the study are congruent with reality. Credibility is one of the most important factors in establishing trustworthiness (Lincoln and Guba, 1985).

In this thesis credibility is being achieved by utilizing well defined research methods such as semi-structured interviews, performing literature reviews of the relevant documents, such as regulations, standards, industry guidelines and published case studies and following clear research steps as described in depth in Chapter 2.

Transferability

Transferability is concerned with the extent to which findings of one study can be applied to other situations (Merriam, 1998).

In this study transferability is achieved by reviewing a sufficient number of cases studies where Agile software methodologies are being used in medical device organizations; by utilizing principles, values and practices commonly used and acknowledged by the Agile community and by performing interviews with industry experts with different roles from organizations where Agile has been taken into use when developing medical device software.

Dependability

Dependability is the equivalent to reliability in a quantitative study. Reliability is based on the assumption of replicability or repeatability. Since in a qualitative study it is impossible to measure the same thing twice, dependability is used to ensure that the study could be repeated although not necessarily to obtain the same results.

In this study, dependability is being ensured by defining a clear research design process and research steps as it has been discussed in Chapter 2.

Confirmability

Confirmability refers to the degree to which the results could be confirmed or corroborated by others.

In this study confirmability is being achieved on one hand, by describing the way the research process has been performed in as much detail as possible. On the other hand, confirmability is achieved by acknowledging the limitations of the research as it was discussed in details earlier in this chapter.

References

Association for the Advancement of Medical Instrumentation (2012) Guidance on the use of agile practices in the development of medical device software

Agile Alliance (2013) What is Agile Software Development? 8th June 2013 Retrieved from https://www.agilealliance.org/agile101/what-is-agile/ [Accessed: 4th April 2015].

Beck, Kent, & Beedle, Mike, & van Bennekum, Arie, & Cockburn, Alistair, & Cunningham, Ward, & Fowler, Martin, & Grenning, James, & Highsmith, Jim, & Hunt, Andrew, & Jeffries, Ron, & Kern, Jon, & Marick, Brian, & Martin, Robert C., & Mellor, Steve, & Schwaber, Ken, & Sutherland, Jeff, & Thomas, Dave. (2001) Manifesto for Agile Software Development. Retrieved from http://Agilemanifesto.org/. [Accessed: 26th May 2015].

Drobka, J., Noftz, D., & Raghu, R. (2004) Piloting XP on four mission-critical projects. IEEE Software, 21(6), 70-75.

Emergo Group (2010) Is my product a medical device? Retrieved from: http://www.emergogroup.com/resources/articles/determining-device-classification [Accessed: 20th June 2015].

European Commission (2007) Directive 2007/47/EC of the European parliament and of the council. 5 September 2007. Retrieved from: http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32007L0047 [Accessed: 15th June 2014].

European Commission (2015) Medical Device Guidance Documents. Retrieved from: http://ec.europa.eu/growth/sectors/medical-devices/guidance/index_en.htm#meddevs [Accessed: 15th June 2015].

European Commission, DG Health and Consumer (2010) MEDICAL DEVICES: Guidance document - Classification of medical devices. MEDDEV 2. 4/1 Rev. 9 June

2010. Retrieved from: http://ec.europa.eu/consumers/sectors/medical-devices/files/meddev/2_4_1_rev_9_classification_en.pdf [Accessed: 15th June 2015].

Grenning, J. (2001). Launching Extreme Programming at a ProcessIntensive Company. IEEE SOFTWARE November/December 2001.

Guba, E. (1981) Criteria for assessing the trustworthiness of naturalistic inquiries Articles ERIC/ECTJ Annual Review Paper. June 1981, Volume 29, Issue 2, pp 75-91.

Humble, J., Farley, D. (2011) Continuous Delivery. Addison-Wesley.

Leffingwell, D. (2007) Scaling Software Agility. Addison-Wesley.

Lincoln, Y., Guba, E. (1985) Naturalistic inquiry, Sage Publications.

Mc Hugh, M., Mc Caffery, F., Casey, V., Pikkarainen, M. (2012) Integrating Agile Practices with a Medical Device Software Development Lifecycle. EuroSPI 2012 Conference.

National Center for Chronic Disease Prevention and Health Promotion (2015) About SUID and SIDS Retrieved from: http://www.cdc.gov/sids/aboutsuidandsids.htm [Accessed on 1st September 2015].

Rouse, M. (2015) User Story Definition. Blog post, February 2015. Retrieved from http://searchsoftwarequality.techtarget.com/definition/user-story [Accessed: 15th June 2015].

Schwaber, k., Beedle, M. (2002) Agile Software Development with Scrum. Prentice Hall.

Shenton, A. (2004) Strategies for ensuring trustworthiness in qualitative research projects. Education for Information 22 (2004) 63–75 63 IOS Press.

Shields, P., Rangarjan, N. (2013) A Playbook for Research Methods: Integrating Conceptual Frameworks and Project Management. New Forums Press.

Subramaniam, V., Hunt, A. (2006) Practices of an Agile Developer. Pragmatic Bookshelf.

U.S. Department of Health and Human Services -. U.S. Food and Drug Administration. (2014) Is The Product A Medical Device? Retrieved from http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/Overview/ClassifyYourDevice/ucm051512.htm [Accessed: 14th June 2015].

U.S. Food and Drug Administration (2002) General Principles of Software Validation; Final Guidance for Industry and FDA Staff. January 11, 2002. Retrieved from: http://www.fda.gov/RegulatoryInformation/Guidances/ucm085281.htm [Accessed: 20th June 2015].

Industry experts interviews

Interviewee #1

Company: GE Healthcare

Title: Lead Software Engineer

Position: Epic Owner / Team Lead

Total work experience: 15 years

Date: 1 August 2015

Below is a summary of the interview organized by the SW development topics that were discussed during the theme interview.

SW life cycle and SW development methodologies

GE Healthcare uses an iterative SW life cycle. However, some activities such as final validation and verification, design transfer to manufacturing and clinical pilots in hospitals are being performed sequentially after the other R&D activities have been completed.

GE follows Agile practices. There are some challenges when following Agile though. For example, it is nearly impossible to do content pruning or to drop features of a medical device (i.e., you just can't sell a patient monitor lacking ECG functionality). Also schedules are really tight in the medical device industry because of the way the purchases process typically works in hospitals: budget allocation for new equipment might be done even before the device has been developed. Once the purchase is done, the devices must be supplied on the agreed time and it must contain all the features. Delays on the delivery of the device caused by SW must be avoided.

SW validation and verification (V&V)

SW verification at GE Healthcare tests that the device works as specified. The SW verification is being done incrementally as the SW is being developed. GE follows a test driven development (TDD) process. Integration testing is being done incrementally

within the sprint and as part of the release activities. All tests results at all development layers are logged and documented for future reference during certification or audits.

SW validation checks that the SW meets it intended use. SW validation is carried out in increments during the development process of the device by clinicians at hospital. They use prototype devices as secondary devices so that they can evaluate how they perform compared to the current equipment.

At the very end of the development process and before “design transfer to manufacturing”, a final round of thorough verification and validation (V&V) is then carried out. For this phase some of the same tests produced during the development are re executed as well as new ones to minimize the chances of defects being found after manufacturing has started.

Safety classification

Each of the features or functions of the device receive a safety classification based on the level of control needed to ensure safety and effectiveness of the device.

The highest classification of each of the functions that the device implements will determine the classification of the device itself.

Risk and change management

During the SW R&D, the team must identify and list all the possible SW hazards that might cause harm to the patient. Each hazard is assigned a probability (likelihood that the hazard occurs), a visibility (if the hazard occurs, will it be visible to the clinicians?) and the harm it can cause (e.g., skin burn). A risk classification is then done using these three parameters as inputs.

The mitigation of the risk will depend on its classification. For some, it will be enough to implement some additional safety features by software. Other will require mitigation by hardware. In any case, risk has to be mitigated as far as it is feasible and reasonable. In the case that there is no way a risk could be completely mitigated, a warning must be placed on the user manual of the device.

Roles and team organization

GE Healthcare follows an Agile approach when it comes to team organization and roles definition. It uses roles and artifact somehow similar to those found in Scrum but using its own terminology: a “Lead System Designer” (Product Owner in Scrum) is responsible for the content and prioritization of the product backlog. He is the contact between the team and marketing and sales organizations as well as clinicians working on the development of the device. An “Epic Owner” is responsible for the end to end functionality and performance of a certain application of function performed by the device (i.e., The Epic Owner for ECG measurement in a patient monitor device is responsible for everything related to ECG “skin to screen”, meaning from sensors that are attached to the patient and acquisition modules all the way up to data visualization on the device’s screen). The team is self-organized with work being divided in “sprints” (usually three weeks). The content of the sprint is agreed between the team and the Lead System Designer during the sprint planning meeting. Within the sprint, the team has daily scrum meetings. The scrum master helps the team on the daily work activities and help solving any obstacles the team might encounter while doing their work.

Interviewee #2

Company: GE Healthcare

Title: Research Engineer

Total work experience: 20 years

Date: 15 August 2015

This interview was focused on the Agile Software Development Life Cycle (SDLC) used at GE Healthcare.

GE Healthcare has developed its own Agile SDLC based on various regulations and international standards: MDD 93/42/EEC, 21 CFR Part 820, IEC 62304:2006 and FDA’s General Principles of Software Validation (GPSV).

The purpose of the SDLC is to protect patient and users of medical devices and maintain public safety by ensuring that the software is safe, effective and that risks are at an acceptable level.

The SDLC developed by GE Healthcare support many SW development methodologies. A dedicated “Work instruction” exists that provide guidance to the use of Agile SDLC.

The SDLC must address the requirements for risk management of medical devices.

GE Healthcare performs formal design reviews (FDR) at regular intervals to ensure that the SW being developed fulfils the requirements set by regulators for medical device software. There are three FDR during the SW development process, each focusing on different aspects of the process. During these formal reviews, the development team and other stakeholders (including management) evaluate aspects such as the software architectural design, software requirements, software design, unit implementation and unit testing, software verification, etc. During the last FDR, the SW release and SW maintenance are evaluated as well as the final design validation.

In an iterative and evolutionary life cycle such as Agile, these design reviews are being performed accordingly in an iterative manner and at different depths during the SW development process.