Agile Development of Applications as a Medical Device in a Small Enterprise

Master's Thesis
Remco van der Veen
17 August 2021

Master's Thesis

Business Information Technology

Remco van der Veen
r.vanderveen-3@alumnus.utwente.nl

Supervisors

Dr.ir. J.M. Moonen (Faculty of BMS, University of Twente)

Dr. M. Daneva (Faculty of EEMCS, University of Twente)

F.J. van der Hoek (Fris & Fruitig)

A. Visser (Fris & Fruitig)

Executive Summary

For micro- and small-sized enterprises developing software in the medical device industry, it is challenging to comply to regulations and standards. This research proposes a method to simplify documentation, tool usage and development lifecycle for micro- and small-sized enterprises, by keeping overhead to a minimum while still complying to regulations and standards. We do this by adapting an agile method tailored to the safety-critical industry, called SafeScrum. The method is operationalized and implemented into a micro-sized enterprise using Action Design Research, after which adjustments are made. We conclude that the method provides promising results for micro-sized enterprises wanting to develop software in lower risk classes. More evidence is necessary to give conclusive results.

Motivation for the research
In the medical device industry, about 80% of enterprises are micro- or small-sized enterprises. They are a main driver for innovation in the industry. These enterprises develop safety-critical software which needs to comply to the Medical Device Regulations (MDR) in the EU. The MDR prescribes that the enterprise has a quality management system (ISO 13485) and risk management system (ISO 14971) in place. Next to that, the software development process must be standardized (IEC 62304). However, research shows that only large enterprises implement these practices. Due to limited financial capabilities and resources, micro and small-sized enterprises struggle with implementing a standardized development process into their software development.

The most used software development method in the safety-critical industry, the V-model, is a traditional waterfall method that provides rigorous documentation to show compliance. However, this comes with a heavy overhead. Medical devices can last over 30 years. With such lifespans, a development method that can decrease overhead on development, while adapting to market needs is preferred. Agile methods are seen as a solution, but proof of compliance is lacking. Besides, adaptations are made for medium- to large-sized enterprises. For micro- and small-sized enterprises, these methods stay out of reach. SafeScrum is an adaptation of regular the regular scrum method, including safety measures in its process.

Adaptation
Concerns amongst researchers using agile in this industry are fourfold. Documentation is necessary to prove compliance, but in agile methods, documentation is kept to a minimum. Requirements traceability throughout the development process shows compliance, while in agile methods, requirements are easily changeable. Testing-first processes are common in agile development, but traditional developers might not be used to developing this way. It is unknown whether an agile development lifecycle produces certifiable software applications.

Practitioner interviews learned us that generally the same questions arise amongst micro-sized enterprises in this industry. The standards and regulations define what needs to be in place, but not how these requirements should be met. They mention that short development cycles can be achieved. Tools are important to make this possible, and to reduce overhead on development. Lastly, a common body of knowledge would help them to setup a development method effectively.

We adapted SafeScrum with a method-engineering approach using solutions from research and practice to make it fit in a micro-sized enterprise. We decreased the original seven documents to three documents that need to be maintained. A System and Software Requirements Specification, Development, Safety and Maintenance Plan and System Design document are created. Risk and Hazard Assessment is conducted.

Tools were an important instrument in making the method work for micro-sized enterprises. We automated revision history of documentation and traceability checks by using functionality in existing programs. Quality parameters were checked automatically through application of a static code analysis tool.

Lastly, roles are combined to be suitable for usage in a micro-sized enterprise. SafeScrum defines twelve roles in its process. We brought this down to three roles within the scrum team being the scrum master, product owner and RAMS engineer, and the scrum team having additional responsibilities. An external safety assessor is a fifth role existing within SafeScrum and our adapted method.

Implementation and validation
Using Action Design Research we implemented and validated the adapted method in practice. Two sprints of a low risk classification project in a micro-sized organization allowed us to validate and further optimize the method. Since the enterprise had no experience with standardized agile development or safety-critical software development, priorities on implementation timeline were important.

Running the project hinted towards promising results for the adapted method. Roles and responsibilities were refined during the preparation phase. The documentation was maintainable for the developers. Using tools to automate safety features made it possible to focus on development. Working according to a standardized process was new for the developers. This became obvious through the missing of certain process steps. However, the quality checks caught these mistakes, and they could be improved in following sprint.

Three developers in a team proves to be too small of a team for safety-critical software development. The responsibilities of product owner, scrum master and RAMS engineer are too large to combine this with effective software development. Therefore, a team with two dedicated developers should provide safety, while also gaining enough development speed.

Recommendations
We exposed that regulations and standards are a barrier to market entry for micro-sized enterprises in the safety-critical medical device software industry. The EU stimulates micro-sized enterprises to innovate in this industry. As regulator, they have the responsibility to make market entry possible for these enterprises. A lot of software that is developed is of low risk classes. Adjusted certification would make sense for a combination of small enterprise size and low risk class.

Future research should focus on improving ways to optimize methods for these micro- and small-sized enterprises. As innovators in the industry, they need methods that allow them to develop effectively, while also complying to regulations. Our research has made a start with low risk classes in a micro-sized enterprise. Research is necessary on higher risk classes, where more thorough documentation is required, and a notified body is involved in certification. Next to that, varying team size would build more evidence for the optimal team size in agile safety-critical software development.

We have three recommendations for practitioners, the entrepreneurs in micro- and small-sized enterprises. We have shown that often, these enterprises do not use formalized methods at all, while developing safety-critical software. Start by using a method early, even if it is not perfect. This will help the certification process. Financial and resource investments are necessary when an enterprise decides to develop safety-critical software. Knowledge networks are available of safety-critical enterprises sharing questions and solutions as found in this research. Connecting with these networks allows the enterprise to gain the knowledge they might be lacking on implementing a development method.

Acknowledgements

This research is the result of months of literature research, discussions, execution and input from others. Without them, this research would not have been possible. This part is dedicated to them as a thank you for their contributions.

As my first supervisor, I would like to thank Hans Moonen for your guidance and input. From start to end of this thesis, revisions and suggestions were spot-on and in-depth, even giving me different directions to consider with new literature and discussions. Thank you for providing answers to my high-level as well as very detailed questions. Our meetings gave me the confidence to bring it to this result. Maya Daneva for your effort as my second supervisor. Your input was very thorough, with surprising insights. Thank you both for bringing my thesis to a higher level.

The practical part of this research, the implementation and validation, was conducted at Fris & Fruitig. A big thank you to my supervisors there, Frank van der Hoek and Arthur Visser. I will never forget the endless and enjoyable discussions on development, processes and other matters. Thank you for allowing me to validate this project within the company. I hope that this research allows you to enter the medical device market and is educative on the hurdles needed to be taken there.

I would like to thank the practitioners that I interviewed during this research. Their input proved invaluable to the practical perspective of the research. Lots of innovation gets unnoticed by research and vice versa. Learning from them, their struggles and their solutions, was not only useful for the research, but enjoyable for my own development as well. Good luck on your businesses, I hope they make it in the industry.

A special thank you goes to Arsha Yuditha Amiranti, my partner. Since the start of my research, she has always been on my side. Proof-reading, discussions, provided suggestions, love, and support. *Terima kasih*. Two important thank yous to my parents. They have supported me through my study time, and from time to time persuaded me to not give up and finish it up. And a thank you to my grandma, who said she stayed alive to see me graduate. I hope to have you around for many more years after this.

A thank you to two groups of friends. Libertas est Felicitas and Het Genootschap Bit. During my time as a student, I have always enjoyed my Tuesdays and Wednesdays. We have made countless memories, and I am happy to be a part of it. You have made my student time unforgettable.

To all others whom have not been mentioned but contributed to my study time or the creation of this thesis. Thank you, I appreciate it.

Table of Contents

List of Tables

List of Figures

1. Introduction

Two airplane crashes in October 2018 and March 2019 costed the lives of 346 people. It granted the new 737 MAX from Boeing that was involved the dubious honor of “deadliest mainstream jetliner” (Newman, 2019). Three days after the second crash, all airplanes from the same model were worldwide grounded.

The 737 MAX was an upgrade of the successful Boeing 737, competing with Airbus’ A320neo. This Airbus was introduced in 2010, with a 15% decrease in fuel usage and 10% decrease in emissions, while maintaining the same range. In order to achieve the same efficiency, Boeing mounted bigger engines under the wings of the airplane. Because those engines could push the nose of the airplane up, which could bring it to a stall, pilots would have to be trained to fly with this new airplane type. Boeing wanted the airplane to be as similar as possible to the existing 737, so pilots would not have to be re-educated. In order to achieve this, a new software system was introduced to correct the nose position of the airplane, based on the angle of attack. The angle of attack is determined by a sensor.

Reports from the national transportation safety boards from Indonesia and Ethiopia on the crashes were clear: a faulty sensor determining the angle of attack in combination with the Maneuvering Characteristics Augmentation System, the software correcting the pitch of the airplane (Komite Nasional Keselmatan Transportasi, 2019; Ministry of Transport Ethiopia, 2019). Because pilots were not explicitly informed about the new system or how to disable it in case it malfunctioned, and this system could overrule pilot’s behavior on the control column, the pilots were practically out of control of the airplane, leading to the crash.

Boeing rushed to get its airplane on the market, skipping safety procedures in order to be able to have an answer to a competitor sooner. In addition to the faulty software above, a new problem arose that prevents the flight-control computers from powering up (*Boeing Finds New Software Problem That Could Complicate 737 MAX’s Return - WSJ*, n.d.). These issues show that the place of software in manufacturing is changing. Software is becoming an integrated part of the product and is ever getting more important in the manufacturing of the product. While products used to be mechanic, they are now gathering data with sensors, communicating with other systems while their software controls critical aspects of the product. A new wave of IT-driven innovation and growth is increasing product capability and performance, making products more efficient, effective, safe, reliable and more fully utilized, and improving the ability to meet business and human needs (Porter & Heppelmann, 2015).

The aviation industry is one of the industries that produce safety-critical systems. Systems that, when they malfunction, could harm or cost human lives. Other industries that produce these systems are the space and defense industries, medical device sector, the automotive industry, nuclear reactors and transportation.

Large companies have shown what kind of benefits agile software development bring with short development cycles (Stålhane et al., 2012). A logical consequence is that the safety-critical industries want to profit from the same benefits. NASA and Thales utilize agile methods in their software development (Larrucea et al., 2013). However, these industries have to comply to strict regulations and standards to guarantee safety. This limits their flexibility and possibility for experimentation.

The struggle between cost-effectiveness and safety is one of big firms, but certainly one for smaller companies. The majority of companies in safety-critical industries are smaller- to medium-sized enterprises (SMEs). While they drive the innovation in these sectors, it is especially important for them to use standards for safety-critical systems. However, studies show that SMEs do not use standards, because it costs precious resources that smaller companies cannot spend on overhead

(Girson & Barr, 2018). In return, they see that standardization gives them very little benefits (Herr & Nettekoven, 2018).

In this research, we review the current state of art of agile development in the medical device industry. We give attention to SMEs in our research, since they are an important innovation factor in this industry, while often struggling to use standardized methods to achieve a higher level of maturity within the organization. This research could support further research into practical applications for agile development in the safety-critical industry.

1.1. Background

In this subchapter, we introduce the medical device industry and commonly used terms in our research.

Medical device industry
The medical device industry, also referred to as the medical technology industry (medtech) is rapidly developing. A global market of €385bn in 2016 with a growth rate of six percent per year until 2030 presents opportunities for the healthcare sector (Heuvel et al., 2018; Kuperus & van der Schrier, 2017). Innovative companies anticipate this growth by developing new medical devices. In Europe, around 27,000 companies are active in medical technology, 500-700 being in the Netherlands (MedTech Europe, 2019). Of those, about 95 percent are small to medium enterprises, with the majority employing 50 or less employees (small companies).

In 2010, the 30 largest medical device companies in the world made up for about 89% percent of the revenue generated in the medical device sector, while SMEs are active in the remaining 11% (World Health Organization, 2010). While SMEs are important in creating jobs and innovation, SMEs often have little or no sales revenue (Select USA, 2012; Verdict Medical Devices, 2010).

Medical device software
The digital transformation is raising high expectations for the health sector, looking at electronic patient records, cloud storage and computing, big data, artificial intelligence and blockchain (Oortwijn et al., 2018). New parties, already gathering health data, are entering the market. Big consumer-oriented organizations such as Amazon, Google and Apple are applying their technologies to health. This forces existing medtech companies to find new business models, selling services rather than just hardware. They transform their pricing methods based on value created for patients, called value-based healthcare (VBHC).

Innovation contributes to the evolution of what is considered as medtech (P. Spence et al., 2017). Technological improvements in sensors, together with artificial intelligence and big data, are broadening the definition of medtech to include digital products and data-driven services.

Three types of medical device software can be distinguished: software as a medical device, software supporting medical devices and embedded software (European Union, 2017). Embedded software is software that controls a medical device and is required for its operation, for example for giving instructions to motors to change direction. Software supporting medical devices and software as a medical device are rather new definitions and refer to their intended use. Since software plays an increasing important role for medical devices, software that fits certain criteria is considered as a standalone medical device, for example when the software decides on a treatment. Software supporting medical devices can control the medical device outside of the device itself, for example via a bluetooth connection.

Regulating medical devices
Regulations are a set of rules used to limit the risk of a product causing harm, not fulfilling its intended purpose or not complying with standards of quality (European Union, 2017).

For manufactured products, such as medical devices, the development and certification of medical technology is regulated (World Health Organization, 2010, p. 16). In Europe, the Medical Device

Regulation (MDR) regulates the market, while in the USA the Food and Drug Administration (FDA) has regulations in place to have medical devices certified. Compliance to these regulations is essential for a device to be allowed on the market, since the devices operate in safety-critical environments.

History
The European Union decided in 1985 to start the New Approach, a series of directives to define essential requirements relating health, safety and environmental issues (CENELEC, 2011). This would allow products to be available on a single market. The Medical Device Directive (MDD, 93/42/EEC) from 1993 is one of these directives, that tried to harmonize the legislation of countries in the EU concerning medical devices (European Council, 2009). Several additions have been made in time, to keep in line with market developments and to incorporate substances from human blood or plasma. A notable revision in 2007 was to recognize stand-alone software as a medical device.

For a medical device to be accepted on the European market, it needs to comply to the MDD. The MDD defines four risk classes a medical device can fall into. These risk classes determine which steps a manufacturer of a medical device must take in order to get their medical device checked, and the process of developing it. This is called an assessment procedure. In the lowest risk class, a manufacturer can self-certify its medical device, only having to inform a competent authority, a national agency responsible for this regulation. In higher risk classes, the assessment is done by a third party (a notified body).

Current state
In 2008 the EU adopted the New Legislative Framework, to improve the internal market for goods and strengthen the conditions for bringing products to the market (European Commission, 2008). Part of this framework was the introduction of the Medical Device Regulation (MDR, 2017/745 EU) in 2017 (European Union, 2017). The MDR builds on and replaces the MDD, having similar classification and assessment, but providing stricter rules.

The MDD is a directive that member countries within the EU had to transpose in their national legal system; the MDR is a regulation, that overrides all national law. There is a transition period of three years, which means that all already-existing medical devices need to comply to the MDR in 2021. The most important changes between the MDD and the MDR are (DARE!!, 2017):

- Stricter inspections of devices with a higher risk.
- Stricter criteria and control of the notified bodies.
- The definition of medical device is broadened, to include more (previously non-medical) devices in the regulation.
- An EU-wide database with information containing all medical devices, to improve traceability and transparency.
- Strengthening the requirements for clinical data and post-market surveillance.
- Improved coordination between EU member states.

Standards
Standards are documented agreements containing technical specifications or other precise criteria to be used consistently as rules, guidelines or definitions of characteristics, to ensure that materials, products, process and services are fit for their purpose (ISO, 2018).

The need for standards
Nowadays, we take for granted that we can communicate worldwide through the internet. On the other hand, incompatible power plugs are an annoyance to people traveling a lot. These are two examples of manufacturers keeping to a standard or failing to do so. Standards can provide reference criteria, provide information that enhances safety, reliability or performance, assure consumers about reliability or other characteristics of a product, and give consumers more choice by allowing products to be substituted for another (Cheng, 2003, p. 19). Because most medical devices are used globally, there is an international public health interest to safeguard safety, performance and consistent quality.

Medical device software standards
To be able to develop software for medical devices, software companies need to comply to the European medical device regulation. This regulation states that a quality management system should be in place. Bujok et al. (2017) specify that ISO 13485 (medical devices - quality management systems) is the first step in obtaining certification and CE mark for products. They point out that the QMS is not strictly related to software development issues, therefore, IEC 62304 (medical device software - software lifecycle processes) is also required. This standard again requires that ISO 13485 and ISO 14971 (medical devices - application of risk management) to be in place. Additionally, there is a technical report IEC 80002-1 (guidance on the application of risk management).

The safety-critical environment
"Safety-critical software is software that by failing will endanger people, equipment or the environment" (Hanssen et al., 2018, p. 18)**.** Safety-critical industries produce products that can impact people's lives when they malfunction. Airplanes, nuclear power plants and medical devices can all directly hurt or kill a person when they work differently or stop working altogether. The risk of this happening is considered unacceptable for humans, which is why these industries are highly regulated in order to guarantee safety (Saunders, 2015). Safety in these industries is the freedom from accidental injury (Ericson, 2011).

1.2. Research Problem

Implementation of the medical device regulation (MDR) created harmonization for medical devices, but also potential barriers for entrants to the medtech market, especially for small and medium enterprises (SMEs). According to Oortwijn et al. (2018), it would be helpful to provide support to SMEs regarding implementation and compliance.

Research has focused on companies that are able to implement standards into their working process (Lepmets et al., 2015). There has been research into smaller companies (Özcan-Top & McCaffery, 2017a), and into the development of software with agile (Özcan-Top & McCaffery, 2017b), but to our best knowledge never together.

Due to the number of standards requirements that companies have to meet, it is difficult for small-sized companies to comply. Therefore, small companies often do not use standardized methods to develop their software, even though they could benefit from using them. Agile development poses itself as a lightweight and easily accessible method for this purpose.

1.3. Research Objectives

To create a software engineering method to be used by small companies that are active in or want to enter the medical device industry, which produces safety-critical software that complies to regulations and standards.

- Gather existing knowledge on the state of art in agile safety-critical software development.
    - Find which characteristics agile software development brings to a safety-critical environment;
    - Know which challenges exist for developing software in a safety-critical environment;
    - Find current solutions to adapt agile methods in a safety-critical development environment.
- Find differences between small and medium-sized enterprises (SME) and large enterprises.
    - Know characteristics for SMEs and large enterprises that impact their business;
    - Determine whether company size influences being able to comply to regulations and standards.
- Find differences and similarities between the medical device industry and other safety-critical industries.
    - Determine which regulations and standards are relevant to develop software for medical devices.

- Gather knowledge on the subject of developing software for medical devices.
    - Know how to measure compliance to regulations and standards for safety-critical software;
    - Find the current methods to produce safety-critical software in the medical device industries and comparable industries.

1.4. Research Questions

In this research, we aim to find a method that can be used by SMEs in the medical device industry to develop software. To achieve this, we find an answer to the following research question.

How can SMEs in the medical device industry perform agile software development that complies to standards and safety regulations?

SQ1. What is the state-of-art in safety-critical software development within SMEs for medical devices?

- SQ1.1. What is the current state of art in agile software development methods for safety-critical software?
- SQ1.2. Which factors impact the business for small- and medium enterprises and large enterprises?
- SQ1.3. How does the medical device industry differ from other safety-critical industries?

SQ2. How to engineer a method for agile safety-critical software development in the medical device industry?

- SQ2.1. To which standards and regulations does a safety-critical software lifecycle methodology need to comply in the medical device industry?
- SQ2.2. Which requirements do practitioners have when developing safety-critical software?
- SQ2.3. How can an agile method be utilized in safety critical software development?

SQ3. What is the performance of the engineered method?

- SQ3.1. How does the method perform in practice?
- SQ3.2. How does the method compare to other methods?

1.5. Structure of this Research

Our research consists of ten chapters.
In Chapter 1, we give an introduction into the research.
In Chapter 2, the methodology that we use in our research is described.
In Chapter 3, SQ1.1 is answered, by describing the current state-of-art in safety-critical software development is described based on our literature review.
In Chapter 4, the factors that impact business for small- and medium enterprises are compared to large enterprises, answering SQ1.2.
In Chapter 5, we compared the industries in safety-critical software development, to give an answer to SQ1.3.
In Chapter 6, the knowledge from an existing method, standards and regulations and practitioners are explored, answering SQ2.1 and SQ2.2, to provide a basis for adapting a new method.
In Chapter 7, we create an adapted method to be used in micro- and small-sized enterprises creating software in the medical device industry, and thereby answering SQ2.3.
In Chapter 8, the engineered method is validated in practice, and compared to approaches to create recent COVID-19 applications in several European countries, giving an answer to SQ3.1 and SQ3.2.
In Chapter 9, the findings of our research questions are discussed.
In Chapter 10, we summarize conclusions from this research, describe limitations and give recommendations for further research.

2. Methodology

Within our research, we aim to create a method that allows small enterprises in the safety-critical medical device industry to develop compliant software. In this chapter we outline how the methodology for this research came to be. Table 1 shows an overview of each subquestion, the method how the question is answered, and the chapter where we discuss the subquestion.

Table 1: overview of research questions, methods to answer the question and chapters where the research questions are answered

Research Question	Method Used	Chapter
SQ1.1. What is the current state of art in agile software development methods for safety-critical software?	Literature Review	Chapter 3
SQ1.2. Which factors impact the business for small- and medium enterprises and large enterprises?	Literature Review	Chapter 4
SQ1.3. How does the medical device industry differ from other safety-critical industries?	Literature Review	Chapter 5
SQ2.1. To which standards and regulations does a safety-critical software lifecycle methodology need to comply in the medical device industry?	Literature Review, Practitioners Interviews	Chapter 6.2
SQ2.2. Which requirements do practitioners have when developing safety-critical software?	Practitioners Interviews	Chapter 6.3
SQ2.3. How can an agile method be utilized in safety critical software development?	Method Engineering	Chapter 7
SQ3.1. How does the method perform in practice?	Action Design Research	Chapter 8
SQ3.2. How does the method compare to other methods?	Comparative	Chapter 8.5

2.1. Literature Review

To answer our first research question, we conduct a literature review. The review method of Levy and Ellis (2006) is utilized in our research to provide foundation for the process of finding and appraising literature. The systematic way of finding and reviewing literature follows three stages of reviewing literature: input, processing and output. The processing phase consists of six steps, which ultimately lead to understanding the literature and being able to synthesize new knowledge from it (Levy & Ellis, 2006). This process is shown in Figure 1.

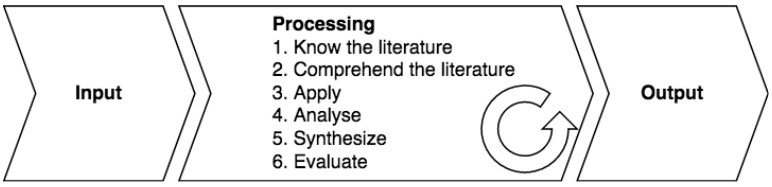

Figure 1: reviewing literature process

Levy & Ellis (2006) start by searching for literature. They propose to search for quality Information Systems (IS) literature in ranked IS journals and ranked and non-ranked conferences. Applicability of found studies can be tested by relevance to build the fundamentals for the theories, constructs and measures. Literature databases such as JSTORE, Elsevier, WilsonWeb and others are advised as sources for finding literature. Keyword search can be the initial start for a search. Based on found literature, search terms can be refined to fit the specific topic that the research is about. Through backwards references search and forward references search, papers are reviewed that are cited by and cite the found literature respectively. Backwards and forwards authors search reviews the literature that authors have published prior and following the found literature. When the researcher

does not find any new constructs in literature or gets the feeling that they have seen or read something similar, the literature review is nearing completion.

In our literature review, we started our search through Scopus with the search terms “quality and medical “software development””. This gave us 159 results, which through review by title and abstract gave us 15 relevant results. These results gave us the insight of specific terms “safety-critical”, “highly regulated” and “MDevSPICE”. Plugging these keywords into Scopus and Elsevier and filtering for research from today until five years ago, the recent literature reviews from Kasauli, Knauss, Kanagwa, Nilsson, & Calikli (2018) and Heeager & Nielsen (2018) gave us a solid foundation for backward and forward searches.

According to Levy & Ellis (2006), the processing phase consists of six repeating steps: knowing, comprehending, applying, analyzing, synthesizing, evaluating. Knowing the literature is characterized by listing, defining, describing and identifying what the conclusion is of a research. Comprehension is shown by summarizing, differentiating, interpreting and contrasting. This includes the extraction of theories, constructs and variables and the models and frameworks they form. Application of the literature is done by identifying the main concepts to the study and placing them in the right category. The fourth step is analyzing, where information is being separated, connected, compared, selected and explained. This shows why the presented information is of importance. In the step of synthesizing the research is made into a whole that exceeds the sum of its parts. The last step, evaluating, compares opinions, theories and established facts and distinguishes them from each other.

The output of the literature review should have a logical structure and shows that the researcher has developed skills and capabilities to the appropriate level (Levy & Ellis, 2006). Arguments and claims should be backed by evidence. A sound argument consists of six criteria: structure, definition, reasons, assumptions, fallacies and evidence (Hart, 1998).

2.2. Practitioner Interviews

In order to retrieve requirements from experts for our adapted method, we interview practitioners. We interviewed three practitioners, who are either active in a small- to medium-sized enterprise in the medical device software industry, or consulting enterprises in that industry. Our interviews followed a semi-structured approach. This method of interviewing allows the researcher to follow a structured approach and receive answers that are comparable from one practitioner to another. The approach also allows the researcher to introduce questions based on an answer given by the practitioner, which opens up a dialogue to gain deeper understanding of the material.

The questions that were prepared for the interviews can be found in Appendix B. The questions can be categorized as follows: background of the practitioner; experience with (agile) software development and methods; findings by the practitioner from expertise, practice and standards.

2.3. Method Engineering

We engineered a method that can be used by micro- and small-sized enterprises in the medical device industry to develop safety-critical software. In order to do so, we found an existing method that we adapted using knowledge from other industries, research and from practice. Mayer et al. (1995) described a method to engineer a method. The approach is documenting motivation, searching for existing methods and either adopt, tailoring or developing a new method based on the found methods. The outcome is then tested and refined. Figure 2 shows the process as outlined by the researchers.

For our motivation, we used the literature review and practitioner interviews we conducted. During our research, we found an existing method which we refined. We integrated findings from other industries, novel findings in research and adaptations necessary for micro- or small-sized enterprises.

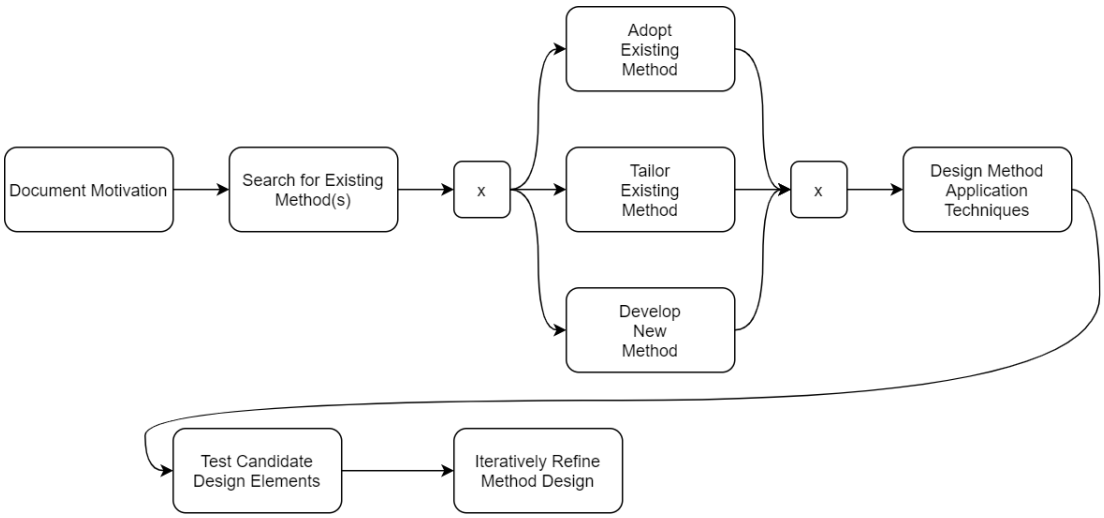

Figure 2: method engineering (Mayer et al., 1995)

2.4. Action Design Research

Within our research, we aim to develop a method that can be utilized by small- to medium enterprises to develop safety-critical software in the medical device industry. Hereby we utilize the Action Design Research proposed by Sein et al. (2011). In a similar way to the iterative nature of agile software development, this allows us to examine our artifact in context and adjust along the way.

An artifact is designed and early in the process tested with practitioners and end users. Researchers can make adjustments during testing and inbetween, improving the artifact by observation of the context. In an agile context, this would allow researchers to improve the process with every sprint iteration.

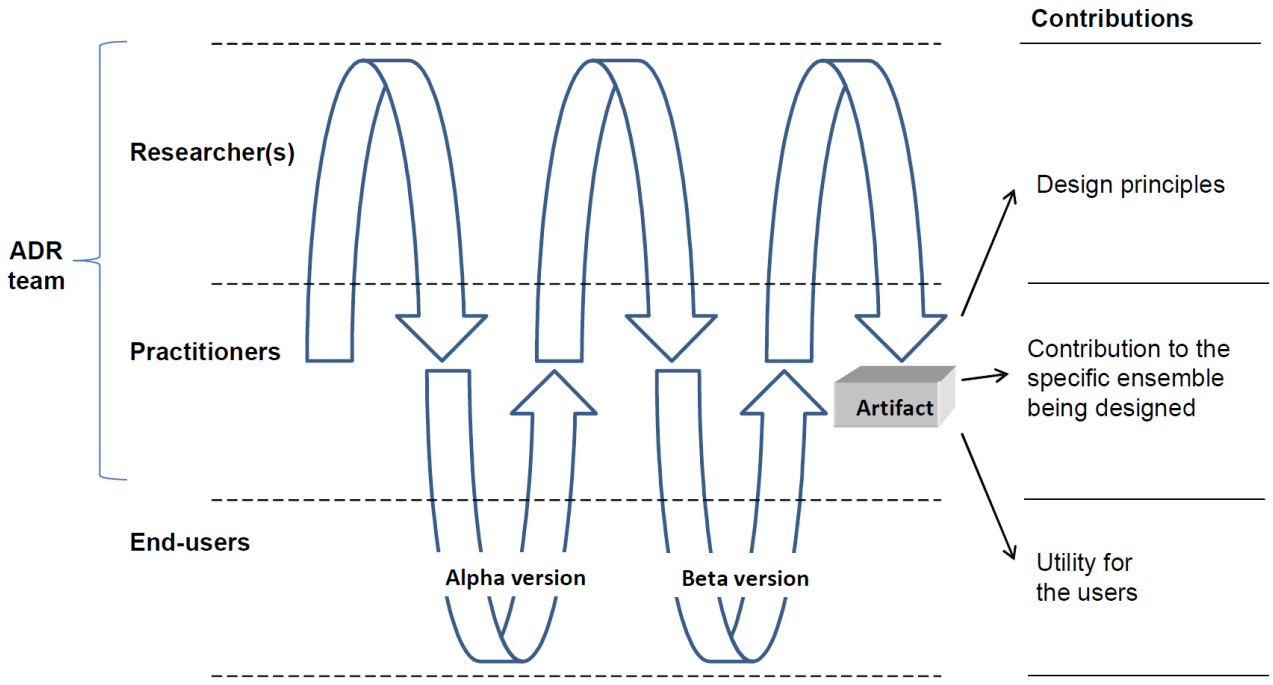

Figure 3: Action Design Research (Sein et al., 2011)

3. Safety-Critical Software

When referring to safety-critical software, we speak of software that can impact lives. More specifically, when the software malfunctions, it can damage or cost human life. Industries that utilize safety-critical software include the automotive, aviation and aerospace industries, railway and transportation, automation, (nuclear) energy and the medical industries. The devices that incorporate this software, such as cars, airplanes and medical devices, are heavily regulated to ensure their safety for humans. A consequence of these regulations is that the software is regulated as well.

Since the 1980s, the importance of software in these traditionally mechanical industries has increased (Larrucea et al., 2013). Complexity and size of the systems increase, so measures for guaranteeing safety need to be revisited. A lot of traditional software development methods for the safety-critical sector are expensive, cumbersome, and as other industries have shown, can produce a lot of overhead and waste. New methods have been proposed, but safety cannot fully be determined with those methods.

Several larger companies are seeking to use agile development methodologies for their safety-critical software. Thales is investigating the use of object-oriented technologies and agile software development methods, and NASA is studying and applying agile development for their safety-critical systems.

3.1. Comparing Methodologies

In the safety-critical environment, mostly the waterfall and V-model are used methods for safety-critical software (Ge et al., 2010; Hanssen et al., 2018). These development approaches can be compared by their characteristic development lifecycle.

Lack of a Software Development Lifecycle Methodology
Companies revert to “chaotic hacking”, as Kettunen and Laanti (2005) call the lack of having a method, for several reasons. A distinction between project initiation, execution and completion is made.

In the initiation phase, two project problems are the primary reasons that companies decide on hacking: unclear project objectives and a lack of resources (P. Kettunen & Laanti, 2005). The lack of a project mission may lead to thinking that the project is ready, when in reality, the product is halfway done. While hacking can save resources, long-term consequences because of left-out documentation can be severe. Hacking is prone to underplanning and leaving out estimating project size, complexity or novelty. New estimates of the project completion day are commonly needed. Phasing and priorities are usually decided by the key designer, which can be an architect. In large projects, hacking easily leads to chaos and bad usage of resources, since coordination is not possible. Lack of competence within the team is directly reflected in the poor quality of the product.

During project execution, working without a method is likely to lead to undocumented code and unspecified interfaces. Lack of documentation makes integration difficult and forces people to share knowledge face-to-face. The requirements phase being skipped or rushed. Next to that, there is usually no systematic architecture design at all, which leads to the risk of using the wrong architecture solutions during the first phases and high cost of reworks. A poor predictability of the project is the result, because the whole schedule depends on a few key persons who might or might not finish the project in time. There are no practices to increase morale, bring geographically dispersed teams together, managing the loss of staff or working with tight schedules, which are all risks. Project cancellation is a considerable risk if the project was setup ad hoc (P. Kettunen & Laanti, 2005).

Finally, when completing the project, issues arise with validating the system: acceptance criteria are determined ad hoc and project end-criteria are undetermined (P. Kettunen & Laanti, 2005). The

outcome might be totally different from the original idea. The undocumented code is hard to maintain and modify and might be unstable or poorly performing.

It is clear that software development methods are necessary to reliably produce quality software. In the following subchapters, we describe the traditional and agile methods and make a comparison between these two methods.

V-Shaped Model

The de-facto standard project lifecycle in safety-critical environments is using the V-shaped model, or in short V-model. The V-model was posed by Rook (1986) and is named after how the lifecycle process is depicted, as shown in Figure 4. It is a heavyweight, traditional method with the characteristics of the waterfall model, where every phase is passed once. Heavyweight methods are characterized by the overhead they require in order to be implemented successfully. Usually, this overhead includes project management staff, thorough documentation of the process and certifications.

It has the well-known waterfall steps on the left side of the shape: planning, defining requirements, building the architecture, creating a detailed design, after which the artifact is implemented. On the right side, every step is mirrored with testing: the detailed design is unit tested, architecture can be tested with integration tests and the requirements are validated by system and acceptance testing. This model provides certainty about validating and verification of the built system, because every step is rigorously tested.

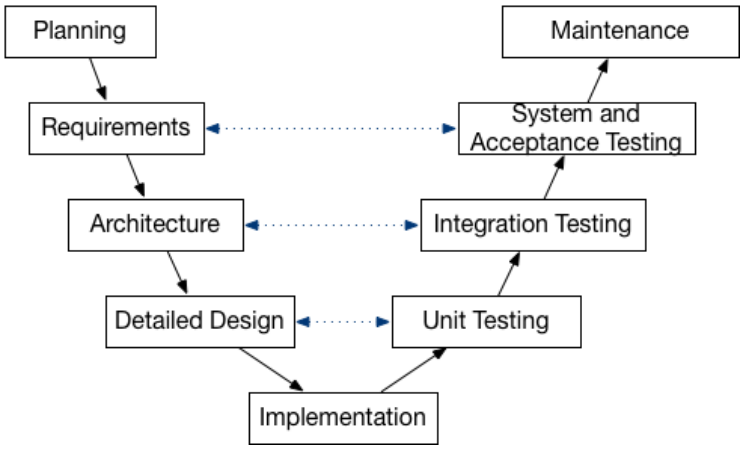

Figure 4: the lifecycle process of the V-model (Rook, 1986)

Agile

As a countermovement of traditional methods, agile is a paradigm for a set of lightweight software development methods that develop incrementally and iteratively, such as scrum and extreme programming (XP). Even though agile methods were developed alongside traditional methods, the underlying values and principles were formalized later in the agile manifesto (K. Beck et al., 2001). Therefore, and because their usage was favored over traditional methods around 2005-2015, these are seen as the new way of working.

Agile methods are typically defined by a short iterative lifecycle such as in Figure 5, which enables the development team to learn from testing and customer feedback. This feedback serves as input for a new cycle, promising a product that has a higher quality and better customer satisfaction. The agile methods are called lightweight because of the focus that these methods have. Four values from the agile manifesto are stated (K. Beck et al., 2001):

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

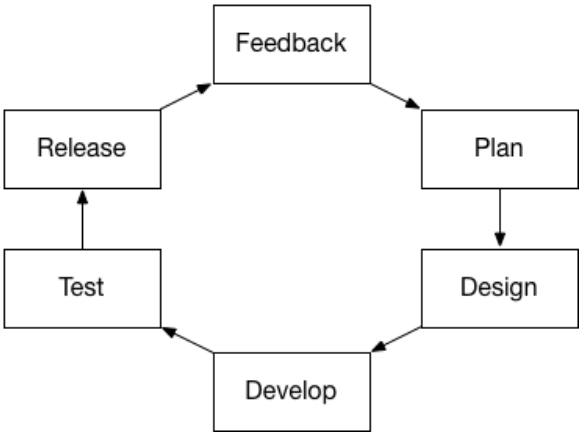

Figure 5: the lifecycle process of agile software development (Myklebust et al., 2016)

A common misconception, as we will see later, is that agile methods prefer to be informal, unstructured and unplanned. According to the methodologists who developed agile methods, the methods are developed to restore the balance between method and people (Highsmith, 2001). They refer to “Dilbertesque corporations” where heavyweight practices have prevailed over productive development work, because of incompetent management overhead – an example is shown in Figure 6. According to Highsmith & Cockburn (2001), “agility... is a comprehensive response to the business challenges of profiting from rapidly changing, continually fragmenting, global markets for high-quality, high-performance, customer-configured goods and services.”

Figure 6: an example of management in a Dilbertesque corporation

Hybrid methods

In the past years, agile has gained popularity over traditional methods, letting the formal processes go in favor of informal meetings and communication for project management. This caused requirements engineering, architecture, quality management and risk management to be done informally or left out altogether. The reasoning for this was that this would obstruct the agile way of working. However, companies that used to work in a traditional way, shifting to agile, or who are bound to regulations, actually implemented a hybrid form of “water-scrum-fall”, as posed by West (2011).

With the current experience that researchers and companies build working agile, many hybrid solutions arise filling in gaps of formal processes in agile. McHugh et al. (2013) mention that mixed

methods should be applied in order to achieve maximum impact in projects. Kuhrmann et al. (2017) notices in their study that indeed, next to fully agile development methods, a combination of traditional and agile methods has become reality. This method emerged from different needs of managers and developers: agile methods are often tailored for system- or development-related tasks, while processes are required by management, such as estimation, planning and controlling are missing (Theocharis et al., 2015). Usually, a combination of requirements engineering, design and architecture (the "water" part) proceeds the development, in which scrum is used. Testing and releasing are done after development, making the "fall" part of water-scrum-fall (West, 2011). An example is depicted in Figure 7.

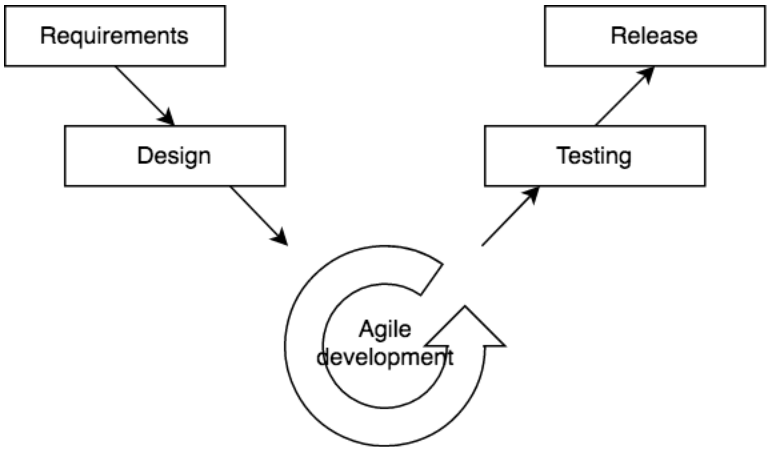

Figure 7: An example of a typical hybrid process (adapted from West, 2011)

In agile methods, architecture has received little attention (Hanssen et al., 2018). While the belief was that the architecture would follow out of the work done in sprints, many agile methods nowadays prefer to define architecture upfront (Myklebust, Stalhane, et al., 2016). Risk- and Cost Driven Architecture (RCDA) is a method that tries to make architecture practices agile, running next to the agile development process (Poort & Van Vliet, 2012). Architects should make decisions on matters to mitigate risk and control costs. In the backlog, a division is made between architectural and functional stories. A similar division is proposed for safety-related stories when developing safety-critical software (Hanssen et al., 2018).

Comparing Methods

When comparing lightweight, agile methods with traditional, heavyweight methods, Awad (2005) found three categories that differentiate agile and traditional methods: project size, people and risk. In this part, we lay out the differences and what this implies for our research.

Project Size

According to Awad (2005) the project budget, duration and team organization make up the project size. More budget or a larger team means a bigger project with more requirements. Traditional methods support this by providing processes and documentation for communication and coordination. The methodology impacts the amount of people necessary to solve a certain problem size (Cockburn, 2000). A depiction of the impact is shown in Figure 8.

Awad (2005) also mentions that a larger team due to a bigger project affects communication and effectiveness person, as shown in Figure 9. Methodologies are a way of structuring communication and coordination (Cockburn, 2000). Therefore, larger projects need a proportional larger methodology, which makes it difficult to keep using agile methods for projects larger than 40 people involved and makes traditional methods preferred. However, this can be solved by splitting larger teams into smaller teams and coordinating them with a "scrum of scrums" (Awad, 2005).

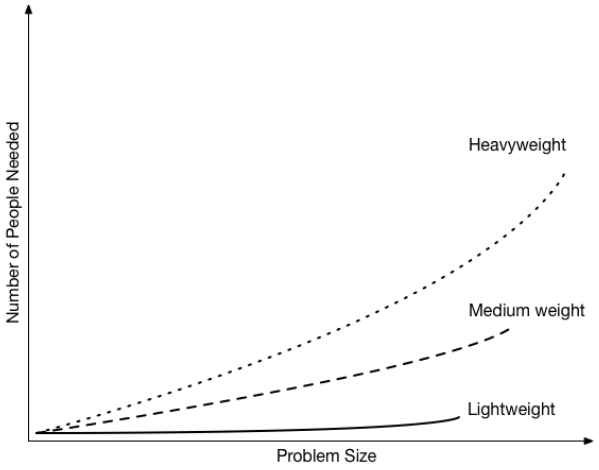

Figure 8: methodology impacting the amount of people for a certain problem size (Cockburn, 2000)

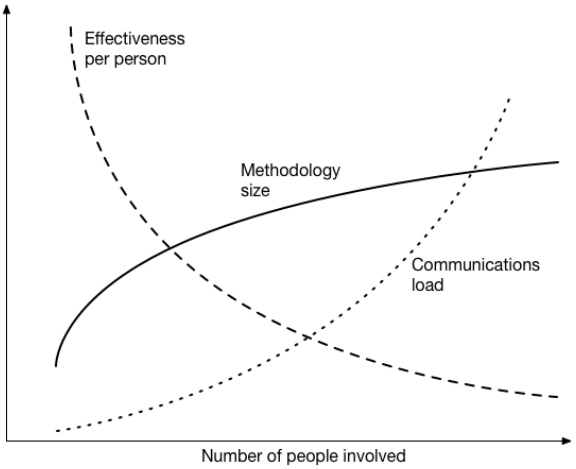

Figure 9: methodology size increases with the number of people involved (Cockburn, 2000)

People Factor

Agile methods emphasize the human factor in methodologies, while traditional methods try to cope with it. This means that when working agile, people are being trusted to communicate, that they are motivated, qualified and produce high-quality results. Therefore, responsibility is delegated to team members. A consequence of this is that people need to be skilled and experienced for the agile method to work well (Awad, 2005). In contrast, in traditional methods, there are checks and balances to prevent and cope with underperformance. There is an overflow in documentation, performance is tightly monitored and there is a strict separation of responsibilities (Awad, 2005).

The organizational culture can either be an enabling or conflicting factor to applying these methods. Awad (2005) states that an organization cannot successfully adopt an agile methodology if it is not responsive to changes or has a lot of procedures.

Risk Factor

Two of the most important risk factors when creating a software process are project criticality and the ability to respond to change, Awad (2005) says. Critical, reliable and safe systems are performed better with a traditional method, since they inherently provide rigorous requirements engineering, quality assurance and project tracking. However, agile methods are naturally better suited for responding to change. Practices such as customer feedback and short iterations allow for better handling changes.

Suitability

Van Dijk (2011) states that determining whether a method is suitable for a project is not something deterministic. Complete suitability only exists in an ideal situation, following a scale for method suitability. The capabilities of a methodology, the capabilities of the organization and the relation with the requirements of the context of the project are three determinants for method suitability. For agile methodologies, these capabilities can be expressed in limitations, and organizational capabilities are expressed in organizational and personnel capabilities. The construct that follows from this is depicted in Figure 10.

Operationalization of this construct is done by answering the following questions (van Dijk, 2011):

1. Which limitations apply to the situation?
2. Is the organization's culture mainly hierarchical?
3. Does the organization currently use agile method components in software developing or had experience with them in the past?
4. Are the personnel capabilities adequate?

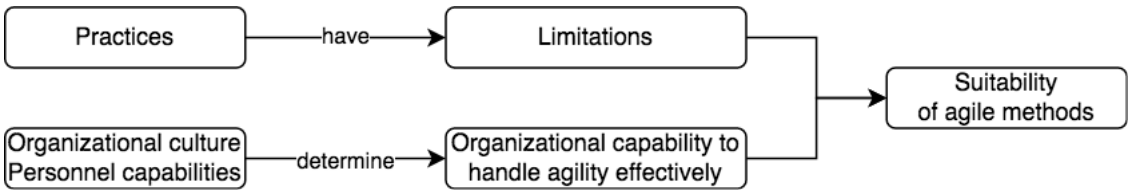

Figure 10: suitability of agile methods in an organization (Van Dijk, 2011)

Awad (2005) found nine discriminators that differentiate agile from traditional methods. In agile, the primary objective is to gain rapid value. Requirements are emergent, could change often and are unknown throughout the project. The size of projects is usually smaller, as is the team. Architecture is only designed for the currently specified requirements. Planning and control are done with internalized plans and qualitative control. Customers are dedicated to the project, knowledgeable, collaborate with the team and can be represented on-site. The developers have an agile mindset, are knowledgeable and collaborate. Refactoring is inexpensive, while risks are left unknown and can have a major impact in the project.

Traditional methods, on the other hand, are highly assured in larger projects and teams. Requirements are known early in the project and stay stable. Architecture is designed for current and foreseeable requirements. Plans are documented, control is based on quantity. Customer interaction is only done as needed, as the project is focused on fulfilling a contract. Developers are usually plan-oriented and have adequate skills. Refactoring in these projects can be expensive, risks are monitored and well-understood, and have a minor impact on the project.

However, Mishra & Dubey (2013) note in their research that the risks in traditional waterfall methods are larger, since a wrong starting point delivers a project that deviates increasingly from the customer's needs throughout the project. Davis, Bersoff, & Comer (1988) already noticed this, and proposed an incremental methodology that would make it possible to develop functionality closer to the user's needs. Agile seems to be the emerging solution that filled this position.

A home-ground analysis can measure how well-suited an organization is for an agile or a traditional method (Awad, 2005). Awad (2005) describes nine discriminators for project suitability of agile or traditional methods: the primary objective, requirements, size, architecture, planning and control, customers, developers, refactoring and risks. The success of a method within an organization can be estimated with a questionnaire, as done by McHugh et al. (2013). They asked six questions about criticality, size of personnel, personnel levels, requirements, culture and dynamism of the organization, to determine whether a traditional or agile method would suit the organization.

3.2. Characteristics of Agile Adoption

Agile methods have inherent characteristics that can be beneficial to companies developing safety-critical software. Agile promises more customer satisfaction, higher quality products and shorter projects by using less formal methods for developing software (K. Beck et al., 2001).

General Characteristics
Overall, agile in a safety-critical environment can improve efficiency. Because agile uses lean manufacturing practices to achieve its goals; if something does not add to the product's final value, it should not be produced (Hanssen et al., 2018). This impact can be seen in less documentation and test cases, only producing ones that are necessary, so they will be used. Reusable frameworks can be built in an agile way, reducing the workload of projects (Kasauli et al., 2018).

The improvement in efficiency makes it possible to bring products to market earlier. This is welcome in the safety-critical industry, where product development usually takes years (Cawley et al., 2010). Simple software design is encouraged in agile methods, making it easier to maintain software (Myklebust, Stalhane, et al., 2016).

Cawley et al. (2010) mention that agile methods like extreme programming (XP), scrum and crystal, are partly aligned with standards for developing safety-critical software. Most requirements in DO-178B for the aviation industry can be mapped to these methods, while some are outside the scope of the method or need to be reworked. However, verification and validation within these methods need proof to be sufficient for certification. Also, refactoring, traceability and motivation play a role in the success of agile methods. We discuss these factors in Section 3.3 and Section 3.4.

Documentation
When using agile, communication is aimed at improving productivity, and therefore performed efficiently. Because of iterative development, only documents that are required for certification are prepared and maintained (Cooke, 2012; Myklebust, Stålhane, et al., 2016). Next to that, documents can be added when they become necessary for management, developers, assessment or customers, instead of when a method requires the documents to be made (Hanssen et al., 2018). Therefore, the available information is used more efficiently (Kasauli et al., 2018).

Requirements
Changing requirements will always be necessary in software development, Heeager & Nielsen (2018) point out. Agile practices are usually more suited to deal with changes compared to traditional methods, because requirements are revisited regularly. Prioritization in agile projects is done by emphasizing high value features first, which leads a thorough definition and validation of important requirements (Kasauli et al., 2018). Safety-related features are these high value features in safety-critical software.

Agile methods bring flexibility into requirements, which improves ability to manage them (Hanssen et al., 2018; Kasauli et al., 2018). Because of iterations in the agile methods, re-planning sprints is possible with the newest understanding of the system (Myklebust, Stalhane, et al., 2016). That puts focus more on innovation, opening up the possibility to change or introduce new requirements, due to changing customer or market needs (Hanssen et al., 2018). Safety requirements might not change that often, but functional requirements changing might have an impact on the safety of the system under development (Kasauli et al., 2018).

Studies, such as Hanssen et al. (2018) propose the use of specific backlogs for safety and functional requirements in a safety-critical context, so that the safety is taken into consideration from the start. This enables integration of safety engineering with software engineering (Stålhane et al., 2012).

Development lifecycle
A safety-critical software project can span several years of development. The iterative nature of many agile development methods forces developers to break down tasks in smaller parts per iteration. This could lead to a deeper understanding of the problem, before attempting to solve the issues (Heeager & Nielsen, 2018). Also, new understanding of the system can cause re-planning (Hanssen et al., 2018). In the V-model the whole project is executed at once, going through each phase one after the other. This leaves little to no space to reflect and improve decisions that have been made earlier in the project.

There are two process steps in agile where customers are involved: requirements engineering and feedback. Because of the iterative lifecycle, this means that every month to two months the development is reflected with the customer. During development, an internal product owner serves as the voice of the customer and makes decisions on design, safety and priorities. The customer, the independent test team and external assessors can supply continuous feedback to the development team (Hanssen et al., 2018). This keeps the team aligned with stakeholders, ensures that customer needs are understood, increases customer orientation and builds trust between parties (Kasauli et al., 2018).

Due to the simplicity of the lightweight agile methodologies, the process becomes more transparent. This gives managers more control over the development process, which helps to deliver high quality projects on time and within budget (Hanssen et al., 2018).

Testing
Through continuous testing and cooperation with the quality assurance roles, product quality risks can be identified and solved early (Hanssen et al., 2018; Kasauli et al., 2018). The use of test-driven development (TDD) produces consistent high quality code for products (Kasauli et al., 2018). In combination with pair-programming, the research of Cawley et al. (2010) shows that test-driven development provides early feedback and better quality. As Paige et al. (2008) notices, test-driven development is compatible with the DO-178B and comparable standards. According to Hanssen et al. (2018), who implemented test-driven development into their SafeScrum method, traceability from requirements to code, and from code to tests is important for certification and can be achieved with test-first development of safety-critical systems.

Test cases also improve in quality when using agile; since focus is not laid on the quantity of tests, but the quality of them, it allows tests to be more aligned with their intent, making them easier to understand and debug (Rasmussen et al., 2009).

According to Kasauli et al. (2018), using agile for safety-critical software improves the safety culture within an organization. Safety analyses can be started earlier in the project, since information becomes available throughout iterations. Correct and faulty system requirements can thus be detected early on (Myklebust, Stalhane, et al., 2016).

3.3. Challenges Using Agile in Safety-Critical Software Development

With safety-critical software development, several issues arise when developed with an agile methodology. Recently, two literature reviews have been conducted by Heeager and Nielsen (2018) and Kasauli et al. (2018). Their researches found similar literature, covering a total of 65 papers. Both researches covered 20 of the same papers, while Heeager and Nielsen (2018) found 31 unique and Kasauli et al. (2018) covered 14 unique papers.

Heeager and Nielsen (2018) included 51 papers. Their research covered agile, XP and scrum in the medical, avionics, robotics and other domains. They found four problem areas, with five relationships that showed up in multiple papers. Figure 11 shows these areas and relationships. We use the problem areas as a guideline, since our literature review found that other researches use similar divisions.

Documentation
Documentation in safety-critical software has two purposes. It can provide communication between developers and development teams and show regulatory compliance by making traceability visible. According to the agile paradigm *working software over comprehensive documentation*, documentation should be "just enough" or "barely sufficient" so developers can focus on working software over comprehensive documentation (Heeager & Nielsen, 2018). This poses a problem while working on safety-critical software. In safety-critical software development, traditionally there is a heavy focus on documentation, because of a fear for regulatory inspections where documentation can show that software complies to regulations.

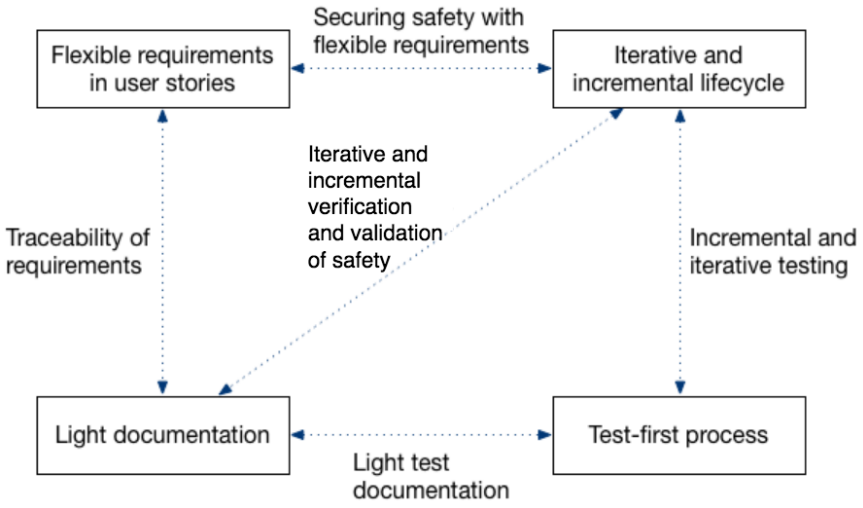

Figure 11: problem areas and relationships within agile safety-critical software development (Heeager and Nielsen, 2018)

In safety-critical software development, there are a lot of stakeholders involved. Kasauli et al. (2018) gives examples of other engineering disciplines, sub-system suppliers, users and certification authorities. The lack of focus on documentation may lead to unclear safety documentation, which hinders the knowledge flow between these stakeholders during development (Wang, Ramadani & Wagner, 2017). A safety expert should provide documentation to the development team. The intended purpose in this documentation could be difficult to comprehend, which could be a roadblock during fast product delivery. The study from Awad (2005) found that knowledge of teams is usually tacitly present within teams, rather than documented fully. This leads to architectural mistakes. Also using agile, architectural decisions may be left out of the documentation when they are predictable.

Development projects for medical device software can take several years. A medical device can have a lifespan of up to 30 years, wherein maintenance of software is necessary. It can be expected that multiple personnel changes take place in this timeframe, or a third party has to maintain, upgrade or improve the software. Therefore, documentation acts as a knowledge transfer medium (Heeager & Nielsen, 2018). Documentation has to provide verification of software safety. Concerns exist about what is considered enough documentation for compliance. Stålhane, Myklebust & Hanssen (2012) raise questions about when a system is sufficiently documented, and when enough documentation exists to proof compliance. They wonder what an assessor would accept as proof of a sufficient documentation and whether a print-out of certain process documents, such as a user story, will be accepted. Without doubt, the conformity documentation should be separated from the agile process, they say. These questions remain to our best knowledge unanswered in literature until now (Heeager & Nielsen, 2018).

Documentation is the primary evidence of traceability of requirements in the software, Heeager and Nielsen (2018) point out. The paradigm of working software over documentation in agile practices inhibits this. Traceability helps establish compliance to standards and regulation (Mwadulo, 2016). Minimal documentation makes traceability difficult. Awad (2005) says that contradicting paradigms are in play. While the agile paradigm calls documentation “a poor way of communicating”, in safety-critical development it “makes it easier to diagnose problems for outside experts”. It might be problematic to find a balance between these paradigms.

Requirements

In safety-critical software development, requirements are well-documented in a complete requirements specification. Complex (medical) devices take years to develop, so it is highly probable that requirements change over time. While safety requirements are usually quite stable, functional requirements change considerably over time (Heeager & Nielsen, 2018). Agile methods are suited for these changes because of their iterative nature.

Kasauli et al. (2018) found that the short upfront design of agile methods might be problematic, because the requirements are analyzed just-in-time during an iteration. The short time span makes it difficult to evaluate the quality of the requirements thoroughly, which could impede certification.

There is a contradiction between flexibility and safety in safety-critical software development. Since agile methods do not provide practical guidelines for change impact analysis, even though they provide flexibility in managing requirements, it can be difficult to manage requirements that change iteratively (Kasauli et al., 2018).

Development lifecycle

Agile differs from the traditional software development lifecycle. In agile, all phases of the development process are included in iterations, to create flexibility and the possibility to adapt to changes. Heeager and Nielsen (2018) notices that literature mentions *iterative* and *incremental* interchangeably. In their paper, they make the distinction between these two terms: iterative development refers to an approach consisting of several cycles, while incremental development refers to the system not being delivered at once. They argue that incremental development is the easier of the two methods to learn, because cutting projects into smaller subprojects is not as hard as deciding when to stop improving a product.

Safety standards prescribe designing upfront, doing hazard identification and analysis. This is done, so certification can be started early in the process of development (Mwadulo, 2016). The V-model incorporates these techniques, while perception is that agile methods are insufficient for safety-critical software development. In iterations, the short upfront design phase and just-in-time analysis of requirements make it difficult evaluate the quality of the safety arguments, which could hinder certification (Heeager & Nielsen, 2018). Next to that, the maturity of agile methods is hard to compare to ISO standards, and to measure it by the Capability Maturity Model (Kasauli et al., 2018).

Due to the iterative and incremental nature of agile methods, during development, refactoring is a common practice during iterations. Refactoring of code invalidates previous certification or security analyses, since compliance has to be shown repeatedly (Cawley, Wang & Richardson, 2010; Mwadulo, 2016).

Testing process

Obviously, safety-critical software requires extensive testing, since malfunctions are unacceptable in an environment where it can damage or cost human life. As shown before, within the V-model, testing is done in the final phases of development. In agile however, test-driven development is widely used (Heeager & Nielsen, 2018).

Kasauli et al. (2018) mentions there is doubt whether agile can provide sufficiently rigorous testing. Agile does not support walkthroughs and code inspections, but focuses on pair programming and informal reviews, which have not proven adequate enough for strict regulations (Awad, 2005). However, agile methodologies can be adapted for safety-critical software, making them useable for systems testing (Paige et al., 2008).

Replacing the testing from traditional methods with agile test-driven development has shown to be problematic, because developers are not used to test in the early stages of development. Next to that, developers write tests themselves. Several standards prescribe that the tester and the developer of a system are separate persons (Heeager & Nielsen, 2018; Kasauli et al., 2018). Otherwise, testing bias might occur by the developer (McBride & Lepmets, 2017).

Testing needs to be automated to gain enough velocity in agile safety-critical systems development (Paige et al., 2008). Velocity is the amount of work that is done within a sprint. A solution that is proposed is the usage of test-driven development. Automated tests cannot cover all code, use cases and requirements. Therefore, in traditional methods, white-box testing such as static verification and coverage testing is done. In agile, these methods are expensive, because they need to be performed iteratively. Acceptance testing is difficult to perform iteratively, because the opportunity to do so is limited and brings high cost.

Light documentation, as agile prescribes, is difficult for testing, given the strict safety standards. Extensive testing has to show that a system is sufficiently safe, which results in large amounts of test documentation. Every iteration should fully be re-tested, to show continuous safety of the system (Ge, Paige & McDermid, 2010; Heeager & Nielsen, 2018). However, that does not prevent that refactoring invalidates previous certification or security analyses (Cawley, Wang & Richardson, 2010; Mwadulo, 2016).

Opinion and perception
Noticeably, throughout the literature we found that opinion and perception play an important role in the decision for using a traditional method over agile methodologies. Kasauli et al. (2018) found a lack of trust in agile methods. According to them, using an agile development methodology makes it hard to determine which level of evidence is sufficient to present for certification. Next to that, within agile methods, the amount of work for delivering a feature seems to be underestimated.

Awad (2005) found that companies that implement agile take a people-oriented approach, rather than a process-oriented approach. This means that the success of the project is dependent on the quality of the developers. Organized and motivated people are required for successful agile projects, and therefore needs management support (Cawley, Wang & Richardson, 2010).

3.4. Adapting Agile

Since agile has been on the rise in business software development, efforts have been made to determine whether agile can be usable for safety-critical software development or can be adjusted to be usable. Research has been directed at finding approaches for using agile in the safety-critical environment. In this subchapter, we discuss ideas that adapt agile methods.

We use the same categorization as the issues we found in Chapter 3.3: to solve issues for providing sufficient documentation, create clear requirements engineering, adjusting the development lifecycle and to have a rigid testing process.

Documentation
The agile paradigm *working software over extensive documentation* is seen as a major mismatch between agile methodologies and safety-critical software development. It does not have to be, though, as the following evidence illustrates.

The pilot study running an agile project method for creating a high-integrity system from Paige et al. (2008) incorporates software tools for generation of documentation automatically out of code. This makes agile compatible with the documentation requirements from standards. Automation prevents excessive time being spent on documentation, keeping the focus on agile working software. They propose to use agile documentation techniques for documents that cannot be automatically generated. Even though less documentation is required because of more face-to-face communication, stakeholder's documentation is still required. A documentation subteam is introduced based on Brooks' "surgical team" model. Stålhane et al. (2012) also proposes a documentation subteam, that will work closely together with the development team to create documentation for the system and to show compliance.

Heeager and Nielsen (2018) continue to argue that documentation does not have to be an inherent problem in agile safety-critical software development. Agile strives to deliver what the customer requested, which can include the wish to include documentation that proves the safety of safety-critical software. That would make documentation a focus area, while still having to consider the purpose of documentation within the agile paradigm. Tools and documentation sub-teams can help with development and documentation focus, they propose.

Requirements
In agile, a product backlog is maintained for features, changes or activities that need to be done within a project. These user stories are given an (estimated) workload and priority. Research

suggests that this backlog is divided into two separate backlogs: one for safety requirements, and one for other (functional) requirements (Myklebust, Stålhane, et al., 2016; Stålhane et al., 2012). Kasauli et al. (2018) endorses this approach: according to their findings, a prioritized backlog with two parts, consisting of functional and safety requirements, should be maintained. The method of SafeScrum for example uses this approach, depicted in Figure 12.

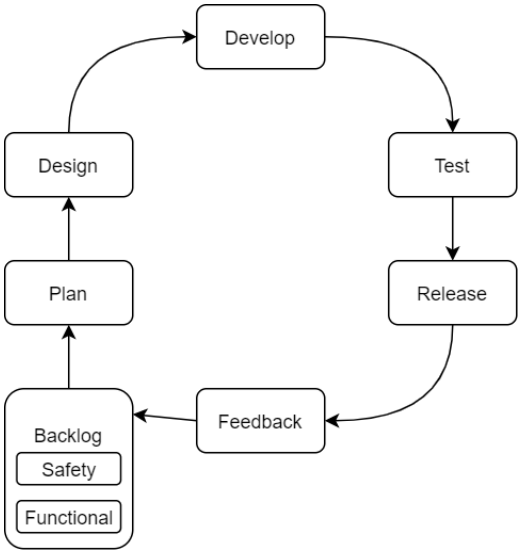

Figure 12: proposed SafeScrum backlog (Myklebust, 2016)

Informal requirements can be expressed in user stories. User stories can provide a good baseline for upfront planning, research shows (Kasauli et al., 2018; Mwadulo, 2016). They provoke discussion between developers and the customer, resulting in better mutual understanding (Heeager & Nielsen, 2018). According to Martins & Gorschek (2016) this communication is the most important factor for successful agile requirements engineering. After discussion, the user stories need to be formalized into specification, by refining them with use case diagrams and textual use cases. They should contain a product definition and high-level requirements. In this way, user stories can be used as a basis for a safety analysis. The upfront design should also be enough to do hazard analysis (Ge et al., 2010). At the very least, some architectural design and the hazard and safety analyses contribute to an essential upfront planning. Hazard lists or checklists can be retrieved from literature and regulations (Kasauli et al., 2018). The hazard analysis is processed on user stories. This can be achieved by refining the user story with use case diagrams and textual use cases. Failure Mode and Effects Analysis is a well-known method to assess potential risks in a requirement. It can be directly applied to user stories.

Cawley et al. (2010) point at two methods to prove traceability throughout the development lifecycle. First, they propose the use of source control management as a best practice to support traceability. Secondly, the TDD process produces a requirements traceability matrix, a matrix that maps requirements to test cases, as byproduct which could show traceability. Fitzgerald, Stol, O'Sullivan, & O'Brien (2013) found that the toolset used during agile software development helps with traceability, since there is transparency in the development process at any moment. They call this "living traceability". The tools can be queried to retrieve the status of the software, in which build certain aspects are implemented and resolved. Developers link their work to certain requirements through their source control. Kasauli et al. (2018) proposes using trace information to determine which tests need to be rerun.

Changing requirements can be managed with a formal change process. This formal change process includes a lightweight change impact assessment (CIA), with quality scoring and prioritization of requirements. The issues that are raised and resolved are maintained in a CIA report. Design reviews for verification and validation also need to be formalized (Kasauli et al., 2018).

Development lifecycle

Developing safety-critical software is about recognizing and mitigating risk, so software can be classified as safe. In order to reach this level, risk can be used as a driver for planning and design.

According to Paige et al. (2008), there needs to be a healthy balance between plan-based and agile approaches in the process of developing. Risk management can be used as a driver to determine the appropriate mix. This risk-based approach can be used during the phases of planning, testing, verification and validation (Kasauli et al., 2018). High-level risks should be identified in the initial phases of the project, while in the whole cycle risk should be considered.

A balance between agile and discipline is proposed for safety-critical software development companies shifting from the plan-driven V-model to agile. According to McHugh, McCaffery et al. (2013) agile and the V-model are not mutually exclusive. Agile can help companies become more flexible to change. A company needs to be suited for “chaos” in their organization, though. Paige et al. (2008) also concludes that agile methods were not designed with safety-critical software in mind but can be adapted for it. Research should not be focused on replacing traditional methods, but rather using strong performing parts from both methods. They propose the development of a hybrid method, that can be adjusted to any project size and complexity.

As explained before, the incremental and iterative cycle of agile development is often seen as a challenge for safety-critical software development, since the developed product should be verifiable and validated to show compliance to regulations. Therefore, iterative development should be in fixed and short iterations (Kasauli et al., 2018). Heeager & Nielsen (2018) found that iterations between six and eight weeks proved the best results: shorter iterations would not provide enough velocity, while longer iterations could lead to loss of organizational focus. The safety analysis and safety validation plan should be done iteratively as well, and the safety case should be incremental with every iteration (Kasauli et al., 2018).

In agile safety-critical software development, the issue of prioritization arises in iterations. To resolve this, Paige et al. (2008) proposes the use of minor and major increments. The minor increments are focused at designing, assessing and implementing, while the major increments will produce software that is ready for acceptance testing. These releases are conditionally safe, since results from testing might require changes in the software.

To keep alignment during development, the customer needs to be represented in all stages of development. The product owner can act as an on-site customer for hazard and safety analysis, the system safety requirements and sprint reviews. Regular communication with the customer makes it possible to adjust to specific needs (Kasauli et al., 2018).

Operating procedures should be standardized and developed by the team (Kasauli et al., 2018). These procedures are part of achieving compliance in agile by management. Short, written policies that the team is familiar with about documentation, review and testing and safety standards are required. These policies can include the requirements change and design review processes mentioned in the previous part.

Besides ownership over the process, teams should also be empowered to deliver and produce software on their own. The code produced is owned collectively by the team, so the whole team can help and knows who did what. Experts in quality assurance and safety should be included in a team, who can interpret requirements and validate implementations (Doss & Kelly, 2016; Kasauli et al., 2018).

As is typical in agile, meetings should be held regularly. In particular, Kasauli et al. (2018) find adjustment for safety-critical software in daily meetings, sprint reviews, planning meetings and retrospectives. Daily meetings should be focused on visibility of safety requirements satisfaction status and promotes collective ownership (Doss & Kelly, 2016). These daily meetings can be in the form of a daily stand-up, to give feedback, communicate, and coordinate technical and organizational dependencies. Teams should set and review weekly goals at the end of every week to focus on team objectives. Deliverables have to be demonstrated to the product owner during a sprint review, which at the same time creates awareness among the team of what has been done. All relevant stakeholders should be included in these sprint reviews. Hazard analysis should be a part of the sprint review, but should be an independent process to ensure quality (Kasauli et al., 2018). In

planning meetings, a software development plan based on features should be created. As mentioned under Documentation, it is important to document modelling approach, implementation language, development environment and assessment tools (Doss & Kelly, 2016). Safety requirements are determined in the planning meeting (Wang et al., 2017). The retrospective looks back at the sprint planning to improve estimations (Fitzgerald et al., 2013).

Testing process
Literature about agile safety-critical software development methods mention test-driven development a lot. In test-driven development, tests are written before the code is produced (Stålhane et al., 2012). This forces developers to think about testing of the code before implementation, and makes regression testing possible. Regression testing is an important technique to ensure the quality of software and makes sure that the quality increases over time (Girson & Barr, 2018). Test scripts and their outcome are accepted as a proof of conformance for IEC 61508, experts say (Stålhane et al., 2012).

To test safety-critical software thoroughly, Kasauli et al. (2018) finds applying continuous development and testing methods useful. During testing and reviewing, a team of peers with roles can perform code reviews or technical reviews. Once a few iterations have been finished, perform unit and acceptance tests. These tests should be automated, risk-based, continuous and extensive, and should include safety tests. Continuous integration and continuous safety builds make it possible to strive for continuous compliance, which facilitates continuous delivery (Fitzgerald et al., 2013). A high coding standard together with pair programming and refactoring support these practices.

3.5. Conclusions

In the past years, agile safety-critical software development has matured. While Paige et al. (2008) conclude in their research that agile should not replace traditional methods, but should produce a hybrid, Kasauli et al. (2018) find that agile can be made to fit in the safety-critical software development lifecycle. We found literature to answer our first subquestion. Four open problem areas are defined and discussed: documentation, requirements, development lifecycle and the testing process. They are summarized in Figure 13.

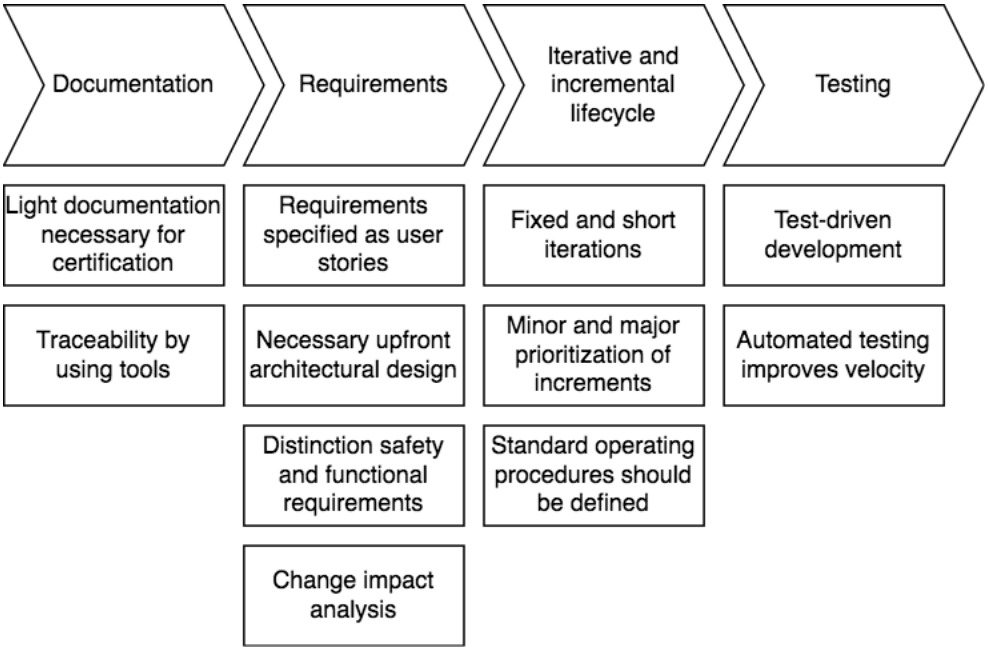

Figure 13: proposed solutions within problem areas

SQ1. What is the current state of art in agile software development methods for safety-critical software?

The span of developing complex products in the safety-critical industry can take years. The lifespan of a medical device is up to 30 years, in which maintenance needs to be performed. Due to the strict regulations that these applications have to comply to, traditional methods provided the necessary processes to cope with these requirements. However, traditional methods have a large overhead, making it costly and time-consuming. Agile methods promised solutions, by emphasizing responsibility of individuals and informal collaboration over formal standards. In the transition from one method to the other, a lot of enterprises ended up with a hybrid method, which became institutionalized. A comparison of these methods is shown in Table 2.

Agile methods force a breakdown of the workload into smaller parts, which improves the planning process for the project and allows for changes in prioritizing and re-planning during the project. The simplicity of lightweight methods improves transparency in the process, helping to produce high quality projects on time and within budget. A shift to produce software for safety-critical devices with an agile method may seem the logical to do.

Table 2: comparison of development methods used in safety-critical industries

	Traditional	Agile	Hybrid	No Method
Project	Large	Small to large	Small to large	Unclear
People	Coping Control	Emphasis Collaboration	Tradeoff management and developers	Face-to-face decisions
Risk	Rigorous methods, not suitable for changes	Tactile, responding to change	Managed by an external method	Project cancellation is considerable risk
Suitability	Hierarchical organization	Gain rapid value	Compromise	None

Documentation of software serves two purposes: communication and showing compliance. Agile methods prefer informal communication over formal documentation, since this could save time and improve efficiency. In safety-critical software however, maintainability of software in long time spans and traceability of decisions are important to show compliance to regulations. Documentation does not have to be a problem when developing safety-critical software, as Heeager & Nielsen (2018) also conclude in their research. When documentation is required to produce working software, it can still be prioritized. Agile methods make sure that no excessive documentation is produced, but only documentation that contributes to the final product. The solution to light documentation should thus be directed at identifying which documentation is necessary to create working, certifiable software. This can be supported by tools, automating the creation of documentation that is necessary for traceability and certification. According to Kasauli et al. (2018) this type of traceability support is underexposed in research.

Requirements in safety-critical software are thoroughly specified upfront, to be able to risk-assess them and prioritize requirements by safety impact. Agile methods make this difficult, since requirements are specified in every iteration just-in-time. User stories, formed with use case diagram and textual use cases, could provide an improved understanding between the customer and developers, while specifying requirements thoroughly at the same time. Necessary upfront architectural design may open the possibility to do risk assessment and safety impact analyses. A reasonable assumption can be made that requirements change over time. Agile methods cope well with change, because they are naturally engineered with change in mind. When requirements change though, their impact on safety is also altered. To assess this, a change impact analysis needs to be performed when requirements change. Researchers assume that safety requirements stay pretty stable, while functional requirements can change over time.

An underexposed issue is the distinction between iterative and incremental in the lifecycle of agile development. Short iterations, alternating between minor design issues and major development issues, should lead to incremental creation of the system. Cawley et al. (2010) state that a lot of focus is given to physical practices, while it would be useful to have research focusing on policies and product lifecycles to support software development teams in an agile manner. Heeager & Nielsen (2018) confirm that this may lead to better development, which is suited better to support safety-critical development. Standard operating procedures is an underexposed topic, according to Kasauli et al. (2018).

Testing has to prove that safety-critical software is free of unacceptable risk. The reports produced by testing are directly used to certify the product, which is required to allow it to be sold on the market. Traditional methods let the produced software undergo testing in the final stages of development. Any negative findings are costly to solve, since it requires a rework of the software. In terms of testing, a lot of research has been focused on user acceptance, unit testing and coding standards (Kasauli et al., 2018). This research could be used to extend from.

The benefits of using the agile practice of test-driven development to develop in the safety-critical environment are clear. Test-driven development emphasizes the importance of testing by forcing the development team to think about writing tests for the functionality they are going to develop, before developing the functionality itself. This shows to improve quality of the produced code and tests, being thought out beforehand. The quantity of tests might decrease, which makes the tests that remain easier to grasp and maintain. Writing tests should be done by a different person than the one implementing the functionality, to prevent bias in testing or implementation. Tests can be extended iteratively, in the process of iterative development. In agile methods, they need to be automated for the process to gain enough velocity, or development speed.

4. The Impact of Company Size

About 95% of all medical device companies are small- to medium enterprises (SMEs) (MedTech Europe, 2019). Larger enterprises (LEs) do not have the difficulty that SMEs experience when implementing a method for safety-critical software development (Lepmets et al., 2016). We compare these scales of enterprises to find differences that lead to these difficulties for SMEs, or the absence of it in large enterprises.

4.1. Definition

In the European Union, enterprises are divided into four categories, shown in Table 3. These categories are defined to create common ground for funding or research from and within the European Union (European Commission, 2003). Two factors determine the category of a company: staff headcount and turnover or balance sheet total.

In Europe, SMEs are defined as companies with less than 250 employees and a annual turnover of 50 million euros or less, or a balance sheet total of 43 million euros or less. SMEs are divided into three categories. The micro enterprise consists of less than ten employees and has a turnover or balance sheet of maximum two million euros. Small enterprises have between 10 and 49 employees, and a turnover or balance sheet of 10 million euros or less. A medium enterprise is all between a small enterprise and the maximum size for SMEs, with between 50 and 249 employees, a maximum annual turnover of 50 million euros or a balance sheet total of 43 million euros. Large enterprises are defined as bigger than SMEs, with 250 or more employees (Eurostat, 2015).

Table 3: enterprise categories in the European Union

Company category		Staff headcount	Turnover	Balance sheet total
Small- and Medium-sized Enterprises	Micro	< 10	≤ € 2 million	≤ € 2 million
	Small	< 50	≤ € 10 million	≤ € 10 million
	Medium	< 250	≤ € 50 million	≤ € 43 million
Large		≥ 250	> € 50 million	> € 43 million

From the non-financial business sector, there are more than 25 million enterprises in the European Union as of 2019 (Muller et al., 2019), and 99.8% of them are SMEs. The vast majority of the total (93.0%) are micro SMEs. SMEs make up 66.6% of employment in the EU, and their value added add up to more than half of the total with € 4357 billion. The full overview is shown in Table 4.

Table 4: enterprise statistics: amount, value added and employees in the EU (Eurostat, 2018)

Company category		Enterprises # (%)	Value added € millions (%)	Employees # (%)
Small- and Medium-sized Enterprises	Micro	23,323,938 (93.0%)	1,610,134 (20.8%)	43,527,668 (29.7%)
	Small	1,472,402 (5.9%)	1,358,496 (17,6%)	29,541,260 (20.1%)
	Medium	235,668 (0.9%)	1,388,416 (18.0%)	24,670,024 (16.8%)
Totals SMEs		25,032,008 (99.8%)	4,357,046 (56.4%)	97,738,952 (66.6%)
Large		47,229 (0.2%)	3,367,321 (43,6%)	49,045,644 (33,4%)
Totals		25,079,237 (100%)	7,724,367 (100%)	146,784,596 (100%)

Compared to the USA, the European system is simple. In the USA, small business sizes are defined by the North American Industry Classification System (NAICS), a classification used in Canada, the

USA and Mexico in an attempt to standardize business statistics. Sizes for small businesses are dependent on their revenue and employee and are different per market. The industry description for “electromedical and electrotherapeutic apparatus manufacturing” defines a size standard of less than 1250 employees as small business (SBA, 2016).

The purpose of defining SMEs in the USA and Europe is to create standardized rules for taxation and subsidizing. Also, definition makes it possible to produce statistics about the certain groups. The difference in definitions in the USA can thus be explained by the way different industries are taxed and subsidized. For our research, we use the European definitions, since it standardizes the definition across all industries which makes it easier to compare SMEs in different industries.

4.2. Internal Features

In a recent research from Mittal et al. (2018), SMEs are compared to large enterprises. They find generalizable features in which SMEs are different from large enterprises. We consider these features and put them in the context of developing safety-critical software for medical devices.

Financial Resources
One of the features of an enterprise, according to (Mittal et al., 2018), is the availability of financial resources. Where enterprises usually have financial resources available by investment or equity, this can be a limiting factor for SMEs, which are often owned by one or a few persons. Also, because of limited assets, it is more difficult for SMEs to obtain credit than it is for large enterprises since they have less collateral (T. Beck, 2007).

Technological Resources
Technology is another defining factor for SMEs and large enterprises. Because of financial constraints, SMEs are limited in their technology adoption, which impacts domains such as quality and human resources. According to Mittal et al. (2018), this prevents SMEs from doing critical research and development. Next to that, IT integration for SMEs is less developed, making the IT solutions of SMEs tailored to solving specific issues faced by the enterprise.

Product Specialization
There are four ways in which SMEs apply competitive strategies to differentiate themselves from competitors: marketing differentiation, segmentation differentiation, innovation differentiation and product service (Julien & Ramangalahy, 2003). SMEs have to specialize themselves, due to a limited amount of resources. These are aimed at differentiating them from competitors and in that way giving them a competitive advantage (Mittal et al., 2018).

Standards and Certification
The use of standards such as ISO is rare in SMEs, due to the resources and preparation required to pass certification (Mittal et al., 2018). Also, frameworks to pursue maturity and for standardization are perceived to be too comprehensive for small companies (Özcan-Top & McCaffery, 2017a). SMEs are mainly externally motivated to pursue ISO certification, to be considered for tenders, increasing their market share, staying in business or gaining market share (Brown, van der Wiele & Loughton, 1998). Internal factors include having a base for quality improvement, improving efficiency or customer service, being a role model to suppliers, achieving a change in company culture or having a new direction after a restructuring.

For the information technology industry, the two most important reasons for certification are to be considered for tenders and to increase market share. For SMEs, certification is an expensive process. Brown et al. (1998) proposes sharing experience with other SMEs and/or involving students in quality programs as good solutions. The biggest disappointment from certified SMEs is that certification is a requirement for being considered for tenders, while customers and governmental bodies are not strictly selecting on these rules. The increase in paperwork and the costs involved is another reason for disappointment.

Denger et al. (2007) conducted a questionnaire amongst 109 companies in the medical device industry to take a snapshot of the practices within the software development industry. They found a lower than expected rate of adoption of sound technologies. While 76% considered themselves building safety-critical software, only the medium- to large enterprises with more than 50 employees use standards such as Capability Maturity Model Integration (CMMI) or Software Process Improvement and Capability Determination (referred to as SPICE, ISO 15504). This is only 18% of the group of respondents. Almost half of the software development teams are smaller than 11 people.

Organizational Culture
Compared to large enterprises, the organizational structure in SMEs is usually less complicated and more informal (Mittal et al., 2018). Because of company culture, innovation is more difficult to pursue than within large enterprises (van de Vrande, de Jong, Vanhaverbeke & de Rochemont, 2009). Within SMEs, balancing innovation and daily tasks and lack of employee commitment or resistance to change are two main hampering factors for practicing innovation. According to Van de Vrande et al. (2009), innovation in SMEs is more focused on external market-related motives, such as meeting customer demands or keeping up with competition.

Decisions are usually made by a decision maker based on “gut feeling”, in contrast to large enterprises, where decisions are made based on market research or analyses (Mittal et al., 2018). In her research, Van Gils (2005) found that in Dutch SMEs, the main decision maker is the CEO. The majority of 60% surrounds themselves with a small top-management team who add to the knowledge within the company. However, the importance of a management team or a supervisory board leading to knowledge diversity should be more emphasized. Knowledge diversity leads to better substantiated decisions. According to Salles (2006), the decision-making process consists of three phases: intelligence gathering, design and choice. Especially in the field of intelligence, SMEs underperform compared to large enterprises which makes their decisions more uncertain. These sounding boards could help fill the gap in knowledge for SMEs.

Employees
The opportunities for employees in large and small- to medium enterprises differ (Mittal et al., 2018). In SMEs, employees are usually expected to have experience in a variety of areas, making their opportunity to specialize less. In large enterprises, employees are usually highly specialized, experts in their fields who know about the state of art in their industry.

According to McAdam & Reid (2001) in large enterprises, knowledge sharing and management is more people-oriented, propagating knowledge through workshops, forums, mentoring and coaching. SMEs focus on more static, mechanistic approach with tools and IT.

Collaboration and Network
Network and collaboration is an important factor for commercial success (Mittal et al., 2018). Due to limited networking opportunities, SMEs might not be familiar with cutting-edge research, since alliances with universities and research institutes are missing. Also, there is a lack of shared knowledge, making it more difficult to learn from their own experience. SMEs are usually dependent on their vendors because their network is not that strong. Compared to large enterprises, who have a large number of options for suppliers and vendors and therefore do not have to rely on one specifically.

4.3. External, Macro-Economic and Political

When looking at macro-economic conditions for success, the diamond of national advantage (popularized as Porter’s Diamond) as shown in Figure 14 comes to mind. With this model, Porter (1990) tried to explain how companies pursue innovation and gain success in a certain market. According to him, four external factors are intertwined in achieving this: firm strategy, factor conditions, demand conditions and related industries. In this subchapter, these four factors are explored to find differences between enterprises of different scale.

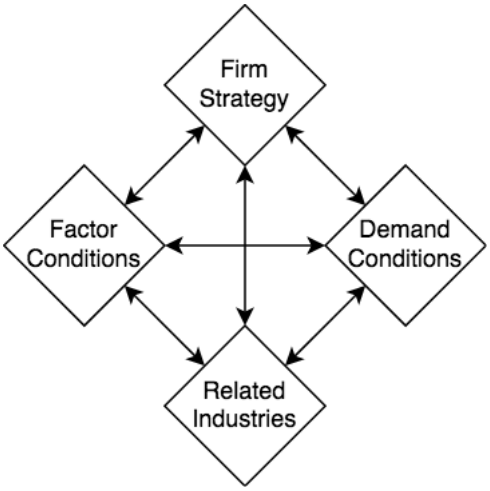

Figure 14: four factors of Porter's Diamond (Porter, 1990)

Different Roles

When zooming out, looking at the larger picture of the economy of a country, the different roles of SMEs and LEs become apparent. Audretsch (2000) mentions that at first sight, SMEs seem to be inefficient, because costs are higher of small-scale production. The organization of an industry with SMEs is sub-optimal, and therefore causes a loss in economic efficiency.

However, SMEs contribute to change and evolution of an industry. Audretsch (2000) calls this the difference between the viewpoint of a static industry, and a dynamic industry. The static industry never changes, while the dynamic industry is subject to change and innovation. It is therefore noteworthy that in the EU, SMEs in the information and communication take a role of innovator (Muller et al., 2019, p. 30). Their contribution to research and development constitutes from high to very high (Muller et al., 2019, p. 27).

Enabling Factors

In developed economies, SMEs thrive better than in developing. Success for SMEs comes from access to finance, the education system and from industrial clusters and global value chains (Herr & Nettekoven, 2018).

Credit is less available to SMEs than to large enterprises (T. Beck, 2007). This has several causes. In countries with macroeconomic and socio-political instabilities, banks compensate the risk by increasing their interest rates (Herr & Nettekoven, 2018). SMEs have less assets available that can serve as collateral than large enterprises. Also, SMEs can be too unattractive for banks to invest in due to difficulties in proving creditworthiness, small cash flows, inadequate credit history, high risk premiums, underdeveloped bank-borrower relationships and high transaction costs (Ardic et al., 2011).

Education of employees is a second enabling factor for SMEs. SMEs thrive on innovation and productivity from employees. Their skillset needs to be adequate to support SMEs in becoming successful. A country can facilitate this by setting up an educational system: primary school, secondary school and if necessary tertiary or vocational education (Herr & Nettekoven, 2018). Proper education alone is not enough though; unemployed academics will move to find employment abroad. Economic measures so the poorer population can afford education, and a safety net for the unemployed is required.

Also, SMEs benefit from industrial clusters. Industrial clusters are companies that are geographically close and share a common market (Porter, 1990). They benefit through their proximity of suppliers, joint use of sources, sharing of labor, more specialized labor supply and knowledge spillovers (Herr & Nettekoven, 2018). A balance between collaboration and competition needs to be found in these clusters. A lack of competition results in rent-seeking and a lack of dynamism; with excessive

competition, companies miss out on collaboration benefits. Horizontal clustering, where resources and knowledge are shared, is an important success factor for SMEs. Clusters need certain characteristics to be successful and usually need governmental help.

Enabling cooperation between research institutes and companies by funding these collaborations has mutual benefits: research can easily test their findings on the market, while companies get product, process or functional upgrades. Herr & Nettekoven (2018) mention that in the specialized suppliers sector, such as specific software, functional upgrading is more frequent, since products or services that meets the needs of customers can only be fulfilled by companies with a high standard of knowledge.

In manufacturing and some services, SMEs are part of global value chains (Herr & Nettekoven, 2018). Instead of producing a full product, every link in the chain produces several tasks to output a finished product. This stimulates technological upgrading and specialization of the enterprise.

4.4. Conclusions

In this chapter, we described the factors that distinguish small- and medium-sized enterprises from large enterprises, as found in literature. Small- and medium enterprises are limited in their capabilities compared to large enterprises. This comes down to seven factors, where the financial limitation is the most prominent, limiting most other factors as well. These factors are summarized in Table 5.

Table 5: comparison summary between small- to medium enterprises and large enterprises

Feature	SMEs	LEs
Financial Resources	Limited	High
Technological Resources	Tailored solutions	More standardized solutions
Product Specialization	Highly specialized	Low
Standards and Certification	Low incentive	Highly standardized
Organizational Culture and Decision Making	Low flexibility By owner or manager	High flexibility By research and analyses
Employees	Experience in multiple fields	Highly specialized Opportunity for knowledge sharing
Collaboration and Network	Specific knowledge Tight relationships with customers and suppliers	Research institutes and universities Low dependence

The financial resources are logically less available in SMEs than in large enterprises. The scale of the enterprise makes it that less assets are available, which means less collateral for getting financing from banks. This causes SMEs to find resourceful solutions to their needs and position themselves differently in the market. Their technological resources are usually adopted when the need arises, whereas large enterprises use fully standardized solutions. Products that are offered are highly specialized in SMEs, tailored to customer's wishes. Large enterprises offer more variety in their products, often being able to accommodate more of the customer demands.

Large enterprises almost always conform to standards and certification, working in a standardized way. SMEs have little incentive to conform to those. Certification trajectories are long and expensive and provide few benefits to the enterprise. Also, frameworks are too comprehensive to be implemented in a small organization. This is notable in the research of Denger et al. (2007), who find that about 18% enterprises are certified through a framework, while 76% create safety-critical software. None of the small enterprises use standards. Therefore, certification should serve a purpose in SMEs, for example meeting customer demands, being considered for tenders or keeping up with competition. Standardization is related to the organizational culture of an enterprise. SMEs thrive on innovation, but because of the limited resources, they are not flexible enough to experiment.

Change and innovation should be focused towards a certain goal. Decisions about these should be based on gathered information instead of gut feeling from the owner or managers.

Knowledge in the industry might be more difficult by SMEs to access than for large enterprises. SMEs have a smaller network and participate less in knowledge sharing with competitors. This results in SMEs having specific knowledge within their industry but missing out on knowledge about cutting-edge research. Employees within SMEs share knowledge through technology and IT. Large enterprises have connections to research institutes, giving them access to recent research. Knowledge sharing in large enterprises is a more organic, institutionalized part of business through workshops, trainings and mentoring.

5. Safety-Critical Industries

Safety-critical industries have two properties: mass and dread (Wears, 2012). Mass, because the malfunctioning of the product or software might result in injuries or loss of human lives. Dread, because people perceive this as out of their control and are unwilling to accept that. Commonly, software accompanies a safety-critical product, such as an airplane or an automobile – things we can all imagine impacting human lives when they malfunction. Industries that deal with this type of software are the medical device industry, aviation, space and defense, automotive, railway, robotics and nuclear industries (Bujok et al., 2017; Heeager & Nielsen, 2018).

5.1. Tradeoffs in Safety-Critical Industries

There are a number of tradeoffs, also called tensions in the safety-critical industries, because they are highly regulated and profit-driven at the same time (Saunders, 2015). Amalberti (2001) and J. Kettunen et al. (2007) mention the balance between performance versus safety. The economic stimulus to generate profit is overridden by the priority of maintaining safety.

Secondly, the tension between centralization versus decentralization of decision-making authority. Centralization works well because it allows a fast, coordinated response, while decentralization works well in uncertain situations because the people with the best knowledge, most skill and position are empowered to act (J. Kettunen et al., 2007).

Another tension is that of redundancy and complexity. Redundancy is often built into safety-critical systems to allow failover when the main system fails. This increases complexity, which makes systems less transparent and can give a false sense in safety margins (Saunders, 2015).

The fourth and last tension is that of the manufacturer's and regulator's different view on safety regulation. In J. Kettunen et al. (2007), the example is given of safety cases in the UK for nuclear plants, which are mandatory and add costs to project, but add little to the safety of the project.

5.2. Comparable Standards

In highly regulated industries for safety-critical software, companies use harmonized standards for quality management, software development and risk management (Bujok et al., 2016). These standards contribute to achieving compliance to regulations.

When looking at standards for quality management, the safety-critical industries use ISO 9001 as a foundation where they build upon. In the medical device industry, the ISO 13485 standard is derived from ISO 9001. The counterpart in aviation and automotive are respectively the BS EN 9115 standard and ISO 16949. All industry-specific standards extend on a generic quality management system standard, adding industry-specific requirements (Bujok et al., 2017).

Risk management is an integral part of quality management in safety-critical industries. The automotive industry uses the ISO 26262-9 standard for risk management, while in the aviation industry has EN 16601-80. In the medical device industry risk management is standardized by ISO 14971 (Bujok et al., 2017).

For the software development process, Bujok et al. (2017) found that more differences exist between the generic and industry-specific standards. In scope, the standards for medical devices and aviation are comparable (Douglass, 2014). The medical device industry uses IEC 62304 and IEC/TR 80002-3, where the automotive industry applies ISO 26262-6 and ISO 26262-8 with ISO/TR 15497. The aviation, space and defense industry use the DO-178C standard (Bujok et al., 2017). IEC 62304 refers to the generic IEC 61508 for the usage of tools, methods and techniques (NEN, 2006).

In industries, research has been going on to create unified methods of working on safety-critical software. SafeScrum is created as an agile method based on the IEC 61508 standard, which

prescribes the process to develop automatic protection systems in the automotive, railway, nuclear and machinery industries (Stålhane et al., 2012). Myklebust, Stalhane, et al. (2016) successfully adjusted this method to be used in the petrochemical industry, where an adapted standard is used, IEC 61511.

The medical device industry has seen research into a unified method for safety-critical software, MDevSPICE. Software companies wanting to start developing applications for medical devices often find themselves overwhelmed by the number of regulatory requirements they have to meet, according to Lepmets et al. (2015). MDevSPICE tries to solve this by being an integrated framework of medical device software best practices. It incorporates a process reference model, process assessment model, assessment method and assessor training and certification scheme. As their research finds, MDevSPICE might prove to be too comprehensive for small companies (Lepmets et al., 2015). Therefore, an adapted, lightweight version for small- and medium-sized companies is under development, named MDevSPICE-Adept (Özcan-Top & McCaffery, 2017a).

5.3. Motivation to Adopt Agile in Safety-Critical Industries

In this part, we compare different safety-critical industries and their motivation to change to agile. A few similarities are found, but factors that are unique to every industry are also highlighted. We compare case studies in medical devices (J. W. Spence, 2005), railway industries (Jonsson et al., 2012), nuclear power plants (J. Kettunen et al., 2007) and aviation industries (Paige et al., 2008).

Similarities
Safety-critical industries have similar reasons to adopt agile in their working environment. For documentation, requirements engineering, process and customer perspective, these are almost identical.

In the petrochemical industry, projects ran with traditional methods are synonymous with being too late and having solid budget overruns (Myklebust, Stalhane, et al., 2016). Paige et al. (2008) criticize undesired documentation costs for the aviation industry. Also in the railway industry, many costly, required documents and other artifacts are seen as a problem of traditional methods (Jonsson et al., 2012). Medical device projects often have to deal with financial planning driving projects and having unrealistic objectives, J. W. Spence (2005) mentions.

For the railway industry, one of the reasons to shift to agile is the difficulty to articulate requirements in the sequential process of waterfall methods, and it is difficult to address new requirements during development (Jonsson et al., 2012). The same issue is mentioned by J. W. Spence (2005) for the medical device industry; he mentions that it is not possible to define requirements completely upfront. In the aviation industry, agile is seen as a way to cope with this requirements volatility (Paige et al., 2008).

Traditional methods are trying to eliminate variability in software development processes by over-specifying how things should be done, as seen in the medical device industry (J. W. Spence, 2005). J. Kettunen et al. (2007) confirm this for the nuclear power industry. The assumption that a focused set of activities can control the safety of a complex system such as nuclear power plants is wrong; it is more complex than that. In the petrochemical industry, the need to change requirements and plans exists (Myklebust, Stalhane, et al., 2016). Also in the aviation industry, flexibility of the development process is desired (Paige et al., 2008).

In some industries, customer involvement is seen as important to the development process. Jonsson et al. (2012) notices that in traditional railway projects, the customer is only little involved. In the medical device industry, software should be developed faster and meet and exceed customer expectations (McHugh et al., 2012).

Differences
Since the safety-critical industries operate in distinct environments, their motivations to adopt agile in their process differ as well.

In the railway industry, the biggest problem with traditional methods is the large and late integration of systems (Jonsson et al., 2012). Railways are managed at a regional or national level, but this type of transportation system often crosses the borders of these regions. Therefore, these systems need to be integrated with each other. Projects are often of a large scale, and late integration of these systems can become costly when errors are found at this stage, because of big rework costs.

The desire to achieve incremental certification is explicitly mentioned for the aviation industry (Paige et al., 2008). At the end of development of a new product in the safety-critical industry, the product is certified by a notified body or a governmental organization. Airplanes get more complex with the introduction of software and updates are becoming standard. Re-certifying the whole airplane, as is the default practice nowadays costs extra work, money and time since components that did not change are also re-evaluated.

Medical device industry
The medical device industry is, at a slower pace than other industries, adopting agile methods to develop their devices and software. J. W. Spence (2005) already concluded that “an agile approach is the approach best suited to development of FDA-regulated medical devices”. However, Fitzgerald et al. (2013) note that a lack of evidence in regulated environments inhibits the agile adoption. They do recognize the need for change of methods, since software is becoming an increasingly flexible and complex part of the medical devices produced. This change is reflected in regulations. While software under the MDD falls into the first risk class and can be self-certified, since 2012, the European Union deems software an active medical device, which shows the importance that software takes and puts it in a higher risk class, requiring a notified body for certification. Twelve years after J. W. Spence (2005), McBride & Lepmets (2017) still state that “medical device software system development will inevitably adopt an agile approach in order to deal with complex problems and situations”.

5.4. Conclusions

Safety-critical industries include the medical device industry, aviation, space and defense, automotive, railway, robotics and nuclear industries. There are a lot of similarities in these industries because the aspect that connects them, safety, is the most important factor. In all industries, tradeoffs between profit and regulations that enforce safety play an important role in business strategy. The standards are comparable, based on a common standard for quality management, risk management and the software development process.

In these industries, tradeoffs have to be made. Four tradeoffs are described: performance versus safety, centralization versus decentralization of decision-making, redundancy versus complexity and the manufacturer’s and regulator’s view on regulation of safety. These tradeoffs are important to comprehend, since a fully safe product can never be created. Companies strive to reach an acceptable level of risk which, in the best case, does not cause any injury or damage at all.

Standards for the safety-critical industries are comparable, studies have shown. If not directly related, they have a common ancestor as foundation. This is true for quality management standards, risk management and software development processes, as summarized in Table 6.

Table 6: standards comparison from safety-critical industries

Industry	Quality Management Standard	Risk Management Standard	Software Development Process
General	ISO 9001	N/A	IEC 61508
Medical Devices	ISO 13485	ISO 14971	IEC 62304
Aviation	BS EN 9115	EN 16601-80	DO-178C
Automotive	ISO 16949	ISO 26262-9	ISO 26262-6

For the medical device industry, an attempt to incorporate standards into a framework has been done in the form of MDevSPICE (Lepmets, Mccaffery, et al., 2015). However, this framework might prove too comprehensive for small-sized companies. Therefore, a lightweight alternative is proposed to be used for SMEs (Özcan-Top & McCaffery, 2017a).

Safety-critical industries have similar and different motivations to switch to lightweight, agile software development processes. An overview is shown in Table 7. Reasons mentioned are projects being too late and having budget overruns, the difficulty to express and change requirements in the process, overspecification in traditional methods to eliminate variability and the wish for customer involvement.

However, different motives exist to shift to agile practices as well. In the aviation industry, an important factor to shift to agile is the desire to achieve incremental certification. The railway industry has systems that should communicate inter-regionally and internationally, but late and large integration of these systems becomes expensive and cumbersome. The medical device industry is behind on the adoption of agile methods to create safety-critical software. The lack of evidence inhibits adoption. Another reason might be that regulation is only recently implemented for software in this industry.

Table 7: similarities and differences between safety-critical industries to shift to agile development

Industry	Author	Similarities	Differences
Medical Devices Aviation, Space, Defense Automotive	(Bujok et al., 2017)	• Tradeoffs are the same in every industry • Industry-specific standards are comparable • Late projects and budget overruns • Using agile to cope with changing requirements	• Differences in software development standards
Aviation	(Paige et al., 2008)		• Incremental certification due to increased complexity
Railway	(Jonsson et al., 2012)		• Large and late integration of systems
Petrochemical	(Myklebust, Stalhane, et al., 2016)		
Nuclear	(J. Kettunen et al., 2007)		
Medical Devices	(Fitzgerald et al., 2013)		• Lack of evidence inhibits adoption of agile

6. Requirements for Adaptation

In this chapter, we provide a basis for adaptation of an agile safety-critical software development method into the medical device industry, fitted to be used by micro- and small-sized enterprises. We do this by giving an answer to two subquestions in our research:

- SQ2.1. To which standards and regulations does a safety-critical software lifecycle methodology need to comply in the medical device industry?
- SQ2.2. Which requirements do practitioners have when developing safety-critical software?

SQ2.1 is answered by Section 6.1 and 6.2. In Section 6.3, we analyze the results given by our interview with practitioners, providing an answer to SQ2.2.

For our research, we use the method SafeScrum (Hanssen et al., 2018). SafeScrum is a method that has been used in several safety-critical industries to transform from traditional methods to agile practices. Therefore, we believe that this method can be adapted to the medical device industry as well.

In order to adapt the method, we need to understand the intended use of the original method. Next to that, we gather new requirements from practitioners and from practice. Together, these premises will be used to adapt the method into a method that can be used by SMEs (Figure 15).

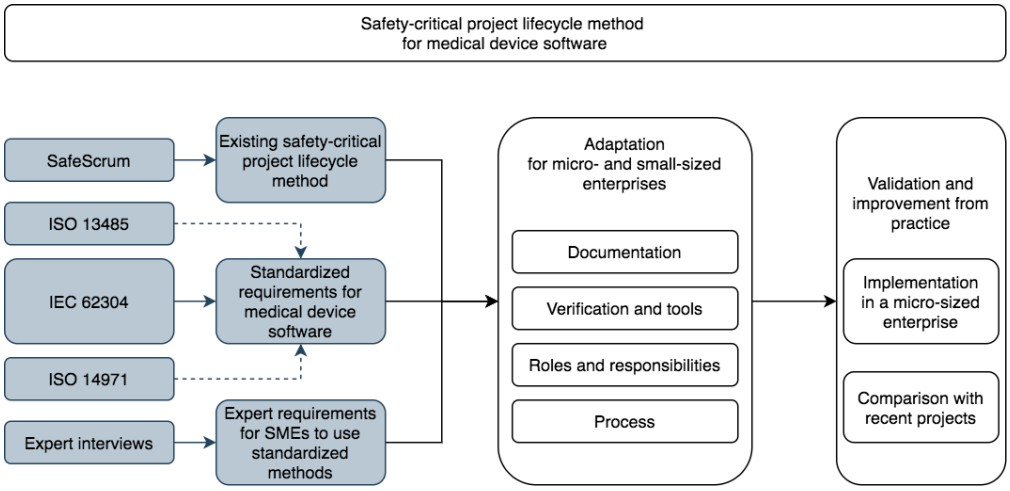

Figure 15: requirements gathering for an adapted method

6.1. SafeScrum

Since we are adapting SafeScrum for the medical device industry, we decompose SafeScrum to see which parts are required or can be adjusted to fit within SMEs in the medical device industry.

Documentation and Stories

Before starting the development process, SafeScrum requires the creation of initial documentation and plans. These documents provide guidelines for development, because requirements, processes and responsibilities are outlined in these documents.

System Requirements Specification

In the system requirements specification (SRS), the requirements for the final product is laid out. Functional and safety requirements form the basis for the product backlog. Other parts of this document might include an overview of the system, lifecycle requirements, information about operation and fault handling, environment, health and safety, and specific information about the system.

Agile Safety Plan
While not strictly required by standards, the agile safety plan (ASP) aims to bridge the requirements from standards in the safety-critical industry and agile software development (Myklebust, Lyngby, et al., 2016). The agile safety plan is set up as a template, which is filled during development as information becomes available in a typical agile fashion. The agile safety plan provides descriptions, deliverables, processes and techniques and measures for the project. The requirements for each topic are adapted from EN 50126, the standard for the railway industry. We listed them in Table 8.

System Design
The system design document contains the high-level design of the system, including the system architecture (outside of SafeScrum), the software design and its components. The system design can be part of the system requirements specification. There are no detailed guidelines on how this document looks, but it should describe how software components are connected to other parts of the system or how the software is divided into subsystems, and their interfaces (Hanssen et al., 2018).

Product Backlog
The defined requirements are within agile transformed into user stories by the product owner, in which functionality is described as a wish from the end user. For example, a user story is normally in the format "*As a [role], I want to [action] to achieve [goal]*". These stories are gathered in the product backlog. The product backlog is prioritized by the product owner, after which stories are refined in a sprint. The functionalities that stories describe, are implemented in sprints.

Table 8: requirements for a safety plan (adapted from Myklebust, Lyngby & Stålhane, 2016)

Descriptions	Engineering and assessment processes	Processes
Policy and strategy	Personnel independence	For safety approval
Scope	Hazard identification and analysis	For system modifications
Roles and responsibilities	Risk assessment and management	For operation and maintenance performance
Lifecycle and safety tasks	Establishment and review of safety requirements	For maintenance of safety-related documentation
Interfaces with other programmes	System design	**Techniques and measures**
Constaints and assumptions	Verification and validation	Checklists
Subcontractor management	Safety assessment	Audit of tasks
Periodic safety audit	Safety audit	Issues of documentation
Deliverables		Review after change in safety plan
Documentation		Review of safety plan after lifecycle phase
Hardware		
Software		

Division of Roles
SafeScrum defines twelve roles necessary to be able to successfully develop safety-critical software. These roles consist of several familiar roles within scrum and agile, with roles added to assure safety. Familiar roles include the scrum master, product owner, the scrum team and a project manager. New roles include the quality assurer, independent tester(s), with a team next to the development team that assures safety: the alongside engineering team. We put the roles into context by their responsibility within the process and documentation in Table 9.

1. The **Scrum master** facilitates the scrum process. This person has to have deep insight into the system under development, and good understanding of safety engineering and standards.

2. The **Product owner** is a part-time role within the team. This person has a good understanding of the market or the customer, as he or she needs to represent the customer and makes decisions on their behalf. The product owner is the main responsible person for the system requirements specification.
3. The **Scrum team** is a team of developers, testers and designers that design, develop and document the solution. These developers take care of the unit tests.
4. The **Quality assurer** (**QA**) makes sure that all quality checks are done throughout the development process, by those who have that responsibility. The QA role can be fulfilled by the Scrum master, a dedicated person serving several teams, or rotated throughout the team.
5. **Independent tester(s)** can be a person who is not part of the team and is a specialized tester, who is responsible for module tests or integration tests.
6. The **alongside engineering team** is responsible for a set of activities that are not directly related to development but are required in order to develop safety-critical software. This includes writing documentation such as the safety plans, verification and validation plan, safety and risk analysis.
7. The **RAMS engineer** is part of the alongside engineering team and is responsible for the reliability, availability, maintainability and safety (RAMS) qualities of the system. Within SafeScrum, the main focus for this role is safety.
8. Coordination of SafeScrum and alongside engineering results
9. The **Independent safety assessor** will assess whether all requirements from standards are fulfilled, by checking the documents received from the RAMS engineer. This role is explicitly not part of the development team.
10. The **Project manager** is responsible for communication and coordination with other project and other parts of the organization. While it is not a role in Scrum, most large projects still include this role.

Table 9: roles and their associated responsibilities within SafeScrum

Role	Process involvement	Documentation involvement
Scrum master	• Product Backlog: Derive user and safety stories • Sprint Planning • Sprints and daily standups • Sprint Review • Sprint Retrospective	• Product Backlog
Product owner	• Product backlog: prioritizing user stories • Sprint Planning • Sprint Review • Sprint Retrospective (optional)	• Product Backlog: creation of user and safety stories
Scrum (development) team	• Sprint Planning • Sprints and daily standups • Sprint Review • Sprint Retrospective	
RAMS engineer	• Safety validation	• System Requirements Specification • Agile safety plan
Quality assurer	• Validation and verification	• Agile safety plan
Project manager	• Overall coordination	• Agile safety plan

Main Process Elements

At high-level, SafeScrum follows the same structure as Scrum (Hanssen et al., 2018). This means that SafeScrum works with a Sprint Planning Meeting, Sprint Workflow, and a Sprint Review Meeting. Additions are made to comply to IEC 61508-3 and due to feedback from the industry. There are three main parts in the process of development with SafeScrum: sprint planning, sprint execution and

sprint review. These three parts are discussed below. A schematic view of these parts is shown in Figure 16.

Before the start of a sprint, a product owner, in collaboration with the scrum master and the RAMS engineer, will define stories within the product backlog. This comprises of describing the story, estimating risk and complexity, defining the criteria that need to be met in order for the story to be considered completed, also called the definition-of-done.

A sprint planning meeting marks the start of a sprint. The scrum team defines the target for the end of the sprint. In the sprint planning meeting, three elements of the sprint are discussed: the sprint goal, team commitment and roles and the creation and division of the sprint backlog. The sprint goal defines in one sentence what the sprint is about, and which functionality of the software is going to be delivered at the end of the sprint. Secondly, the team clearly defines which persons will be a part of the team during the sprint and state their availability. The roles that are changeable within SafeScrum, like the quality assurer, can be appointed during this meeting. Based on priority as set by the product owner, as much stories as possible within the available time are picked from the product backlog and inserted into the sprint backlog. These stories are due to be finished by the end of the sprint.

During the actual sprint, developers take up stories which define a new functionality, improve the software or enhance the safety of the product. SafeScrum implicitly assumes that development teams use a source control versioning system, such as Git or Mercurial. In those versioning systems, a developer often works on a copy of the code base which temporarily diverts from the main code, also called a “branch”. These branches can safely be worked in without affecting the software and can later be merged into a new software version. For every story, a developer creates a branch to work in. When the story is finished according to the developer, with a “pull request” from the branch into the main code, a co-developer is requested to review the code. If the code is not acceptable, the reviewer provides remarks to the developer, which the developer uses to improve the code. The review can also leave problems unresolved. In that case, a quality assurer needs to verify that all quality metrics are met, and traceability is assured. The developer can use this input to improve their code. However, when problems arise in stories of high complexity or risk, these stories are placed on a quality assurance list. Stories on this list will be discussed at the start of the sprint review meeting. A story that is found acceptable is merged into the main code base by the developer responsible for that story.

At the end of the sprint, a sprint review meeting is held. In this meeting, any stories that are finished are demonstrated and approved by the product owner. Stories that have unresolved quality issues are discussed. If an acceptable solution can be worked out during the meeting, the story is approved and finished. Otherwise, the story might need additional details. These details are added to the story, which will then be placed back in the backlog to be developed in the next sprint.

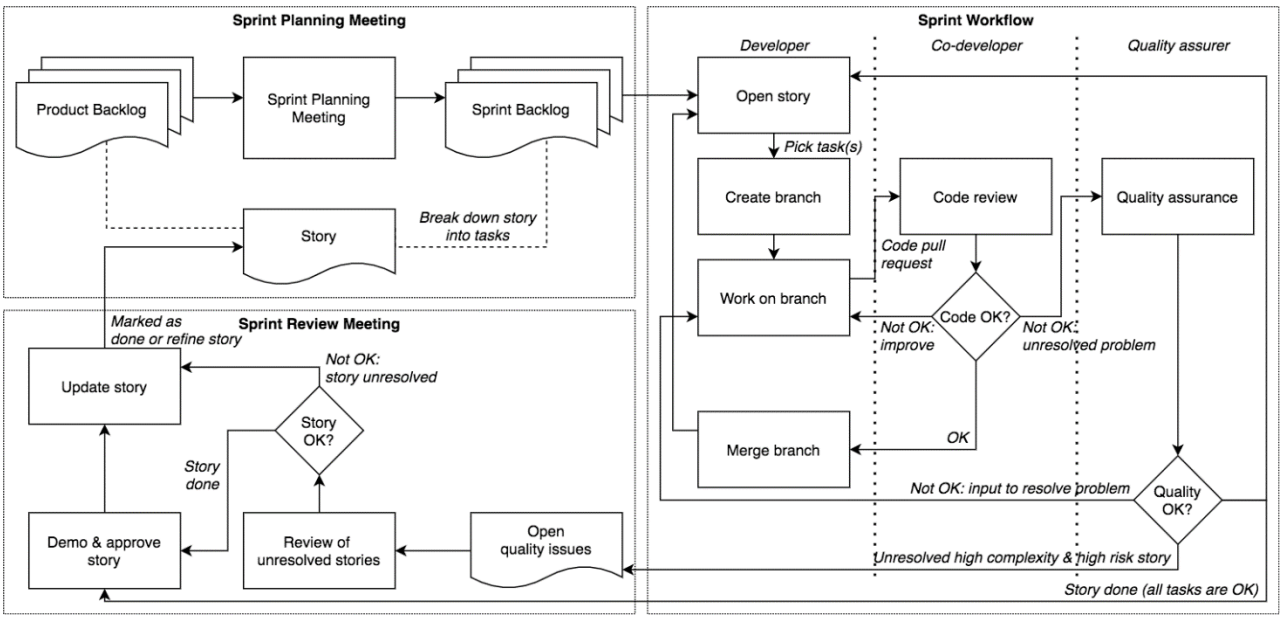

Figure 16: SafeScrum main process elements (adapted from Hanssen et al., 2018)

6.2. Standards and Regulations for Medical Devices

As we have seen in Section 5.2, three standards are important for the development of medical devices and their software: ISO 13485 – quality management for medical devices, ISO 14971 – risk management for medical devices, and IEC 62304 – medical device software – software lifecycle processes. Directly and indirectly, these the use of these standards are the result of the Medical Device Regulation from the EU, which prescribes a quality management system and risk management.

IEC 62304 - Software lifecycle processes
The standard that plays a central role in our research, IEC 62304, describes requirements for processes, tasks and activities for the medical device software lifecycle (NEN, 2006). The software is placed in the context of the V-model, describing phases in order of development planning, requirements analysis, architectural design, detailed design, implementation and verification, integration testing, system testing and release. The standard assumes and mandates that the software is developed and maintained within a quality management system and risk management system. The separate phases are recorded in separate documents.

The software development and maintenance plan describes processes, roles and resources, software configuration, verification and test activities and maintenance. Processes include the software development process, risk management, usability and problem resolution processes. Resources include personnel and technical resources. For software configuration, verification and test activities, evidence of implementation and execution is gathered which is recorded in their own respective documents, listed below.

In the software requirement document, all requirements for the software are recorded. These requirements can be written as user stories and are categorized into functional, in- and output and interfaces, security, user interface, database and regulatory requirements. The requirements need to have a unique identification, so traceability can be acquired in the development process.

After that, the software requirements are to be transformed into an architecture. The standard does not prescribe how this can be achieved. In practice, often a diagram is used to show the architecture of the software (Eidel, 2020c). Also, interfaces are documented in this part. These can be interfaces that communicate with other parts of the software.

Software verification needs to be documented as well, according to IEC 62304. The purpose of verification is to make sure that the software implementation meets the written requirements. We discussed methods of verification in Section 3.3 and 3.4. Methods include static verification, code reviews by other developers in the team and unit and integration tests. The processes and evidence, like test reports, are included in the software verification.

System testing includes verifying that all software requirements are met. This can contain software verification, but also includes software requirements that cannot be tested by automation. Therefore, a software system test plan which is performed manually, with screenshots or configuration can serve as evidence that requirements such as firewall settings have been implemented.

Lastly, for software release, IEC 62304 sets a number of conditions that have to be met and recorded. The software verification must be completed, activities and tasks are completed, anomalies during the development process are documented, the released version is recorded and archived, and the delivery of the released software needs to be reliable.

ISO 13485 - Quality management for medical devices
Relevant for the application of IEC 62304 as a part of the quality management system is that software that is used within the organization is also validated (ISO, 2016, sec. 4.1.6). This means that software that is used for the development of medical device software, also called tools within this research, within the enterprise is listed and that relevant requirements for selection are chosen.

ISO 14971 - Risk management for medical devices
To guarantee safety, an important part of building safety-critical software is risk management. The MDR requires a risk management system to be in place, referring to ISO 14971. As in any risk management, steps in the process include identifying hazard and hazardous situations, estimate the risk that is accompanied by these situations and evaluating the risk. Estimation of risk is done by the potential harm that a risk can induce, as categorized in Table 10, and the probability that a risk can occur, which is shown in Table 11.

Table 10: risk severity levels (ISO 14971)

Term	Description
Catastrophic	Results in patient death
Critical	Results in permanent impairment or life-threatening injury
Serious	Results in injury or impairment requiring professional medical intervention
Minor	Results in temporary injury or impairment not requiring professional medical intervention
Negligible	Inconvenience or temporary discomfort

Table 11: risk probability occurrence (ISO 14971)

Term	Probability range	Written
Frequent	$\geq 10^{-3}$	Once in every 1000 times or more frequent
Probable	$< 10^{-3}$ and $\geq 10^{-4}$	Between once every 1000 times or once every 10000 times
Occasional	$< 10^{-4}$ and $\geq 10^{-5}$	Between once every 10000 times or once every 100000 times
Remote	$< 10^{-5}$ and $\geq 10^{-6}$	Between once every 100000 times or once every 1000000 times
Improbable	$< 10^{-6}$	Less than once every million times

When these two factors are outlined in a matrix, their intersection form the risk level (Table 12). The risk level can be low, medium and high. With every risk, an estimation should be made whether a risk is acceptable or not, and whether or not risk reduction is required. For medium and high risk, it is required to perform a benefit-risk analysis.

Table 12: risk matrix based on severity and probability

	Negligible	Minor	Serious	Critical	Catastrophic
Frequent	Medium	Medium	High	High	High
Probable	Low	Medium	Medium	High	High
Occasional	Low	Low	Medium	Medium	High
Remote	Low	Low	Low	Medium	High
Improbable	Low	Low	Low	Low	Medium

6.3. The Perspective of Practitioners

While safety-critical industries share a lot of similarities, they have their own standards and difficulties that industries have found solutions for. To gather knowledge from the industry about their experience in developing software for medical devices, we interviewed practitioners from the industry using a semi-structured approach, as we explained in Section 2.2. Our practitioners are either active in an SME that develops software for medical devices or are indirectly involved as consultant to these enterprises (Table 13). The interviews were performed one-on-one, during which we used the questions from Appendix B. and noted the answers. These notes are qualitatively compared and analyzed. The findings of this analysis are summarized in this subchapter.

Table 13: practitioners interviewed for our research

Practitioner	Years active in industry	Position
Practitioner 1	12	Head of Regulatory Compliance
Practitioner 2	4	CTO
Practitioner 3	7	Medical Software Consultant

Lifecycle
Development of software in the medical device industry largely happens agile, according to the practitioners. Their team are mainly kept out of the process of making the software certifiable. In the enterprise from practitioner 1 and 2, two people are responsible for regulatory compliance: the Chief Technology Officer and a regulatory compliance employee. Practitioner 1 mentioned that they are trying to involve the development team in safety practices. This proves difficult due to lack of interest from developers, originating from the viewpoint that it distracts them from their software development tasks.

The length of the iterations is a subject that many practitioners are interested in improving. Our literature showed that a sprint time of six to eight weeks would be ideal (Heeager & Nielsen, 2018), since shorter sprints would not generate enough velocity. Our practitioners mentioned cycles of four weeks and even two weeks as being ideal for them. This allowed for the necessary documentation and gained enough velocity for actual development. Practitioner 1 explained that the cycle of four weeks is divided into three parts: one week for preparation of development, in which they assess the impact of changes for the current sprint, two weeks of development, followed by one week of verification and validation. A cycle of two weeks was realized by practitioner 2 by performing regulatory work by a dedicated employee and parallel to the team. His reasoning for this shorter development cycle was that this allows them earlier verification and validation, in the spirit of agile development (K. Beck et al., 2001). The idea of a separate regulatory team is supported by research (Hanssen et al., 2018).

Tools
The practitioners emphasize the importance of using tools during the development process in order to create documents, keep track of the process and to automate important verification and validation. These tools help them to decrease the overhead that comes together with developing software specifically in the safety-critical industry, by automating repetitive tasks. It can also help to standardize and increase safety, according to them, because important steps cannot be forgotten

when they are enforced by tools. In testing, the same test suite will run, no matter which developer runs the tests.

According to our practitioners, several tools are commonly used in the industry. For requirements engineering in the medical device industry, Matrix Requirements is a tool that all experts use. Atlassian JIRA is the de-facto standard for defining user stories and keeping track of sprints, according to them. GitLab and GitHub are mentioned as code repositories. A slight preference for GitLab is heard from a few experts, because of better Continuous Integration and Continuous Delivery (CI/CD) integration.

In one of the enterprises, an attempt was made to start with an electronic quality management system (eQMS) fully based on GitLab. In theory, this should be possible using digital signatures and code reviews. However, attempts were aborted once it became clear that FDA approval was preferred. According to the practitioner from this enterprise, the FDA is stricter and more conservative in their requirements concerning traceability in a paper trail.

Lastly, all three practitioners agree that software reuse should more closely be studied within the safety-critical domain. They mention that de-facto standard software, such as frameworks and libraries, are being used. This is acceptable for notified bodies, as long as these frameworks are either tested, or not part of the critical part of the software. Practitioner 2 says that their notified body is interested in proportionality of risk and whether the libraries are documented over testing.

Lessons Learned and Improvements
All of our practitioners mentioned a lack of pre-existing knowledge within the industry. A lot of startups or SMEs try to implement the requirements themselves, based on the standards and regulations. However, standards and regulations prescribe what needs to be in place, without elaborating on the process, tools or knowledge that is required to achieve compliance. This can be especially challenging for startups and small enterprises, since they commonly do not have the knowledge in-house to interpret the weight of certain requirements. The practitioners thus emphasize the need for a body of knowledge that is easily accessible to these startups. Common issues that arose were:

- Which tools are appropriate for safety-critical software development for medical devices?
- Which programming language is acceptable to use for software for medical devices?
- How to make the development cycle faster?

Another consideration from one of the practitioners was the focus on standards. Their initial focus as a software development team was on the IEC 62304 standard, describing the process for developing software. After consulting with a notified body, the principles of ISO 13485, the necessity for a quality management system, showed more important to the auditor. The auditor is also more interested into documented considerations about decisions, than into formal process documents, according to this practitioner.

The alternative, as practitioner 3 mentioned, is involving an experienced (external) consultant or hiring a regulatory and compliance employee from the start. This, however, brings higher starting costs, which might prove difficult for startups or SMEs.

6.4. Conclusions

In this chapter, we have gathered requirements for a method from three sources: an existing method for safety-critical software development, from standards and regulations in the medical device industry, and from practitioners in the medical device software industry.

7. Adaptation for Small Enterprises

The method that we are developing in this research is grounded on preceding research, an existing method, experience from practitioners and requirements from standards and regulations; requirements we gathered in Section 6. In this chapter, we outline how we incorporate these parts into an adapted method for micro- and small-sized enterprises (Figure 17) and thereby provide an answer to *SQ2.3. How can an agile method be utilized in safety critical software development?* In Section 8, we validate this adapted method in practice.

We have seen that a lot of documentation is required for software development in the safety-critical domain, even when this is performed agile. The requirements from the standards within the medical device sector still needs to be met. We outline necessary documentation in Section 7.1.

Tools are, according to our experts and existing research, essential to achieving a compliant development process for medical device software. Therefore, in Section 7.2, we show commonly used tools and verification methods for medical device software we found during our research.

Roles and responsibilities within organizations are still divided with the mindset of an abundance of resources, as is available in medium- to large-sized organizations in the medical device sector. To adapt this for micro- to small-sized organizations, in Section 7.3 we describe the possibility to combine roles into one, while maintaining the division of concerns.

The process of an agile method is different than the process in a V-model, as we have seen before. For small organizations with limited resources, strict role division and responsibilities of maintaining documentation, we outlined the process after adaptation in Section 7.4.

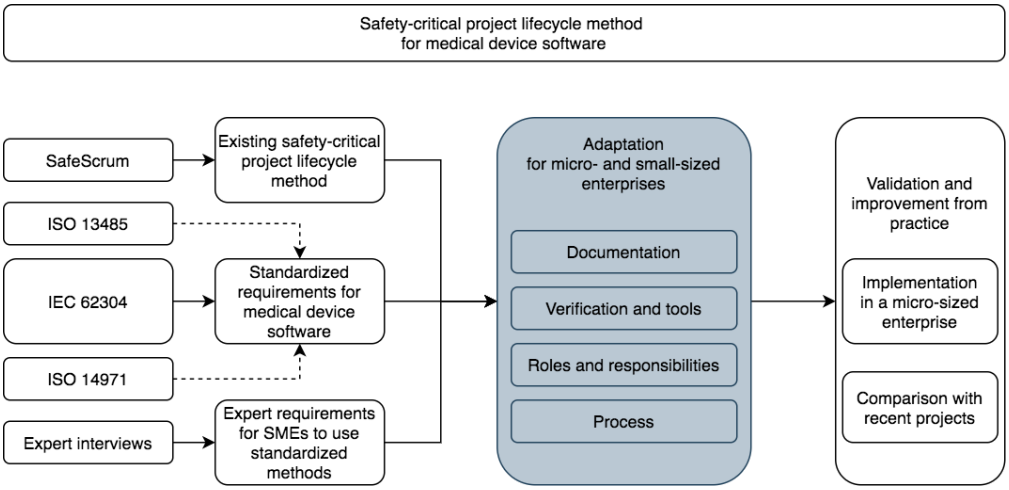

Figure 17: adaptation for micro- and small-sized enterprises

7.1. Documentation

As we have seen before, documentation is an essential part of development in a safety-critical environment (Section 3.1). However, in agile development, practitioners aim to reduce documentation to the minimum required. Documentation should serve a purpose to the final product. In the case of safety-critical software, documentation is necessary for certification of the product. Therefore, it is intertwined with the lifecycle of development. Because micro- and small-sized enterprises have limited resources, we suggest starting with minimum documentation, and append this documentation when necessary.

In Section 6.4, we have covered the documentation within SafeScrum and IEC 62304. The requirements from these sources show overlap in the creation of a requirements specification, development plan, an operation, release and maintenance plan, software validation and a system design. We summarized this overlap in Table 14. Based on the gathered requirements from these sources, we propose to setup three documents during development: a system and software requirements document, a development, safety and maintenance plan and a system design.

System and Software Requirements Specification
In the system and software requirements, we define all requirements that the system under development, and software as part of the system or being the system, should comply to. This can be done in the form of user stories, or by defining requirements. Requirement categories are functional, external interfaces (user, hardware, software and communications), and nonfunctional requirements (performance, safety, security and software quality).

Development, Safety and Maintenance Plan
The development, safety and maintenance plan incorporate several requirements from the agile safety plan within SafeScrum, and software development and maintenance, testing and release requirements from IEC 62304 (NEN, 2006). Within SafeScrum, the agile safety plan documents processes and actions that guarantee safety during development and safety to the final product. This document is extensive, and therefore has overlap with the requirements specification, development and maintenance plan and system design. We leave the requirements and system design parts out, as these are already described in their respective documents.

The development and safety plan describes the process in which the product is developed. In Section 7.4, we describe the process for the product under development, adapted from SafeScrum and implementing the requirements that are specific for the medical device industry.

Every sprint results in a product that should be ready to be released to the public. In order to do this in a structured way, a release must be made. IEC 62304 prescribes that this release is documented properly, as we have seen in Chapter 6.2. The release plan is described in a chapter of this plan.

After release of a product, standards prescribe that the manufacturer of a product has taken appropriate action to monitor, act and improve the product based on feedback from the market. Therefore, the software company needs to have a plan for maintenance of the product. The chapter about maintenance is focused on this process.

Within SafeScrum, the verification and validation plan is considered a project manager's activity, and therefore not a part of the development lifecycle (Hanssen et al., 2018, p. 87). However, verification is mandated by IEC 62304. We included a chapter on verification in the development plan. In this chapter, the verification process and activities are described.

System Design
Within the system design, according to SafeScrum, the system architecture, software system design and main components are outlined. In IEC 62304, the software architecture is worked out in a separate document, but also SOUP needs to be documented. In the system design document, these parts together are documented.

Table 14: documentation overlap and adaptation

SafeScrum	IEC 62304	Adapted
System Requirements Specification	Software Requirements	System and Software Requirements
Agile Safety Plan	Software Development and Maintenance	Development, Safety and Maintenance Plan
Development Plan		
Operation and Maintenance Plan		
Verification Plan	Software Testing	
Release Plan	Software Release	
System Design Architecture Software system design Main components	Libraries / Dependencies Software Architecture	System Design

Hazard and Risk Analysis
In safety-critical software development for medical devices, assessing and mitigating hazards and risks is essential to receive certification. Every sprint, hazards and risks that arise or are foreseen will be assessed and documented in the hazard and risk document, so that during certification, proof can be provided that risks are considered, and action is taken upon them. The hazard and risk document is divided into three parts: hazard identification, risk description and impact, and risk mitigation. The owner, status of the risk and date are registered.

A hazard is anything that can cause harm to a person, which can be a certain situation, a tool or a certain working of software. First, hazards are identified, and which hazardous situations might arise from that hazard. Also, the potential harm is described.

Next, the risk is identified. The risk is a combination of the probability and the severity in case the hazard occurs. The probability (see Table 11) and severity (see Table 10) are noted, just as the risk classification that results from the combination of probability and severity, as we saw in the risk matrix of Table 12.

Lastly, the risk mitigation is described. In order to do so, the event or events that trigger the risk are documented. When a risk is identified, based on the classification, three types of actions can be taken. If the risk is low, the decision can be made to accept the risk in the final product. This can occur if the risk has a low probability or a low severity, or both. Otherwise, a preventive action must be taken. Preventive actions might include describing the risk in the manual, training of usage of the product or describing the indented use more stringent. If the risk is considered unacceptable, such as a high risk with a high probability and a high severity, then the product needs adjustment so this risk cannot occur.

7.2. Tools and Verification of Code

In methods, standards and by experts, tools are recommended to be used (Hanssen et al., 2018, Chapter 8.3). Tools can be used to enforce the process, to automate certain process steps and to standardize documents. In this subchapter, we will discuss tools that are extensively used in the industry and make recommendations for micro- and small-sized enterprises.

Quality Management System
A quality management system (QMS) describes standardized processes and policies within an organization. The usage of such a system in the organization is required by ISO 13485 and can be implemented in several ways. A traditional way of implementing a QMS is by printing documents to hand-sign signatures. An often-heard consideration to stick with this so-called paper QMS is because of acceptance by the FDA in the USA. Under strict rules, the FDA will accept digital signatures as a valid form of electronic record control (Hurd, 2018).

The alternative is an eQMS. An electronic QMS has typical advantages of a digital system over a traditional paper system: it is easier accessible, increases transparency by allowing searching and indexing in a faster pace. Startups in the medical device software sector often aim to use a digital QMS, since they do not have the legacy of a paper QMS, our interviews showed.

Eidel (2020a) made an extensive comparison between eQMS tools and listed requirements for QMS software. Their comparison selects eQMS tools based on five technical requirements, and three qualitative requirements. The technical requirements specify that the tool should allow the creation and formatting, revision history, and a review process for documents, including the tamper-proof signing of documents. Three qualitative requirements are defined. Firstly, the tool should be easy to use without extensive training. Secondly, the tool should be accessible to every employee within the company. As these tools are usually sold on a per-user license, the price is an important factor for this requirement. Lastly, the tool should be opiniated, so the default usage results in compliance, without requiring configuration effort. The findings are summarized in Table 15.

Table 15: list of eQMS for the medical device sector (adapted from Eidel, 2020a)

Name	Type	Interface	Usability	Cost
Confluence	Self-hosted	▢	▢	Low
GitLab / GitHub	Self-hosted / hosted	++	++	Low
Google Docs / Drive	Hosted	++	++	Low
Greenlight Guru	Hosted	+	+	Very high
MasterControl	Hosted	▢	▢	Very high
MatrixQMS	Hosted	-	▢	High
Orcanos	Hosted	--	-	Very high
Polarion	Self-hosted	--	-	High
Qualio	Hosted	▢	▢	Very high

++ excellent, + good, ▢ acceptable, - insufficient, -- bad

In this comparison, the tools that have a low pricing range, offer the best interface and usability. This is important for our selection. Micro- and small-sized enterprises often do not have the time and resources to implement an extensive quality management system, and many already use git for their source control and a Google Drive solution for their email and documents. Both offer a good usability in an easy interface. However, important issues arise with these tools. Since they are not designed to be used as a QMS, traceability, reviewing and signing of documents are not built in. A process should make sure that these steps are adequately implemented into an organization. Due to their ease-of-use and pricing, we recommend the usage of these tools for the quality management system in these enterprises.

Requirements Engineering

Requirements within safety-critical software development are at the basis of traceability. They determine what the product needs to be able to achieve and which safety measures need to be taken in order to keep risk at an acceptable level. Requirements are gathered in a requirements specifications document or in a tool.

Requirements can be managed using tools. As requirements are automatically numbered and these tools allow for linking with other systems, traceability might improve when using them. Not only requirements are managed in these tools, often these tools also allow for test cases to validate the requirements to be managed. For medical device software, a few requirements tools are available. First, the option is to maintain the requirements in a document. This document can fit within the quality management system, with appropriate processes to maintain it. A requirements document is described in Section 7.1.

A tool that several SMEs in the medical device industry use for their requirements is Matrix Requirements.

Table 16: list of requirements tools for the medical device sector

Name	Type	Traceability	Pricing	Notes
Requirements Document	Paper	▢	Low	*Needs manual process for traceability*
Matrix Requirements	Hosted	+	High	
Requirement & Test Management	Self-hosted / hosted	++	Low	*Needs JIRA installation; does not work standalone*

Implementation
During development, the usage of tools can support developers to create high-quality software and prevent them to make mistakes. The toolset that a developer uses for development can differ per project and is dependent on the preference of the enterprise, the programming language and personal preference of a developer (Gasparic et al., 2016).

7.3. Roles and Responsibilities

As we have shown before, in the SafeScrum method, ten roles are defined amongst two teams that will work on the same project (see the leftmost column in Table 17). In their method, Hanssen et al. (2018) note that several roles are optional or can be combined. In SMEs, resources are valuable and scarce. Therefore, in our adapted method, we propose to combine several of these roles, as summarized in Table 17.

We propose to combine the roles of Scrum Master and Quality Assurer, as Hanssen et al. (2018) mention that these roles can be combined in small projects. Alternatively, the role of Quality assurer can be rotated throughout the team, changing every sprint.

In large projects, a project manager is a common role to have, according to Hanssen et al. (2018). In small projects, often a product owner takes the role of a project manager (Cottmeyer, 2009).

SafeScrum does not require the development team members and the alongside engineering team members to be mutually exclusive. To improve integration of these teams and to align their goals and communication, we propose these teams to operate as one. In that way, coordination between these teams becomes an integral part of the team instead of a separate role.

Within SafeScrum, the independent tester is not a part of the scrum team, since some standards or assessors might have objections against teams who test their own code (Hanssen et al., 2018, Chapter 6.3). Testing bias might occur when a developer tests their own code (McBride & Lepmets, 2017). As long as the tester and the developer are a separate person, this problem would be sufficiently mitigated (Heeager & Nielsen, 2018). In practice, practitioners mention no objections from assessors with another team member testing the code from a developer from the same team. Teams in micro- and small-sized companies are limited to sometimes only one team. Therefore, we suggest rotating the role of tester within the team, letting a separate developer test the implemented code.

Table 17: adapted roles for a micro- or small-sized enterprise

<table>
<tr><th>Original Roles</th><th colspan="2">Proposed Roles for Small Teams</th></tr>
<tr><td>Scrum Master</td><td rowspan="2">Scrum Master and Quality Assurer</td><td rowspan="8">Part of Scrum team</td></tr>
<tr><td>Quality Assurer</td></tr>
<tr><td>Product Owner</td><td rowspan="2">Product Owner and Project Manager</td></tr>
<tr><td>Project manager</td></tr>
<tr><td>RAMS engineer</td><td>RAMS engineer</td></tr>
<tr><td>Scrum team</td><td rowspan="4">Scrum + Alongside engineering team + Coordination + Testers</td></tr>
<tr><td>Alongside engineering team</td></tr>
<tr><td>Coordination</td></tr>
<tr><td>Testers</td></tr>
<tr><td>Independent Safety Assessor</td><td colspan="2">Independent Safety Assessor</td></tr>
</table>

7.4. Process

The development process in safety-critical software is stricter than in regular software development. Regulations prescribe verification of implementations, to guarantee safety and to prevent accidents. These aspects are embedded in the V-model, but can be achieved in agile development as well, as SafeScrum shows (Hanssen et al., 2018). Next to that, all our practitioners use a form of agile development for their medical device software (see Section 6.3).

We base our process on the SafeScrum process elements as shown in Figure 16. We use iterations which are called sprints to develop the product. The development cycle is split into three parts: sprint planning, sprint execution and sprint retrospective and review. During these phases, hazard identification and quality assurance are embedded parts of the process.

Before starting sprint cycles, we need a baseline of understanding between customer and developer. To achieve this, requirements are gathered in the system and software requirements specification, as described in Section 7.1. This will be the starting point for the product owner to define the initial product backlog. Next to that, the initial documentation is setup by the product owner, the scrum master, and the RAMS engineer.

Sprint Planning
When starting development, the sprint is started with a meeting to plan the development in this iteration. This sprint planning can take up about five percent of the time for the full sprint. In case of a four-week sprint, this would be eight hours. Within this meeting, three aspects are discussed.

Firstly, the sprint goal is determined by the product owner and the team. The sprint goal describes the objective of the sprint in a specific and measurable way.

Next, the team is made explicit, and the commitment of the team is discussed. Ideally, a team consists of the same members as the predictability and experience of their capabilities increases over time, according to Hanssen et al. (2018). Based on the team members and their commitment, the available time within the sprint can be determined. The hours that team members work are added together, the meetings that scrum requires can be subtracted from that amount. This results in the sprint capacity. According to the scrum method, teams start by planning about 70 percent of their available time, keeping 30 percent available for resolving issues, administration, other meetings, or calls.

The third step is to create a sprint backlog. Based on the prioritization of the product backlog created by the product owner, the top priority stories that fit the sprint goal are selected by the development team. The commitment of time of the team, as determined in the previous step, is used to select the volume of stories that can be completed in this sprint.

Sprint Execution
During the execution of the sprint, every developer works on the tasks that they are assigned during the sprint planning. In order to safeguard the quality of the code and compliance to regulations, some additional steps are added in the process. An overview can be seen in Figure 18.

First, the developer starts to work on a story. They mark the story in as “in progress” to show traceability in the progress and they create a new branch in the git repository. This is a temporary diversion from the main code. It allows for simultaneous development of several features as each developer works in their own branch, while not affecting the main code base until the feature is finished and quality checked.

As found by several authors, test-driven development is becoming the standard in safety-critical software development, proving to be a more rigorous method that forces developers to think ahead of their code (Heeager & Nielsen, 2018; Kasauli et al., 2018; Stålhane et al., 2012). Therefore, we propose to start development by the creation of test cases for the code to be written. To make sure

that the test cases work, they are ran once and will return that they fail, because the code to make them succeed has yet to be implemented.

After this, the actual implementation takes place. The developer creates the code that they wrote tests for and runs the test cases intermediately or at the end. A developer adds their code into the branch via a “commit” action. If all the test cases pass, they can create a pull request. A pull request is done to add the newly created code into the main code base. In the pull request, they explain how the code is implemented and how a tester can check the functionality. The test results from the test cases are added to the pull request, providing traceability throughout the process. Lastly, they assign a code reviewer for their code, which is a co-developer who has not directly been involved in the implementation of the code.

Once the code reviewer receives the pull request, they go into a process of reviewing the code on quality and giving feedback to the developer. They download the specific state of the code that the developer created onto their computer in an action that is called a “git checkout”. The code reviewer then runs the tests to check whether they also pass on their computer. If that is the case, they dive deeper into the code, to ensure that the code structure is logical, standards have been followed and edge cases have been prevented. Usually, a code reviewer will have some remarks at either the test step or the code review step, which they attach to the pull request. The developer has to resolve these issues and submit a new review request. When the code is up to quality and the code reviewer is satisfied, they approve the code review.

A last check is done by the quality assurer, who checks the metrics of the code such as code review comments, code complexity, test coverage, documentation coverage, and traceability from requirements to code. If the metrics are acceptable, the story is accepted, the documentation for traceability is exported and the developer is notified that they can merge the code into the main code base. The quality assurer might find an issue with the metrics. In that case, and when the risk is low, the developer improves their code, and the code reviewer will review the updated code again. However, if the risk is medium or high, a quality issue is raised by the quality assurer and the story is added to the agenda for the sprint review.

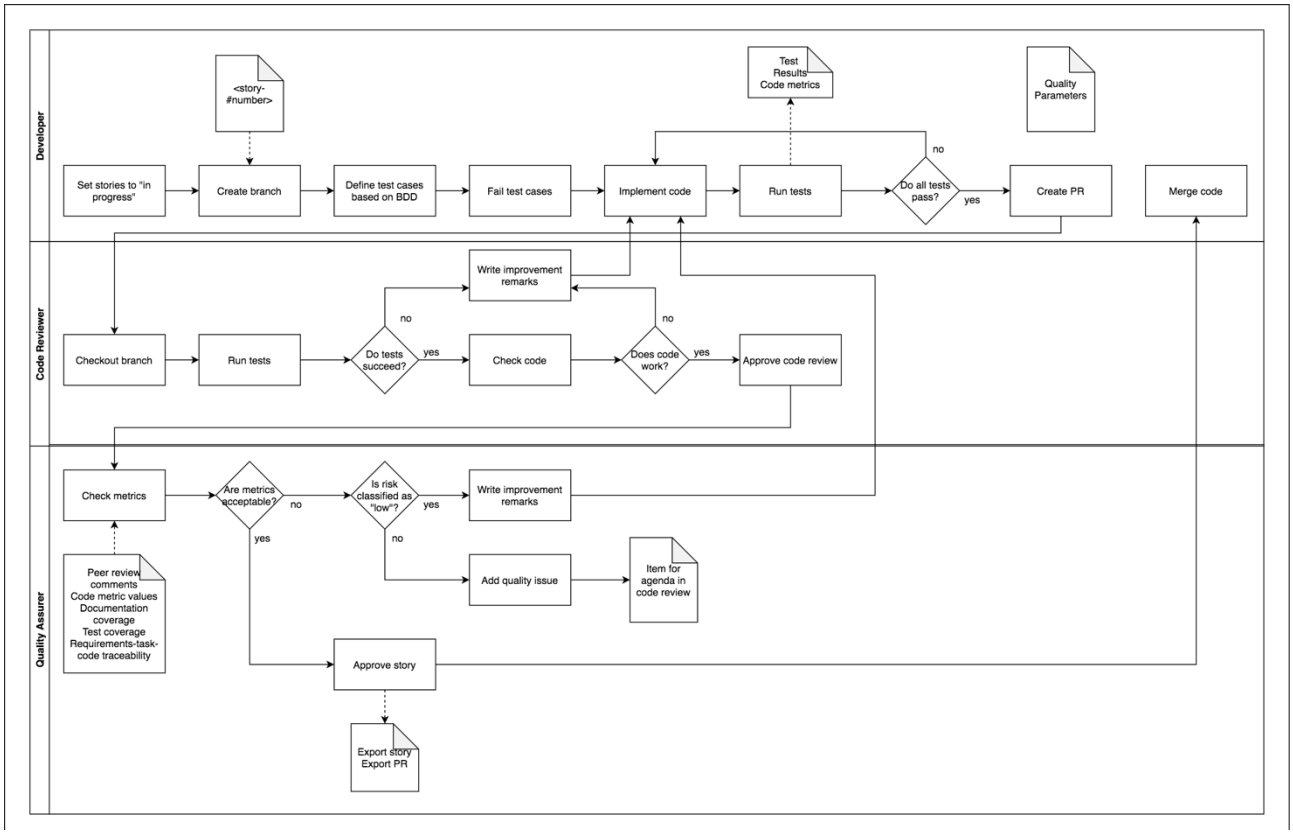

Figure 18: overview of activities per role in the adapted method

Daily standup
Since the method is based on scrum, in the same fashion as scrum, daily within a team a standup meeting is organized at the start of the day. This standup lasts about 15 minutes and in this meeting every developer mentions what they worked on the day before, what they will be working on today and any problems they face while working. Other developers can comment on these problems and ask questions to get a better understanding. However, if in-depth discussion is required, a moment outside of the daily standup should be planned to keep the standup meeting short and concise. This step increases the transparency within the team, by keeping each other updated, and helps to resolve potential impediments of the project right away as they are discussed early on detection.

Sprint review
At the end of the sprint, a sprint review takes place. During the sprint review, the results of this sprint are presented to the stakeholders in the project. These stakeholders can be present themselves or are represented by the product owner. A sprint review usually takes about 2,5% of time of the total sprint. For a four-week sprint time, this accounts to six hours.

In the second part of the meeting, unresolved stories are discussed. First, stories that have open quality issues are addressed. Per story the risk is assessed. For quality issues, during the meeting a decision is made whether the deviation from standards is acceptable, and this decision is documented. In the case it is acceptable, the story is accepted. One might argue this is the most important part of the meeting in a safety-critical environment, since it considers the safety aspects of the software.

The first part of this meeting is used to show the deliveries of this sprint. This can be in the form of a presentation; in the case of software, often new functionalities are demonstrated. Once the product owner accepts the functionalities, their respective stories can be closed and finished. If any issues are found or the story is not resolved to the satisfaction of the product owner, comments are added to the story. These comments should make it clear for the developer on how to solve the story in a next iteration, for example by updating the definition of done.

Sprint Retrospective
In-between sprints, a sprint retrospective is organized with the team. The goal of the sprint retrospective is to streamline the sprint process by discussing and resolving any impediments encountered. In this way, the quality and effectiveness can be increased. Topics discussed are personal work, interactions, processes, tools and definitions of done (Schwaber & Sutherland, 2020). The team discusses which parts of the sprint went well, which parts need improvement and how to achieve that improvement. A sprint retrospective takes up about 1,875% of the total sprint time. For a four-week sprint, this is 4,5 hours.

Phase / meeting	Purpose	Time
Hazard identification	Discover potential harm and risks	Throughout the sprints
Sprint planning	Planning the upcoming weeks of work.	5% of sprint duration
Sprint execution	Resolving stories to create deliverables.	~70% of sprint duration
Daily standup	Team update on progress, where each individual mentions their impediments.	15 minutes per sprint day
Sprint review	Discussion and showcase of deliveries in the past sprint to stakeholders.	2,5% of sprint duration
Sprint retrospective	Discussion about process of past sprint to improve effectiveness and quality.	1,875% of sprint duration

8. Development of Software as a Medical Device

Our method is created and validated in three parts: using experts, an existing method and standards and regulations to obtain requirements and validation as described in Section 6, adaptation from these requirements in Section 7, and by performing action research within a company setting. These results are then compared by other agile methods recently used. The last part of this process is shown in Figure 19. In this chapter, we answer *SQ3.1. How does the method perform in practice?* and *SQ3.2. How does the method compare to other methods?*.

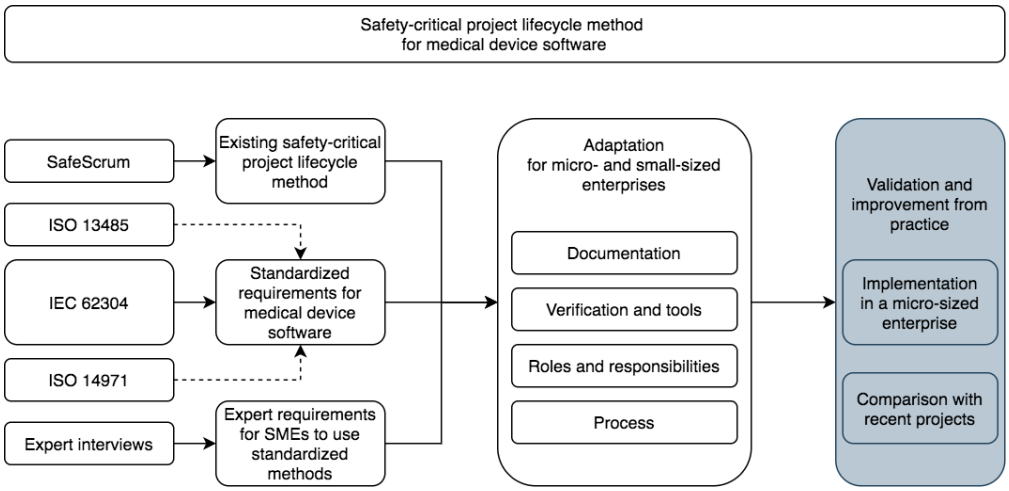

Figure 19: validation and improvement of the adapted method

8.1. Background on the Implementation

The company
The method is implemented in the company *Fris & Fruitig*, a micro-sized software company creating software for healthcare, that wants to transition to creating software for medical devices. Currently, they do not have a standardized process to develop software. The process can thus be described as the “Lack of a Software Development Method” from Section 3.1, while the company tries to work agile as much as possible. This works, because the company consists of four employees, projects are relatively small, and communication is direct. However, with the requirements that developing software for medical devices bring, the desire and need exist to transition to a more structured approach.

Project ERAS
To develop and improve our method, our method is implemented within the company in a new project. The project comprised of the development of an application that will be used in hospitals by patients to aid their recovery after surgery. Three developers were involved in the project. One developer took the role as scrum master, one as product owner, and one as RAMS engineer.

Specifically, this project aims to support the surgery of esophageal cancer. Historically, esophageal cancer surgery is a complex and major surgery with high morbidity rates (59%) and a 90-day mortality rate of 13%. In order to decrease these figures and increase success of this surgery, the enhanced recovery after surgery (ERAS) protocol was introduced in 1997. This approach includes all medical disciplines involved in the surgery: surgeons, physicians, nutritionists, and anaesthesiologists, while also activating the patient. The approach has proven to decrease morbidity, reduce surgical stress response, shorten the length of hospital stay, and expedite recovery. The implementation of ERAS protocols has decreased the costs of treatment, without compromising outcomes (Ashok et al., 2020).

A Dutch hospital has effectively been able to implement ERAS protocols inside of their organization. This includes all professionals involved. A key component in the effectiveness of ERAS protocols is including the patient. The patient should adhere to a certain diet before and after the surgery, and should perform certain exercises before, during the hospital stay and after the surgery. Implementing this part has proven challenging for the hospital. Patients are guided and monitored on a regular schedule (monthly before the surgery, weekly after surgery). However, the hospital wants to make it easier for patients and caregivers to adhere to the schedule of activities and monitor the patient intermediately.

Currently, most of the activities and diets are shared and performed by specialists and nurses during meetings with patients, which is time-intensive for these caregivers. Many of these tasks are simple and could be performed without help. Examples include sitting on the bedside for an amount of seconds, drinking a few sips of water, or walking a few steps. However, due to the frequency of the exercises, it can be difficult for patients to remember to perform them. Next to that, patients might feel insecure if they are performing the activities well.

A mobile application is suggested to help with instructing patients, listing their activities, and reminding them when the activities need to be performed. In this way, patients can be sure that they perform the activities in the right way and at the right times, giving the caregivers time to focus on their primary tasks.

Tools

The company uses Google Drive for their documentation and administration and GitHub for their source control. Since the company is used to these tools and our research shows these tools can be appropriate for certified software, we adapted these tools within the project. We set up the documents in Google Docs, giving the advantage that revision history is automatically stored. A signing section is added to the main page.

Within GitHub, an issue tracker is included into the source control. The user stories that arise from the requirements are stored as issues in GitHub. We created a template for user stories in which the product owner and scrum master make sure that traceability to requirements is preserved (Appendix G). Due to the workings of GitHub, if a code commit is made with the issue number in the message, it will automatically be linked to the user story. In this way, a developer also makes sure that traceability from user story to code is met.

Process

When a new user story is picked up, the developer creates a new git branch from the main branch and commits and pushes their branch to GitHub. In that way, the development is separated from the latest stable version. The developer creates their code in this branch. In order to comply to agreed quality standards, GitHub Actions are implemented and automatically ran when a commit is made. This automation checks whether the code can be compiled and has the right formatting and complexity. If the automation finds a quality issue, this is added to the GitHub issue, allowing the developer to fix it before the code review. This saves time and effort for the developer as well as the code reviewer.

When the developer deems the development finished, according to the standards of our method, they open a pull request in the GitHub repository. They select the main branch and the branch that they are working on, and GitHub automatically checks if they can be merged.

Three employees in the company are involved in the project. These employees all have experience working with the programming language and tools we outline for the project. One of the employees is also the researcher of this paper. This employee takes the role of scrum master in our method, to guide the correct usage of the method and to detect and document impediments. One other employee who has direct contact with the customer takes on the role as product owner, leaving the role of RAMS engineer for the last employee. This division of roles is in line with our action design research method (Chapter 2.3); active involvement from researchers and practitioners is proposed to design the artifact.

8.2. Preparation

Before starting the project, the documents that are required for certification had to be prepared. In this subchapter, we outline the specific steps that have been taken to gather and document the necessary information.

System and Software Requirements Specification
During the initial phase of the project, interviews were conducted by the product owner with stakeholders within the project. Three employees who will be users of the system were asked about their needs and wishes for the new application. Based on these interviews, the product owner formalized requirements. These were then checked with the interviewees, making sure that the requirements align with their needs.

In our system and software requirements specification, we start with an outline of the full system. The Product Perspective gives a summary of the purpose of the application under development. Our mobile application under development supports quick medical rehabilitation by fast mobilisation. To be able to achieve that, patients are presented several easy-to-perform activities in the application, which they can perform before, during their hospital stay and after their surgery. Examples of these are sitting on the bedside for a few amount of seconds, drinking a few sips of water or walking a few steps, depending on the intensity of the surgery. These activities will increase in amount every day the surgery has passed.

Next to that, the Product Features paragraph subchapter in detail which functionality is available in the application. In our application development, we focused on creating a proof of concept, covering the minimum feature requirements. Activities are shown as tasks to the patient. They need to perform these tasks. Instructions on how to perform the tasks are shown to the patient by text and video. The application includes a list of these tasks, to give an overview of tasks that a patient has to perform in a day. Also, to cover the case of patients forgetting the routine of activities, reminders are included in the application. Lastly, a patient, but also the caregiver, can see the progress of activities in the application.

The Operating Environment of the application is placed inside of a hospital, since in the initial case, patients will get this application after surgery. The application will be pre-installed in an Android 8.0 tablet with a screen size of 10 inch. A progressive web application is developed to allow rapid prototyping. In order to store data, the local storage is used within the browser of the application. The setup of the tools is common for the developing company: TypeScript with React, HTML and CSS.

After defining the system and functionality, the future features are formalized into software requirements. This is necessary to guarantee traceability throughout the development process. We categorized our requirements using the categories described in Section 7.1. To maintain traceability, we number the requirements. These numbers are later used during the user story creation to show the connection between user stories and requirements. The requirements we extracted from the interviews are listed in Appendix E.

Development, Safety and Maintenance Plan
As described in Chapter 7.1, the processes are to be documented. We described our processes in Chapter 7.4.

In this document the project is described from an organizational perspective. Therefore, the project purpose, scope and objectives are included. The project aims to deliver a Minimum Viable Product application, that can be used to verify the added value and functionality in the ERAS protocol. The application is aimed at patients who are recovering from surgery under ERAS-lead protocol. Deliverables are defined are a demonstratable version of the application after two sprints. Also, the project organization is described. As mentioned in Chapter 8.1, three developers work on the project, in the roles of scrum master, product owner, and RAMS engineer.

Hazard and Risk Assessment
Risk assessment is a fundamental part of safety-critical software development, described in ISO 14971. In order to comply to regulations, risks need to be managed. Our risk assessment was kept thorough enough for certification, and at the same time not intrusive for development. In a document, we described potential hazards and potential risks, classifying them in a risk category. From the initial assessment, no unacceptable risks were found. Later in the development project, no unacceptable risks were assessed either. Due to the nature of the project, potential hazards were kept in check.

However, in order to practice the method, we added risk mitigation to several improbable risks we identified. Patients might hurt themselves while performing the activities the application prescribes them. Therefore, during the testing phase of the project, it is important that a caregiver is present when the patient performs an activity. For the inner workings of the application, we guarantee data integrity by saving data with a checksum, thereby mitigating the risk of showing wrong or corrupt data.

Classification
As part of the certification, the software needs to be classified as a medical device and categorized into a risk class. The intended use of the application is to guide patients in their recovery. Therefore, according to the MDR, the application is seen as a medical device in terms of monitoring and treatment of disease or disability (European Union, 2017). A flow diagram in IEC 62304 provides guidance in classifying the software in risk classes (Figure 16). Class A is seen as to incur no injury or damage to health when the medical device is used. Class B medical devices can incur no serious injury, while class C devices might incur serious injury or death when the device is not used properly or malfunctions.

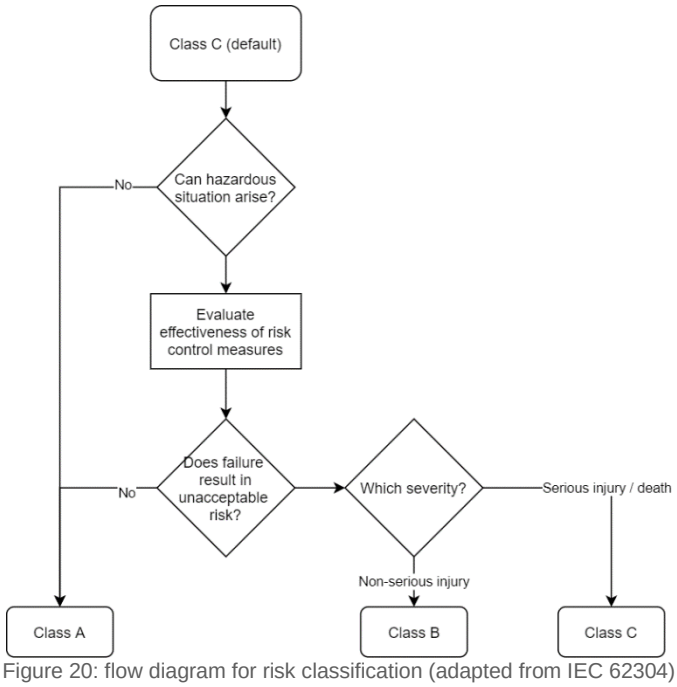

Figure 20: flow diagram for risk classification (adapted from IEC 62304)

In the MDR, these risk categories are refined. According to the MDR, software is seen as an active device. According to the clasification rules, if the software is independent of any other device, it is classified in its own right (European Union, 2017). Rule 10a for classification specifically mentions software. This rule aligns the definition of risk classes in the MDR with the risk classes in IEC 62304, as shown in Table 18 (Michaud, 2016).

Table 18: comparison of risk classes between IEC 62304 and MDR

Highest hazard	IEC 62304	MDR
No injury	A	I
Non-serious injury	B	IIa
Serious injury	C	IIb
Death	C	III

Applying this to our project, we start with the flow diagram. As we have seen during the hazard and risk assessment, hazardous situations can arise from failure of the software. In order to prevent this, risk control measures have been built in to the application. Therefore, no unacceptable risk is left when the software malfunctions. Next to that, specialists mention that performing the activities we outline in the application help reduce (improve) mobility time, but if they are not performed, do not cause harm. We conclude that following IEC 62304, our application falls into class A. If we convert this to MDR classification, it results in class I.

8.3. Execution of the Method

In the previous subchapters, we have implemented and prepared the adapted method to be used for a safety-critical software development project in a micros-sized enterprise. In this subchapter, we outline the process and proceeding findings. We conducted the project preparation and two sprints within the organization (Figure 21). The timeline of events is shown in Table 19.

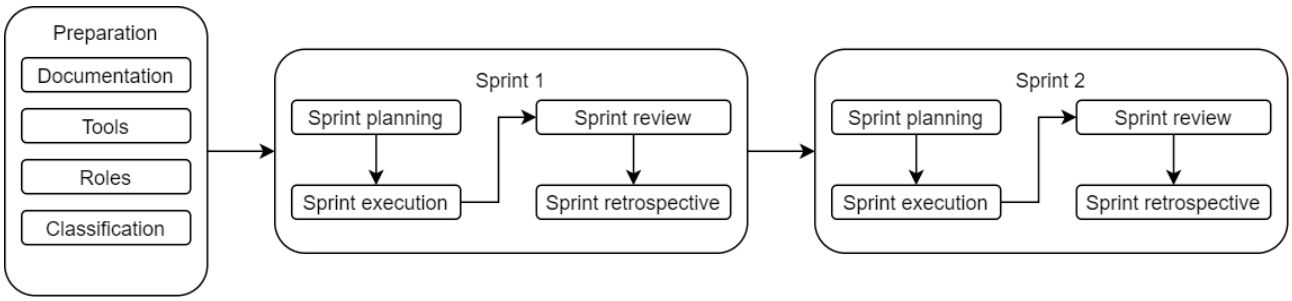

Figure 21: process of our research's findings

Table 19: timeline of events

Activity	Date
Preparation: Requirements and Hazard Analysis	03-08-2020
Sprint Planning #1	10-08-2020
Development / Standup	17-08-2020
Development / Standup	24-08-2020
Sprint Review	02-09-2020
Sprint Retrospective	02-09-2020
Sprint Planning #2	02-09-2020
Development / Standup	10-09-2020
Development / Standup	16-09-2020
Sprint Review #2	23-09-2020
Sprint Retrospective #2	23-09-2020

Preparation
During preparation, we focused on readying the team for development. The documentation is filled as far as possible by the product owner, the scrum master and the RAMS engineer, classification is determined, requirements are written down by the product owner and risks are assessed by the team. For a more detailed description, see Section 8.2. At the start of the project, the responsibilities

per role are unclear. Therefore, these responsibilities are written down at the beginning of the project. See Appendix F for the description of responsibilities.

The company is new to using a structured agile safety-critical methods, so unknown situations were to be expected. Since development was for an application that runs on a mobile device or tablet, the question arose whether this tablet also has to be certified for usage in a safety-critical environment – the hospital. This depends on the intended use of the medical device. Since the tablet is enabling the application under development to be used, but malfunctioning of the tablet would not result in unacceptable harm, the tablet does not have to be certified (Eidel, 2021).

Another point of discussion was the used architecture during development. Software reuse is common practice in the software development practice. Frameworks, such as React for JavaScript, and libraries are often used in order to save time and effort for the developer. These reusable software packages are created and maintained by an external developer or a team. However, since these packages are externally maintained, the safety of these packages cannot be guaranteed by testing. Research has shown that actively maintained packages are used in safety-critical software in the automotive, aerospace, medical and nuclear industries (Sulaman et al., 2014). Next to that, from our interviews in Section 6.3, we learned that frameworks and libraries can be used, as long as they are risk-assessed or tested for safety. End-to-end testing of the application with sufficient test cases could show safety. In our setup, we decided to use React and supporting libraries, since no unacceptable risk was identified by usage of reusable software.

The quality attributes specified have an impact on development and safety, since each quality attribute has to be checked and verified before development can continue. A trade-off must be made here between safety and making progress. We use the tool ESLint, a static analysis tool that helps developers to find and fix problems in JavaScript. Prettier makes code easier to read but does not change the code automatically, thereby not changing any possible side-effects.

In order to measure safety, our quality attributes were specified as follows. We defined acceptable code coverage of testing at 100%, as is common in the safety-critical industry. The test cases should produce no errors. We used code complexity as a measure of quality; this measure counts the possible execution paths through an application or code function. We kept this complexity score below 10.

Sprint 1: creation of architecture and development environment
In the first sprint, focus was put on setting up the architecture of the software and the development environment. A list of tools that change the code was created, in order to estimate the risk that they pose. TypeScript and Babel are code compilers and thereby potentially change the functionality of the code, while ESLint is a static analysis tool that only gives suggestions to the developer based on its analysis. Prettier is a tool that automatically formats the code, thereby making it easier to read for a developer, but keeping functionality the same.

During setup, the configuration of tools is discussed. ESLint checks code statically, and thereby has rules that can be configured to allow for strict, loose or no checking at all. These rules are based on best-practice, safety, and coding standards. Therefore, the decision is made to use the default settings for these rules. ESLint also produces a report, adding to the documentation for certification.

Two paths are taken in setting up the architecture: using a framework and starting empty. Starting new has the added advantage that control over code is guaranteed. It can be fully documented, checked by developers and tests can be written for it. However, starting new is found to be time-consuming, because configuration, maintenance and tests must be specifically designed for it. A trade-off between control and development effectiveness has to be made. Since frameworks are increasingly used within safety-critical software development and no unacceptable risks are identified, we continued using React.

Supporting libraries in JavaScript are called packages. In IEC 62304, these packages are called Software Of Unknown Provenance (SOUP). Research often describes them as reusable software. The packages are documented in our software development plan, as suggested by Eidel (2020a).

In the process of development, the code review stalled at the opening of a pull request within GitHub. Because multiple code reviewers were assigned or no code reviewer was assigned, these code reviews were left until the end of development or forgotten altogether. During the sprint retrospective, this point was discussed. The solution came by assigning one code reviewer on opening a pull request and informing that person when the pull request was open. In that way, responsibilities are clear and the code reviews should become an integral part of the development process.

To achieve a test-first approach, we compared testing frameworks and chose jest as our JavaScript testing framework. Jest allows writing tests in a test-driven or behavior-driven approach. Also, tests can be run automatically by using GitHub Actions after commit. Results are added to a test report and serve as documentation to compliance.

During quality review, we made sure that the quality attributes were met. This is done by the report that ESLint produced. Test verification was checked through the report that jest provides. The comments of code reviews had to be incorporated into the work. Lastly, traceability was assured, checking that every addition of code could be traced back to requirements from the documentation.

The results of sprint 1 can be seen in Appendix C1 Sprint Review and C2 Sprint Retrospective.

Sprint 2: functionalities
Attention went into making sure that the software was developed according to safety-critical standards. The developers automated the quality assurance step by running ESLint after each time a developer pushes a change of code to GitHub. In this way, a static code analysis is done every time a developer changes code. More parts of the process, such as making sure that traceability is transparent, the right references are made and the documentation is filled, can be automated in a similar fashion. This saves time for the developers, while making sure that the quality assurance role is fulfilled.

During retrospective, the question arises whether the code reviewer and quality assurance can be combined, since most of the quality assurance is automated (Appendix C2). The code reviewer would, during code review, also review the quality assurance report that is generated, to make sure that all of the requirements are met. In case an adjustment has to be made found during quality assurance, the code reviewer mentions this to the developer together with remarks from the code review.

The results of sprint 2 can be seen in Appendix D1 Sprint Review and D2 Sprint Retrospective.

8.4. Findings

In the previous subchapter we have shown how the method was executed and discussed findings per step. In this subchapter, we zoom out and discuss general findings per category of research in safety-critical development: development process, documentation, and tools.

Development Process
We found that for this micro-sized enterprise with a small project, it was difficult to start creating safety-critical software totally from scratch. Since a quality management system, risk management system and official procedures were not present yet, there was quite a lot of focus to be placed on defining structures and creating a repeatable process, climbing the stairs of the Capability Maturity Model Integration. There are several factors to consider. Developers get responsibilities in a defined role, writing and maintaining documentation, following process steps, and using the output of tools.

Operationalization led to new insights. While on a high level the method is engineered, practical implementation is rarely discussed in research. Therefore, it was unknown whether it is acceptable

to use SOUP within safety-critical software development. Other topics that are left untouched, but are found during execution of the method, are to our knowledge acceptable programming languages and their compilers, or showing compliance of the process. During execution a community of medical device software developers was approached, who exchange knowledge on software certification for the MDR. These communities seem of value for micro- to small-sized enterprises, because they share information that is not publicly available.

The adapted method we created provides a basis for safety-critical software development in this project and organization. Since the process is similar to a regular scrum method, the developers adapt quite easily to the method, adding some steps to guarantee safety and update documentation.

We noticed that filling all responsibilities in a team of three was difficult, if not impossible. There are four roles in the team that are vital to the success of the method. Development is hindered by the responsibilities that these roles bring. In a team of five these roles could be fulfilled, while there would still be developers fully focused on development. Our practitioners mentioned the focus on regulatory approval takes about two full-time employees in a team of five. The roles of scrum master, product owner, and RAMS engineer, could thereby be taken up by three developers, leaving two developers to focus on the implementation.

Our quality review as an additional step in the development process proved important to show compliance to regulations. As an extra validation step where quality parameters, documentation, traceability and testing were checked, this part of the process shows the safety of the software.

Documentation
For documentation the wiki functionality of GitHub, together with Google Docs are used. These two options should comply to regulations, as they provide traceability of owners and changers of documents, which could even be verified with digital signatures. However, as our practitioners have mentioned, it is up to the notified body to decide whether digital signatures are acceptable. In the USA, digital signatures are not accepted.

Documentation can be compressed into several files, instead of having lengthy monoliths to show compliance. These documents are setup and maintained by the developers. Because of this, the documents become an integral part of the development process, and cross-fertilization between updating documents and writing code is achieved. The amount of documentation is manageable for a small team. Keeping the documents updated however, requires a conscious process step during every sprint.

Tools
Tools are an important aid for developers, as we have seen during our development. In safety-critical software development, proof of safety can be shown by making sure that the software works as expected. In software development a common practice is to write tests to automate this. Given that tests are required to show compliance, we have worked with a testing kit in order to write and execute these tests. With a test-driven development approach, developers implemented tests first, let them fail, and then implemented code. This approach was new to the developers, but easy to get used to and implement in the development process. The result of writing tests first and then implementing the code was that the developed code worked faster, while in the meantime a test report could be generated proving the code was tested as well.

External libraries and frameworks were seen as an indispensable tool for software creation. We developed using two approaches: one starting blank, and one using a framework and using libraries. The blank method gave us the opportunity to have the biggest control over our code, tests and thereby our compliance. However, development in a small company is hindered by having to substitute for certain functionalities, often available in well-maintained libraries. Arguably, it is safer for a micro-sized organization to use external libraries and SOUP that are maintained by a group of external developers, because these libraries are used in a lot of other (production) applications. Since there is a bigger base of usage of these libraries, chances are larger that faulty code or bugs

are found and fixed immediately. Next to that, we found that the usage of SOUP is acceptable, as long as a list is maintained of used libraries. This is the option used in our research.

Quality metrics should be measured and maintained to measure the reliability of the software, the regulation prescribes. One of these quality metrics is the test reports. Next to that, we defined quality measures in our requirements. The code that we wrote was assessed with a complexity score, which we kept within boundaries to improve readability and maintainability. Next to that, the test coverage of code was measured and kept at 100 percent of the developed code.

8.5. Comparison to the Creation of the COVID-19 apps

In the timeframe our research was conducted, a global pandemic of COVID-19 emerged. Interesting to our research was one of the approaches in getting this outbreak under control. Several governments, including the Dutch government, created mobile phone applications that could tell users whether they had been in close contact to people who had been identified to have contracted the virus. By notifying these people, they could get themselves tested. Governments viewed this as an effective method to identify and contain the virus outbreak.

While the functionality of these application differs per country, they are always developed by a private organization in collaboration with the government of the country (Blasimme et al., 2021). Only in the UK and in Switzerland, the applications are registered as Class I medical devices. All other organizations consider their application not to be a medical device; for example, in the Netherlands, the reasoning is that “the application does not contribute to diagnosing a disease” (van der Voort & Kroeks-de Raaij, 2020). However, diagnosis is not necessary to be considered a medical device. In the definition of the MDR, applications that contribute to the prevention or monitoring of diseases. Since the application decides a risk of infection based on proximity to another infected person, this can definitely be considered a medical device.

Since regulations for medical devices are not taken into account in many of these development trajectories, this might prove difficult in a later stage. Some applications are extended to screen users for symptoms and assess their disease. Most definitely, these applications fall into higher classifications of the MDR.

We compare our adapted method to the Dutch CoronaMelder app, which is developed by a freelance team for the Dutch government and which is not certified, and the English NHS COVID-19 app, developed by an external party and certified class I medical device. A summary is shown in Table 20.

Projects
As we have seen before, the adapted method was tested in a team of three in a micro-sized organization. We used an agile method to develop an application that aids patients to accelerate their recovery from surgery. The application was classified as risk class I and was developed using a simple quality management and risk management system.

The Dutch medical device is not considered a medical device. This is remarkable, since the objective of the application is to prevent and monitor the spreading of disease. Patients are mentioned in the description of the application (Jansch et al., 2020). The application development costed about 5 million euros and had 50 people involved in the development (Borgerink, 2020). Development took place in an agile way. No quality management system was used during development, and no risk management took place.

The English application was developed by a team of 70. The company that was hired, had already have experience to develop the Swiss COVID-19 tracing application. Next to that, the enterprise is familiar with developing medical device applications, as shown in their portfolio (Zühlke, 2021). Not surprisingly, both of these applications are certified COVID-19 applications. The English COVID-19 application has the capability to, based on symptoms people enter, estimate the chance that

someone is infected. This functionality should bring it to a higher classification level, looking at the MDR. The method that the enterprise used is a waterfall method, implementing a rigid quality and risk management during their project.

Comparison

While the project we ran was conducted with three people, the projects we compare it to had considerable larger teams. They would be placed into a medium-sized enterprise. As we have seen before, medium-sized enterprises have more financial capability and resources to develop safety-critical software. Therefore, it is remarkable that during development of the Dutch application, no safety-critical software development practices were used. Given the characteristics of the application and the team, this would have been possible. Next to that, the Dutch team could have learned from the (earlier developed) Swiss COVID-19 application, which does have a classification as software as a medical device.

The purpose of the application defines whether the application should be considered a medical device or not. With an intended use of “preventing and monitoring of spreading”, the CoronaMelder app would fall in (at least) risk class I of the MDR. The application of the NHS is currently classified at risk class I, while it allows for detecting symptoms and making a risk assessment based on the symptoms. This would, under the MDR, increase the risk classification to IIa.

Our project used a simple quality and risk management system, since it ran in a micro-sized enterprise. We can see to our knowledge, risk management and quality management were not present during development of the CoronaMelder application. While security impact analyses and requirements are defined, no information is available on risk assessment or a standardized process. The NHS application however has been developed with classification in mind. The developer has experience with developing these applications in a waterfall-based approach. Therefore, a rigid process was used.

The costs of the projects differ significantly. We did not consider costs for our ERAS project, so we could not compare that to the other applications. Publications on CoronaMelder and the NHS COVID-19 application however, show that the cost for development and maintenance of the applications were five million euros and 35 million British pounds respectively. We cannot determine for sure whether this is due to the difference in development process, the usage of a quality and risk management system or the in- or exclusion of the classification.

Table 20: comparison between three apps and their development method

	Adapted method project ERAS	**CoronaMelder NL**	**NHS COVID-19 EN**
Team	3	50	70
Cost		€ 5 million	£ 35 million
Process	Agile	Agile	Waterfall
Purpose	Aid to accelerate recovery	Prevent and monitor spreading	Detect symptoms
Classification	I	N/A (I)	I (IIa)
Quality management	+	N/A	++
Risk management	+	N/A	++

9. Discussion

In our research, through a literature review, interviewing practitioners, adapting a method and implementing it in a micro-sized enterprise, we aimed to answer our research question. This chapter combines the results from the individual subquestions and gives an answer to the main research question.

SQ1. What is the state of art in safety-critical software development within SMEs for medical devices?

In recent years, the shift from traditional methods to agile methods has increased in the medical device industry. Just as in other safety-critical industries, software is becoming an increasingly important part of the products that are being developed. Software makes it possible to extract data from the devices and opens the opportunity to integrate products with each other.

Software can relatively easy be updated. With new insights, the safety of a product can be increased by changes in the software, without having to recall the product to the manufacturer. However, this might lead to the impression that mistakes in the initial process can be solved later. The significance for a safety-guaranteed process for development is self-evident, for the product under development and after-market maintenance.

For SMEs, having a rigid software development process is more challenging than for large enterprises. Traditional methods provide the certainty of certification, but require overhead in terms of resources, documentation and testing unforeseen by SMEs. Large enterprises shift to agile methods because it improves transparency in the process, enables customer involvement and increases satisfaction, resulting in a product better aligned with the market needs and of higher quality. SMEs can also benefit from agile, lightweight methods which require less overhead, provide a greater flexibility and promote the standardization of the process. This, in turn, could ease the way for certification. Certification could, with the right adjustments in procedures and legislation, be conducted iteratively. Iterative certification is easier implementable for SMEs in their agile process.

With the problems and proposed solutions assessed, it seems that ample research has been conducted into developing software agile in the safety-critical environment. Three out of four main problem areas have solution candidates ready, while the lifecycle of development seems to be a problem that is underexposed and poses itself as a direction to research.

Careful attempts to formalize research into adapted agile methods have been proposed, such as SafeScrum by Hanssen et al. (2018). This method aims to provide an implementable method to develop software in the safety-critical industry building on the IEC 61508 standard, for automotive, railway and other transportation. A successful adaptation of this method is made by research of Myklebust, Stalhane, et al. (2016) for the petrochemical industry, proving it is possible to change this method to fit in a similar safety-critical industry's context. Hanssen et al. (2018) provide adaptations for the aviation industry and the railway industry as examples that the method can be adapted without any serious problems.

Safety-critical industries include the medical device industry, aviation, space and defense, automotive, railway, robotics and nuclear industries. There are a lot of similarities in these industries because the aspect that connects them, safety, is the most important factor. In all industries, tradeoffs between profit and regulations that enforce safety play an important role in business strategy. The standards are comparable, based on a common standard for quality management, risk management and the software development process.

Traditionally, the safety-critical industries were manufacturing industries (Porter & Heppelmann, 2015). The problems that show up in products reveal an underlying change in nowadays' products. Software is not supporting the hardware anymore, but is an integrated part of the product, that can be changed and updated, and where mistakes in manufacturing can lead to the same results as the

product it drives. Because of the long lifecycle of these safety-critical products, software needs be maintained and supported for just as long.

The medical device sector has been slower than other industries to endorse the importance of software in their development and regulations. This view changed with the addition of software into the medical device directive, and the upcoming medical device regulation, where software is categorized as active medical device. Since then, research has been conducted into creating a framework to enable enterprises to create safety-critical software for medical devices, such as MDevSPICE. Still, this framework is too comprehensive for SMEs to be used. An adapted version, MDevSPICE Med-Adept, can help small companies maturing their development processes.

Even though these methods exist, adoption of agile methods by the industry is slow, because of the fragile nature of safety-critical products. Lack of proof in the market makes companies hesitant to change.

In order to accelerate the adoption of standardized agile methods for micro- and small-sized enterprises in the medical device software sector, evidence of a working standard is necessary. The safety-critical industries have comparable standards, similar motivations and produce the same type of software products. We propose to adapt existing examples that have proven to work in other safety-critical industries for the medical device industry, by applying industry-specific standards and regulations.

Adapting a method from another, comparable industry is a common practice in research, which can accelerate the process of adoption. Adopting an existing, proven methodology helps gathering evidence of the possibility of agile development of safety-critical software. Therefore, we propose to use the SafeScrum standard. This method can be adjusted in order to meet the requirements from the development lifecycle standard for medical device software IEC 62304, which builds on the medical device-specific standards ISO 13485 for a quality management system and ISO 14971 for risk management. Experts can add to the body of knowledge by sharing practical knowledge. This may lead to a new lifecycle method to be used to develop medical device software, as shown in Figure 22.

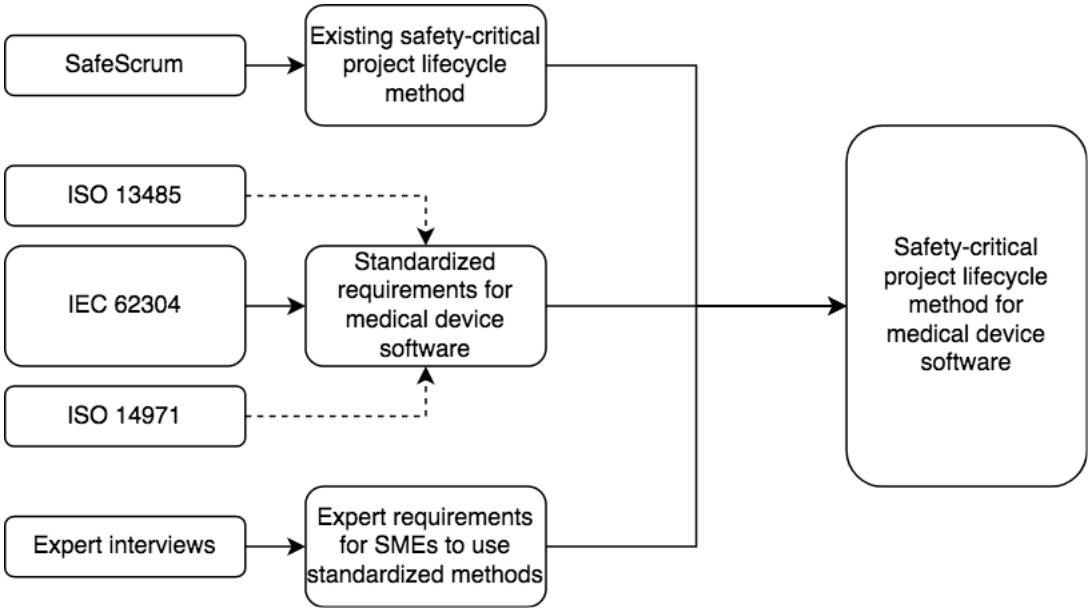

Figure 22: conducted study to create a project lifecycle method for medical device software

SQ2. How to engineer a method for agile safety-critical software development in the medical device industry?

Our research consisted of three parts. We gathered requirements for the method by interviewing experts (Section 6.3), reviewing the existing standards and regulations (Section 6.2), and took

lessons from an existing and validated agile method in the safety-critical industry, called SafeScrum (Section 6.1). Based on these results, we adapted the method to be used in a micro-sized organization (Section 7). This foundation for the method was put into practice and improved into an organization (Section 8). The results were validated through expert interviews. Interestingly, in all three stages, the same practical issues arose.

The operationalization issues that occurred can be put into three categories: process, tools and technology stack, and documentation. While processes and documentation have extensively been described in literature (Heeager & Nielsen, 2018; Kasauli et al., 2018), the focus of tools has been of individual nature, not a holistic view of tools in their context. We did not find any clear reason for the lack of research; however, the need of research in this direction is clear since tools support organizations in reducing overhead when developing safety-critical software. Literature pointed at this necessity (Hanssen et al., 2018), which was supported by our interviews with practitioners (Section 6.3).

Practitioners emphasize the wish and need from enterprises in the medical device software industry to develop agile. The requirements that we found from practitioners in Section 6.3, were related to certainty that their product would pass certification. Practitioners mention the difficulty to extract a practical implementation from standards and regulations, because according to them, these standards talk about what needs to be in place, not how this can be achieved. They mention the lack of a body of knowledge, based on successful examples from companies preceding them. As we described before, our research supports this. While research has been conducted in larger enterprises and theoretical research exists, evidence is missing from small enterprises. In online and offline knowledge sessions and websites, knowledge is shared by practitioners from SMEs in the medical device software industry. These informal and often temporary knowledge bases are worth investigating and solidifying in research, since they summarize best practices from practitioners who experienced and solved issues in the field. This would enrich the body of knowledge for small enterprises.

We found that for safety-critical software development in the medical device sector, important standards are ISO 14971 – risk management for medical devices, ISO 13485 – quality management for medical devices and IEC 62304 – processes for medical device software development. Because our research was aimed at the process for development of medical device software, we utilized requirements from the last standard. One of the practitioners we interviewed mentioned that from their experience, a bigger emphasis is placed on the quality management system by notified bodies than on a fully documented process. Also, notified bodies regard documented decisions higher than a fully documented process, according to this practitioner.

For our research, we adapted an existing method SafeScrum to fit the needs of a micro- to small-sized organization. This method allows for agile development, while also complying to regulatory standards. During adaption, we focused on documentation, tools usage, roles and the development process.

SQ3. What is the performance of the engineered method?

Due to the need for an operationalization of a method to be used for the development of medical device software, we operationalized and implemented our adapted method in a micro-sized enterprise (Section 8). We ran one project for six weeks, in two sprints of three weeks each, using this method. The development team consisted of three persons, including the author from this research. Three persons was the assumed minimum amount of people possible, based on the roles in the adapted method. The project consisted of the development of software as a medical device, an application to be used in hospitals to support the early rehabilitation of patients (ERAS) after surgery according to the standardized ERAS protocol.

We found that the adapted method gives a basis for micro- to small-sized enterprises developing applications in a safety-critical industry. The method provides guidelines for documentation, tool usage, process workflow and roles and responsibilities, while keeping overhead to a minimum.

However, during operationalization, practical questions arose from developers. While research often focuses on the theoretical part of a method, in this case, the questions were of a practical nature. Together with the enterprise and practitioners, we found solutions to these questions and solidified them in our research. By documenting decisions that are made, we were able to use the preferred programming language, framework, SOUP and digital documentation method.

By answering our sub-research questions, we were able to give an answer to our main research question:

How can SMEs in the medical device industry perform agile software development that complies to standards and safety regulations?

In this research, we aimed to develop a method that can be used by micro- and small-sized enterprises to develop safety-critical software in the medical device sector. For micro and small-sized enterprises, it is difficult to start developing safety-critical software due to strict regulations, a lack of financial capability and limited resources. Since agile methods are lightweight, they lend themselves for implementation into small organizations. Traditionally however, safety-critical software is developed in a waterfall V-model. Within the V-model, safety and traceability can be guaranteed. On first sight, there seems to be a mismatch between agile development and safety-critical software. In the past years, research has been aimed at making agile development possible for safety-critical systems in medium- and large sized organizations.

Micro- and small-sized enterprises have been left out of the research area, while they represent the largest number of companies in the European Union with about half of the workforce being employed in these small-sized companies (Section 4.1). No body of knowledge is available for small companies that want to start developing software for medical devices. A solution that is available to these companies, is hiring an external consultant or a specialized employee to interpret safety standards and regulations, establish their process and documents (Section 6.3). However, due to their nature in limited resources and lack of financing, this is often not an option for micro- and small-sized enterprises (Section 4.4).

During our research, we found that micro- and small-sized enterprises in the medical device software industry have similar issues and the path towards findings answers is comparable, when starting with safety-critical software development. Due to similarity in safety-critical industries and the size of these organizations (Section 5), we assume that issues like these arise in all safety-critical industries. What these companies need, we found, is an operationalized method that they can implement in their organizations. In our research we tried to solve several of these issues by operationalizing and adapting an agile process for developing software in the medical device industry.

Our research has been focused on the medical device industry. This is only one of many safety-critical industries that develop software. We have seen that standards and regulations are comparable in industries (Section 5.2). Motives for agile adoption are shared between industries (Section 5.3). While the medical device industry moves at a slower pace, this method might be applicable to other micro- or small-sized enterprises in other safety-critical industries.

We have applied our adapted method in a micro-sized enterprise in the Netherlands. We can argue that application in other micro-sized enterprises, in the Netherlands or abroad, would prove possible. Agile, and scrum methods have proven themselves all around the world with little adaptation necessary per country. SafeScrum has been used in several enterprises (Hanssen et al., 2018; Myklebust, Stalhane, et al., 2016; Stålhane et al., 2012). Our scope has focused on EU regulations. While standards might be globally shared, we have seen that other regulatory bodies such as the FDA in the USA have different interpretations and regulations. Therefore, outside of the EU adaptions may be necessary before the method can be used.

10. Conclusions

The tragic case of the Boeing 737 MAX proves misjudgment of tradeoffs that safety-critical industries have to deal with can happen, even if there are standards and working methods in place to prevent these. Realistically, the goal in these industries is to minimize risk and to rule out unacceptable risk. Traditional methods interact with the company structure and keeps hierarchy in place. In the case of Boeing, fast decisions and a management pushing deadlines led to failure in identifying risk, even though engineers warned during the development process.

This research aimed to identify the state of art in safety-critical software development within SMEs for medical devices. Based on a literature review, we conclude that SMEs rarely use standardized methods to develop software in the safety-critical industry, because of caused overhead and a perceived lack of benefit. Agile methods pose themselves as lightweight candidates for SMEs to use. In the safety-critical industries, comparable to the medical device industry, companies are shifting from traditional methods to agile methods, due to the increase in efficiency and quality of delivered products seen by regular industries. The medical device industry is slower than other safety-critical industries to adopt these methods, because of a lack of proof that these agile methods enable certifiable software.

However, agile methods are difficult to implement into safety-critical industries. Four problem areas are found and explored: documentation, requirements, development lifecycle and testing. For three of these (documentation, requirements and testing), findings from the literature review suggest solutions. Documentation needs to fit certification, so in an agile method the documentation of should become part of the development lifecycle. Requirements are necessary for traceability and understanding, so they should be documented and referred to during development. Agile methods allow for a test-first process, where tests are defined before implementation.

The development lifecycle of agile methods is more comprehensive, which could be the reason that in recent years this topic has had little research. We conducted a research into developing a method to support small- and medium-sized enterprises to develop software agile for the medical device industry. By using a standardized, repeatable process, these companies could produce certifiable software that is of higher quality, thereby driving innovation in the industry. If the medical device industry closes the gap with other safety-critical industries, it becomes easier to share and propagate knowledge across industries.

Sharing knowledge in practice is important in this industry. While regulations give an impression about what needs to exist for certification, they do not provide the information on how to achieve certification. Our research gave insight into possibilities for micro- and small enterprises to work agile in the safety-critical industry, while striving for certification of their software. By decreasing documentation, automating testing using tools, and setting up a defined process, the overhead for development is kept to a minimum.

Regulatory-wise, we have shown that regulations and standards do not officially prescribe a method, but using another method than the traditional development method is implicitly discouraged by the rigorous framework the regulations set. In order to open up market entry for smaller companies, we would expect legislators and standards organizations to open up their framework to newer methods such as agile.

10.1. Contributions

Micro- to small-sized enterprises make up a large part of the safety-critical software development industry for medical devices. Their needs are very specific; due to limited resources, time and budget, enterprises of this size cannot afford the overhead that traditionally comes with regulations and compliance. During our literature research, we have seen that due to strict regulation, complexity in documentation, individualistic approaches on tools and the rigidness of traditional waterfall methods, the entry to safety-critical software development is difficult for micro-sized organizations.

Therefore, this research exposed research questions that concern these small organizations. We adapted a method that is used in practice by medium- to large-sized enterprises to develop software in the safety-critical industry. Specific attention went to adapting this to the medical device industry, where more than on average micro- and small-sized enterprises are represented. We integrated knowledge from other safety-critical industry: aviation, aerospace, railway systems and the automotive industry.

Contribution to Science

Before our research, to our knowledge there was little attention for micro- to small-sized enterprises developing software in the safety-critical medical device industry. We found that about 95 percent are micro- to small-sized enterprises in the medical device industry. Because of the importance that this group of enterprises gets the possibility to enter the market due to their innovative nature, we focused our research on creating a method to develop compliant software.

We have exposed in our research that these enterprises have a very specific and pragmatic challenge in implementation. While regulations prescribe *what* has to be done in order to be compliant, it does not prescribe *how* to achieve compliance. Current methods are too comprehensive for micro- to small-sized enterprises, and micro- to small-sized enterprises do not have the financial capabilities or resources to invest in creating a compliant development process. Our interviews with practitioners show that this is a recurring issue (Section 6.3).

Scientific research has focused on four problem areas and their relationships in safety-critical software development. Heeager & Nielsen (2018) mention on four aspects problems with agile development:

- Documentation in an agile method should be “barely sufficient”. We have decreased documentation to fit in micro- and small-sized enterprises.
- Changing requirements might be problematic, since it is difficult to show traceability. We created documentation digitally and kept track of changes using built-in methods. This improves tracking of revisions and authors of changes.
- A testing-first method was seen as good practice in agile, but difficult to implement because developers are not used to it. We used testing-first as our process for implementing user stories, with good results. Because of the testing-first approach, first implementations of code would lead to less code reviews than what the company was used to.

Lastly though, the development lifecycle is seen as an open and underexposed research topic (Kasauli et al., 2018). During our implementation, we have shown that the process of SafeScrum provides safety features to develop safety-critical software. Hazard and risk assessment, a safety plan and safety procedures in the method safeguard the development process.

A lack of evidence inhibits the adoption of agile methods in safety-critical software development, Fitzgerald et al. (2013) mentioned. Our research contributes to the knowledge base on this subject in two ways. By implementing and validating our adapted agile method, our evidence shows that in micro- and small-sized enterprises, agile development of safety-critical software is possible. Also, our practitioner interviewees all mentioned using agile methods in their SMEs.

Contributions to Practice

In practice, operationalization and implementation of methods using standards have shown to be problematic for micro- and small enterprises (Denger et al., 2007). Practitioners confirm this in our interviews (Section 6.3). Due to a lack or resources and financial capability, elaborate agile or traditional methods are not suitable for micro- to small-sized enterprises.

We have tried to optimize the usage of resources. Documentation that is prescribed in SafeScrum has been combined where possible to make documents concise for the enterprise, while the documentation is still comprehensive for certification. Keeping track of documentation revisions was done digitally. We combined roles for smaller teams, in an effort to make safety-critical software development possible for micro-sized enterprises. In this way, it becomes possible for smaller enterprises to comply to regulations while developing safety-critical software. Quality assurance was

largely automated with the usage of tools, to decrease the necessity of manual work. Tools can be an important help for micro- and small-sized development enterprises moving into safety-critical software.

Research suggested that real evidence of work practice is missing (Hanssen et al., 2018; Heeager & Nielsen, 2018; Kasauli et al., 2018). In order to bring the research into the real world, we operationalized and implemented our adapted method in practice in a micro-sized enterprise. This has shown that adapting a method for usage by micro- and small-sized enterprises allows these enterprises to develop compliant software.

More importantly, we have shown that agile working is possible for micro- and small-sized organizations developing safety-critical software. When documented well, these processes are fit for use in the medical device industry. Practitioners exclusively use agile methods to develop their software. Methods can be adapted to fit smaller and larger organizations. For a team of three, the responsibilities could not sufficiently be divided to fill all roles. A team of five however, might be able to fill the requirements, as one of the practitioners has mentioned as well. The roles of scrum master, product owner, and RAMS engineer could then properly be filled by members of the development team, leaving two developers for full-time development to balance the speed of development.

10.2. Limitations

In our research, we explicitly focused on micro- to small-sized enterprises, because these categories have the biggest problems setting up a method. For medium-sized enterprises, it seems that existing methods suffice. It also becomes easier for medium-sized enterprises to hire professional support (consultants, employees) that have proficient knowledge about setting up these methods and regulations.

Due to the limited time of study, we investigated two sprints within a micro-sized enterprise. Ideally, the method would be tested in multiple iterations in a micro-sized enterprise. The method improves every sprint, with a feedback loop built into scrum.

We aimed at developing at a low-risk class in order to make certification easier for the application in development. In practice, this risk classification can be derived from the intended use of the application and can in that sense be influenced by it. This makes it difficult to generalize the method for higher risk classifications.

10.3. Recommendations and Future Research

This research engineered a method for safety-critical software development by micro- and small enterprises in the medical device industry. While this research made a start on answering research questions for science and practice, we recommend three actors to continue research in varying context to make the method more accessible, and market entry easier.

Legislators
During our research, we encountered two common recurring themes: innovation and barriers. While the European Union stimulates startups in the medical device sector to innovate, complex regulations and standards make it difficult for these startups to get to market. It would make sense that regulations and standards open up, and make certification accessible for micro- and small-sized enterprises. We see that micro- to small-sized enterprises often develop software in lower risk categories. Making an adjusted certification for these lower risk classes would make sense, since risks are lower but investments in resources and time are still serious.

Researchers
One of our literature findings was that proposed solutions often lack proof in practice (Heeager & Nielsen, 2018). We have shown that adapting a method allows micro- and small-sized enterprises to develop compliant software. Due to the limited scope of the method in practice, we suggest

repeating the action design research with the adapted method of SafeScrum. In this way, the method gets more refined over time, while building proof that the method works. We have considered a low risk classification and a small team. Different team compositions, and higher risk classifications are recommended candidates to build proof in the method.

For higher risk classifications, it is expected that more documentation is necessary to show compliance. The interaction with a notified body to get the software certified is another aspect that comes into play with a higher risk class. Lastly, a larger team will allow for more focused roles. The interaction and responsibilities will change with changing team sizes. While roles can be combined, the focus of certain roles is necessary in order to satisfy a balance between compliance and development progress. Researchers should consider that having a method available does not mean that a method gets adopted easily. Effort should be put in making the method easily understandable and accessible due to the nature of the enterprises the method is created for – with small financial capabilities and resources.

For micro- and small-sized enterprises new in this industry, we see a challenge in starting to develop safety-critical software. Setting up a process and using default tools, documenting can be quite overwhelming and prioritizing which steps to take might lead to no adoption at all. A maturity model fit for safety-critical software development could guide these enterprises in the right way of certification, giving them a guideline to start standardizing the most important parts of the process first. A maturity level within the enterprise, where processes are defined and a repeatable process is possible, is necessary in order to create compliant software. MDevSPICE Med-Adept opens itself up as possible candidate, but is focused on the Irish market (Özcan-Top & McCaffery, 2017a).

In this research, we did not regard the security of software. Of course, the interconnected world we live in nowadays pose new risks concerning software security. Therefore, software security has to be seen as an integral part of software development. Recent hacks have shown that medical device software does not go without attention, and data of patients is prone to abuse (Heikkila & Cerelus, 2020). Future work should focus on including software security in the software development lifecycle of safety-critical software to guarantee safeguarding often intimate data of patients. This might include including risk assessment, quality attributes and testing on the security of software.

Practice

We have seen that micro- and small-sized enterprises struggle with setting up a process for safety-critical software development (Section 6.3). This is due to the complexity of legislative requirements, where implementation is left up to the enterprise itself. One of the practitioners suggested to contact a consulting firm when setting up a process within the enterprise. However, research suggests that a consultant is no guarantee for success. Next to that, consultants might still cut into financial capabilities of the company. We found knowledge networks that are active in the medical device software industry, and similar industries. They actively share knowledge on legislation, implementation and agile development of software. Documentation of this can be found online. Connecting to such a knowledge network might allow micro-sized enterprises to gain the knowledge they lack when entering this market.

Enterprises need to realize that entering the safety-critical software markets require investments in financial capabilities and resources. At the moment, this does not seem easily possible when building enterprises in a traditional way. External investment is necessary in order to be able to finance resources, the setup of quality and risk management systems, and certification of process and software.

Lastly, research suggests that small enterprises rarely use formalized methods to develop safety-critical enterprises (Denger et al., 2007). We found that setting up a method in a micro-sized enterprises is time-consuming and requires serious effort. Enterprises notice difficulties when trying to show compliance after developing software, without a structured approach. We therefore recommend micro-sized enterprises to start on documenting their process as soon as development starts and to maintain it. Focus on elaborating decisions in the development proces, as our practitioners suggest (Section 6.3).

Bibliography

Amalberti, R. (2001). The paradoxes of almost totally safe transportation systems. *Safety Science*, *37*(2–3), 109–126. https://doi.org/10.1016/S0925-7535(00)00045-X

Ardic, O. P., Mylenko, N., & Saltane, V. (2011). Small and Medium Enterprises A Cross-Country Analysis with a New Data Set. In *World Bank Policy Research Working Paper* (Vol. 5538, Issue January). https://doi.org/10.1596/1813-9450-5538

Ashok, A., Niyogi, D., Ranganathan, P., Tandon, S., Bhaskar, M., Karimundackal, G., Jiwnani, S., Shetmahajan, M., & Pramesh, C. S. (2020). The enhanced recovery after surgery (ERAS) protocol to promote recovery following esophageal cancer resection. *Surgery Today*, *50*(4), 323–334. https://doi.org/10.1007/s00595-020-01956-1

Audretsch, D. B. (2000). The Economic Role of Small- and Medium-Sized Enterprises: the United States. *World Bank Workshop on Small and Medium Enterprises*, *9*(2), 69.

Awad, M. A. (2005). *A Comparison between Agile and Traditional Software Development Methodologies*.

Beck, K., Grenning, J., Martin, R., Beedle, M., Highsmith, J., & Mellor, S. (2001). Manifesto for Agile Software Development. *The Agile Alliance*. http://agilemanifesto.org/

Beck, T. (2007). Financing Constraints of SMEs in Developing Countries : Evidence , Determinants and Solutions. *Financing Innovation-Oriented Businesses to Promote Entrepreneurship*, *April*, 1–35.

Blasimme, A., Ferretti, A., & Vayena, E. (2021). Digital Contact Tracing Against COVID-19 in Europe: Current Features and Ongoing Developments. *Frontiers in Digital Health*, *3*. https://doi.org/10.3389/fdgth.2021.660823

Boeing Finds New Software Problem That Could Complicate 737 MAX's Return - WSJ. (n.d.). Retrieved May 4, 2020, from https://www.wsj.com/articles/boeing-finds-new-software-problem-that-could-complicate-737-max-return-11579290347

Borgerink, R. (2020). *Van privacy tot stroomverbruik: antwoorden op al je vragen over de CoronaMelder - RTV Oost*. RTV Oost. https://www.rtvoost.nl/nieuws/334550/Van-privacy-tot-stroomverbruik-antwoorden-op-al-je-vragen-over-de-CoronaMelder

Brown, A., van der Wiele, T., & Loughton, K. (1998). Smaller enterprises' experiences with ISO 9000. *International Journal of Quality and Reliability Management*, *15*(3), 273–285. https://doi.org/10.1108/02656719810198935

Bujok, A. B., MacMahon, S. T., Grant, P., Whelan, D., Rickard, W. J., & McCaffery, F. (2017). Approach to the development of a Unified Framework for Safety Critical Software Development. *Computer Standards and Interfaces*, *54*, 152–161. https://doi.org/10.1016/j.csi.2016.11.013

Bujok, A. B., MacMahon, S. T., McCaffery, F., Whelan, D., Mulcahy, B., & Rickard, W. J. (2016). Safety Critical Software Development – Extending Quality Management System Practices to Achieve Compliance with Regulatory Requirements. In *Software Quality Professional* (Vol. 19, Issue 2, pp. 17–30). https://doi.org/10.1007/978-3-319-38980-6_2

Cawley, O., Wang, X., & Richardson, I. (2010). Lean/agile software development methodologies in regulated environments - State of the art. *Lecture Notes in Business Information Processing*, *65 LNBIP*, 31–36. https://doi.org/10.1007/978-3-642-16416-3_4

CENELEC. (2011). *CENELEC - About CENELEC - What we stand for - Support legislation*. https://www.cenelec.eu/aboutcenelec/whatwestandfor/supportlegislation/newapproachdirectives.html

Cheng, M. (2003). *Medical Device Regulations - Global Overview and Guiding Principles*.

Cockburn, A. (2000). Selecting a project's methodology. *IEEE Software*, *17*(4), 64–71. https://doi.org/10.1109/52.854070

Cooke, J. L. (2012). *Everything You Want to Know About Agile : How to Get Agile Results in a Less-than-agile Organization*. ITGP. https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=nlebk&AN=571555&lang=nl&site=ehost-live&custid=s3628809

Cottmeyer, M. (2009). The Agile Project Manager. *Manager*, 1–4. http://www.stickyminds.com/sitewide.asp?ObjectId=15869&Function=DETAILBROWSE&ObjectType=ART&sqry=*Z(SM)*J(MIXED)*R(relevance)*K(simplesite)*F(prodcut+owner)*&sidx=9

&sopp=10&sitewide.asp?sid=1&sqry=*Z(SM)*J(MIXED)*R(relevance)*K(simplesite)*F(product+owner
DARE!! (2017). *MDD to MDR :: DARE!! EU*. https://www.dare.eu/notified-body-medical-devices/from-mdd-to-mdr
Davis, A. M., Bersoff, E. H., & Comer, E. R. (1988). A Strategy for Comparing Alternative Software Development Life Cycle Models. *IEEE Transactions on Software Engineering*, *14*(October), 453–461. https://doi.org/10.1109/9781118156674.ch6
Denger, C., Feldmann, R. L., Host, M., Lindholm, C., & Shull, F. (2007). A Snapshot of the State of Practice in Software Development for Medical Devices. *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, 485–487. https://doi.org/10.1109/ESEM.2007.54
Doss, O., & Kelly, T. (2016). The 4+1 Principles of Software Safety Assurance and Their Implications for Scrum. In *Lecture Notes in Business Information Processing* (Vol. 251, pp. 286–290). https://doi.org/10.1007/978-3-319-33515-5_27
Douglass, B. P. (2014). Meeting Industry Standards. *Real-Time UML Workshop for Embedded Systems*, 67–88. https://doi.org/10.1016/b978-0-12-407781-2.00003-9
Eidel, O. (2020a). *How to document SOUP for IEC 62304 compliance?* https://openregulatory.com/how-to-document-soup-iec-62304/
Eidel, O. (2020b). *QMS Software for Medical Devices (ISO 13485): The Ultimate Comparison*. https://openregulatory.com/qms-software-iso-13485-comparison/
Eidel, O. (2020c, July 26). *Medical Device Software Architecture Documentation (IEC 62304)*. https://openregulatory.com/medical-device-software-architecture-documentation-iec-62304/
Eidel, O. (2021). *Is Our Software a Medical Device (and Do We Need to Certify It)?* https://openregulatory.com/is-our-software-medical-device/
Ericson, C. A. (2011). *Concise Encyclopedia of System Safety : Definition of Terms and Concepts*. Wiley. https://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=nlebk&AN=364837&lang=nl&site=ehost-live&custid=s3628809
European Commission. (2003). Commission Recommendation of 6 May 2003 Concerning the Definition of Micro, Small and Medium-Sized Enterprises. *Official Journal of the European Union*, *L 124*, 6. https://doi.org/10.1093/nq/s10-I.5.88-c
European Commission. (2008). *New legislative framework | Internal Market, Industry, Entrepreneurship and SMEs*. https://ec.europa.eu/growth/single-market/goods/new-legislative-framework_en
European Council. (2009). *SCADPlus: Medical devices*. https://web.archive.org/web/20090324163309/http://europa.eu/scadplus/leg/en/lvb/l21010b.htm
European Union. (2017). Regulation (EU) 2017/745 of the European Parliament and of the Council. *Official Journal of the European Union*, 175.
Eurostat. (2015). *Small and medium-sized enterprises (SMEs) - Eurostat*. https://ec.europa.eu/eurostat/web/structural-business-statistics/structural-business-statistics/sme
Fitzgerald, B., Stol, K. J., O'Sullivan, R., & O'Brien, D. (2013). Scaling agile methods to regulated environments: An industry case study. *Proceedings - International Conference on Software Engineering*, 863–872. https://doi.org/10.1109/ICSE.2013.6606635
Gasparic, M., Janes, A., & Ricci, F. (2016). Development tools usage inside out. *Lecture Notes in Business Information Processing*. https://doi.org/10.1007/978-3-319-33515-5_28
Ge, X., Paige, R. F., & McDermid, J. A. (2010). An iterative approach for development of safety-critical software and safety arguments. *Proceedings - 2010 Agile Conference, AGILE 2010*, 35–43. https://doi.org/10.1109/AGILE.2010.10
Girson, A., & Barr, M. (2018). *Embedded Systems Safety & Security Survey*. *1*(866), 1–61. www.barrgroup.com
Hanssen, G. K., Stålhane, T., & Myklebust, T. (2018). SafeScrum® – Agile Development of Safety-Critical Software. In *SafeScrum® – Agile Development of Safety-Critical Software*. https://doi.org/10.1007/978-3-319-99334-8
Hart, C. (1998). The literature review in research: Releasing the social science imagination. In *Doing a Literature Review*.

Heeager, L. T., & Nielsen, P. A. (2018). A conceptual model of agile software development in a safety-critical context: A systematic literature review. *Information and Software Technology*, *103*(June), 22–39. https://doi.org/10.1016/j.infsof.2018.06.004

Heikkila, M., & Cerelus, L. (2020). *Hacker seeks to extort Finnish mental health patients after data breach – POLITICO*. https://www.politico.eu/article/cybercriminal-extorts-finnish-therapy-patients-in-shocking-attack-ransomware-blackmail-vastaamo/

Herr, H., & Nettekoven, Z. M. (2018). *The Role of Small and Medium-Sized Enterprises in Development : What can be Learned from the German Experience?* (No. 53).

Heuvel, van den R., Stirling, C., Kapadia, A., & Zhou, J. (2018). Medical devices 2030 - Making a power play to avoid the commodity trap. *KPMG International*, 25. https://assets.kpmg.com/content/dam/kpmg/xx/pdf/2017/12/medical-devices-2030.pdf

Highsmith, J. (2001). *History: The Agile Manifesto*. https://agilemanifesto.org/history.html

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

Hurd, D. (2018). *Paper-based QMS*. https://elsmar.com/elsmarqualityforum/threads/paper-based-qms.74564/#post-620472

ISO. (2016). *ISO 13485 Quality management for medical devices*.

ISO. (2018). *COPOLCO*. https://www.iso.org/sites/ConsumersStandards/1_standards.html

Jansch, I., Duran, E., Broersma, B. W., Maarten, D., Scholten, A., de Valk, T., Schultze, H., van Gulik, D.-W., Kruining, C., & Koot, W. (2020). *nl-covid19-notification-app-coordination/Solution Architecture.md at master · minvws/nl-covid19-notification-app-coordination*. https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/Solution Architecture.md#native-vs-hybrid-development

Jonsson, H., Larsson, S., & Punnekkat, S. (2012). Agile practices in regulated railway software development. *Proceedings - 23rd IEEE International Symposium on Software Reliability Engineering Workshops, ISSREW 2012*, 355–360. https://doi.org/10.1109/ISSREW.2012.80

Julien, P.-A., & Ramangalahy, C. (2003). Competitive Strategy and Performance of Exporting SMEs: An Empirical Investigation of the Impact of Their Export Information Search and Competencies. *Entrepreneurship Theory and Practice*, *27*(3), 227–245. https://doi.org/10.1111/1540-8520.00013

Kasauli, R., Knauss, E., Kanagwa, B., Nilsson, A., & Calikli, G. (2018). Safety-critical systems and agile development: A mapping study. *Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018*, 470–477. https://doi.org/10.1109/SEAA.2018.00082

Kettunen, J., Reiman, T., & Wahlström, B. (2007). Safety management challenges and tensions in the European nuclear power industry. *Scandinavian Journal of Management*, *23*(4), 424–444. https://doi.org/10.1016/j.scaman.2007.04.001

Kettunen, P., & Laanti, M. (2005). How to steer an embedded software project: Tactics for selecting the software process model. *Information and Software Technology*, *47*(9), 587–608. https://doi.org/10.1016/j.infsof.2004.11.001

Komite Nasional Keselmatan Transportasi. (2019). *FINAL Aircraft Accident Investigation Report. Lion Airlines. Boeing 737-8(MAX)*. *8*(October 2018). http://knkt.dephub.go.id/knkt/ntsc_home/ntsc.htm

Kuhrmann, M., Diebold, P., Münch, J., Tell, P., Garousi, V., Felderer, M., Trektere, K., McCaffery, F., Linssen, O., Hanser, E., & Prause, C. R. (2017). Hybrid software and system development in practice: Waterfall, scrum, and beyond. *ACM International Conference Proceeding Series*, *Part F1287*, 30–39. https://doi.org/10.1145/3084100.3084104

Kuperus, K., & van der Schrier, B. (2017). *The MedTech market in the Netherlands*.

Larrucea, X., Combelles, A., Favaro, J., & Spa, I. (2013). Safety-Critical Software. *IEEE Software*, 25–27.

Lepmets, M., Mc Caffery, F., & Clarke, P. (2015). Piloting MDevSPICE: the medical device software process assessment framework. *Proceedings of the 2015 International Conference on Software and System Process - ICSSP 2015*, *August*, 9–16. https://doi.org/10.1145/2785592.2785598

Lepmets, M., Mccaffery, F., & Clarke, P. (2015). *Piloting MDevSPICE : The Medical Device Software Process Assessment Framework*. 9–16.

Lepmets, M., McCaffery, F., & Clarke, P. (2016). Development and benefits of MDevSPICE®, the

medical device software process assessment framework. *Journal of Software: Evolution and Process*, *28*(9), 800–816. https://doi.org/10.1002/smr.1781
Levy, Y., & Ellis, T. J. (2006). A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research. *Informing Science: The International Journal of an Emerging Transdiscipline*, *9*, 181–212. https://doi.org/10.28945/479
Martins, L. E. G., & Gorschek, T. (2016). Requirements engineering for safety-critical systems: A systematic literature review. *Information and Software Technology*, *75*, 71–89. https://doi.org/10.1016/j.infsof.2016.04.002
Mayer, R. J., Menzel, C. P., & Painter, M. K. (1995). PS deWitte, T. Blinn, and B. Perakath. Information integration for concurrent engineering (IICE): IDEF3 process description capture method report. *Knowledge Based Systems Inc*, *April 1992*.
McAdam, R., & Reid, R. (2001). SME and large organisation perceptions of knowledge management: Comparisons and contrasts. *Journal of Knowledge Management*, *5*(3), 231–241. https://doi.org/10.1108/13673270110400870
McBride, T., & Lepmets, M. (2017). Quality assurance in agile safety-critical systems development. *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016*, 44–51. https://doi.org/10.1109/QUATIC.2016.016
McHugh, M., Casey, V., Pikkarainen, M., Hugh, M. M., Caffery, F. M., Casey, V., & Pikkarainen, M. (2012). *Integrating Agile Practices with a Medical Device Integrating Agile Practices with a Medical Device Software Development Lifecycle*. 0–8.
McHugh, M., McCaffery, F., Fitzgerald, B., Stol, K. J., Casey, V., & Coady, G. (2013). Balancing Agility and Discipline in a Medical Device Software Organisation. *Communications in Computer and Information Science*. https://doi.org/10.1007/978-3-642-38833-0_18
MedTech Europe. (2019). *The European Medical Technology Industry – in figures 2019*. 44.
Michaud, C. (2016). *Is my software in class I, IIa, IIb or III - 2016 Revolution - Software in Medical Devices, by MD101 Consulting*. MD101 Consulting. https://blog.cm-dm.com/post/2016/07/22/Is-my-software-in-class-I%2C-IIa%2C-IIb-or-III-2016-Revolution
Ministry of Transport Ethiopia. (2019). *Aircraft Accident Investigation Bureau Preliminary Report*. *8*. http://www.ecaa.gov.et/Home/wp-content/uploads/2019/07/Preliminary-Report-B737-800MAX-ET-AVJ.pdf
Mishra, A., & Dubey, D. (2013). A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios. *International Journal of Advance Research in Computer Science and Management Studies*, *1*(5), 2321–7782. http://www.ijarcsms.com/docs/paper/volume1/issue5/V1I5-0008.pdf
Mittal, S., Khan, M. A., Romero, D., & Wuest, T. (2018). A critical review of smart manufacturing & Industry 4.0 maturity models: Implications for small and medium-sized enterprises (SMEs). *Journal of Manufacturing Systems*, *49*(November), 194–214. https://doi.org/10.1016/j.jmsy.2018.10.005
Muller, P., Robin, N., Jessie, W., Schroder, J., Braun, H., Becker, L. S., Farrenkopf, J., Ruiz, F. A., Caboz, S., Ivanova, M., Lange, A., Lonkeu, O. K., Muhlshlegel, T. S., Pedersen, B., Privitera, M., Bomans, J., Bogen, E., & Cooney, T. (2019). *Annual Report on European SMEs 2018/2019. Research & Development and Innovation by SMEs*. https://ec.europa.eu/docsroom/documents/38365/attachments/2/translations/en/renditions/native
Mwadulo, M. W. (2016). Suitability of Agile Methods for Safety-Critical Systems Development: A Survey of Literature. *International Journal of Computer Applications Technology and Research*, *5*(7), 465–471. https://doi.org/10.7753/ijcatr0507.1009
Myklebust, T., Lyngby, N., & Stålhane, T. (2016). *The Agile Safety Plan*. *October*.
Myklebust, T., Stålhane, T., & Hanssen, G. (2016). Use of Agile Practices when developing Safety-Critical Software. *ISSC 2016*.
Myklebust, T., Stalhane, T., & Lyngby, N. (2016). An agile development process for petrochemical safety conformant software. *Proceedings - Annual Reliability and Maintainability Symposium*, *2016-April*(December 2017). https://doi.org/10.1109/RAMS.2016.7448075
NEN. (2006). *Nen-en-iec 62304*.
Newman, R. (2019). *The Boeing 737 Max is now deadlier than the Concorde*. Yahoo Finance. https://finance.yahoo.com/news/the-boeing-737-max-is-now-the-deadliest-mainstream-

jetliner-203441321.html
Oortwijn, W., Weistra, K., Nieuwdorp, C., Meindert, L., & Hurkmans, E. (2018). *The future of the medical technology market: Addressing challenges and utilising opportunities* (Issue september).
Özcan-Top, Ö., & McCaffery, F. (2017a). A Lightweight Software Process Assessment Approach Based on MDevSPICE® for Medical Device Development Domain. In *Computer Standards and Interfaces* (Vol. 36, Issue 1, pp. 578–588). https://doi.org/10.1007/978-3-319-64218-5_48
Özcan-Top, Ö., & McCaffery, F. (2017b). How Does Scrum Conform to the Regulatory Requirements Defined in MDevSPICE®? In *Computer Standards and Interfaces* (Vol. 60, pp. 257–268). https://doi.org/10.1007/978-3-319-67383-7_19
Paige, R. F., Charalambous, R., Ge, X., & Brooke, P. J. (2008). Towards agile engineering of high-integrity systems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5219 LNCS*, 30–43. https://doi.org/10.1007/978-3-540-87698-4_6
Poort, E. R., & Van Vliet, H. (2012). RCDA: Architecting as a risk- and cost management discipline. *Journal of Systems and Software*, *85*(9), 1995–2013. https://doi.org/10.1016/j.jss.2012.03.071
Porter, M. E. (1990). The Competitive Advantage of Nations. *Harvard Business Review*, 73–93.
Porter, M. E., & Heppelmann, J. E. (2015). How smart, connected products are transforming companies. *Harvard Business Review*, *2015*(October).
Rasmussen, R., Hughes, T., Jenks, J. R., & Skach, J. (2009). Adopting agile in an FDA regulated environment. *Proceedings - 2009 Agile Conference, AGILE 2009*, 151–155. https://doi.org/10.1109/AGILE.2009.50
Rook, P. (1986). Controlling software projects. *Software Engineering Journal*, *1*(1), 7. https://doi.org/10.1049/sej.1986.0003
Salles, M. (2006). Decision making in SMEs and information requirements for competitive intelligence. *Production Planning and Control*, *17*(3), 229–237. https://doi.org/10.1080/09537280500285367
Saunders, F. (2015). *Safety–critical industries: definitions, tensions and tradeoffs | Pedagogy and Practice: Educating the World of Project Management*. http://fionasaunders.co.uk/safety-critical-industries-definitions-tensions-and-tradeoffs/
SBA. (2016). *Table of USA Small Business Size Standards*. *202*, 46. https://www.sba.gov/sites/default/files/files/Size_Standards_Table.pdf
Schwaber, K., & Sutherland, J. (2020). *Scrum Guide | Scrum Guides*. https://scrumguides.org/scrum-guide.html#sprint-retrospective
Sein, Henfridsson, Purao, Rossi, & Lindgren. (2011). Action Design Research. *MIS Quarterly*, *35*(1), 37. https://doi.org/10.2307/23043488
Select USA. (2012). *The Medical Device Industry in the United States*. https://selectusa.github.io/events/industry-snapshots/medical-device-industry-united-states.html
Spence, J. W. (2005). There has to be a better way! *Proceedings - AGILE Confernce 2005*, *2005*, 272–278. https://doi.org/10.1109/ADC.2005.47
Spence, P., Babitt, J., Welch, J., & De Busscher, L. (2017). *As Change Accelerates, How Can Medtechs Move Ahead and Stay There? - Pulse of the Industry 2017*. http://search.ebscohost.com/login.aspx?direct=true&db=buh&AN=16069837&site=ehost-live
Stålhane, T., Myklebust, T., & Hanssen, G. (2012). The application of safe scrum to IEC 61508 certifiable software. *11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference 2012, PSAM11 ESREL 2012*, *8*, 6052–6061.
Sulaman, S. M., Orucevic-Alagic, A., Borg, M., Wnuk, K., Host, M., & De La Vara, J. L. (2014). Development of safety-critical software systems using open source software- A systematic map. *Proceedings - 40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014*, *May*, 17–24. https://doi.org/10.1109/SEAA.2014.25
Theocharis, G., Kuhrmann, M., Münch, J., & Diebold, P. (2015). Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices. In *International Conference on Product-Focused Software Process Improvement* (pp. 149–166). https://doi.org/10.1007/978-3-319-26844-6_11
van de Vrande, V., de Jong, J. P. J., Vanhaverbeke, W., & de Rochemont, M. (2009). Open

innovation in SMEs: Trends, motives and management challenges. *Technovation*, *29*(6–7), 423–437. https://doi.org/10.1016/j.technovation.2008.10.001
van der Voort, M. G. P. E., & Kroeks-de Raaij, C. C. M. (2020). De “concessievrije” CoronaMelder? In *Tijdschrift voor Internetrecht* (pp. 285–287).
van Dijk, R. W. (2011). Determining the Suitability of Agile Methods for a Software Project. *15th Twente Student Conference on IT*. http://referaat.cs.utwente.nl/conference/15/paper/7267/determining-the-suitability-of-agile-methods-for-a-software-project.pdf
Van Gils, A. (2005). Management and governance in Dutch SMEs. *European Management Journal*, *23*(5), 583–589. https://doi.org/10.1016/j.emj.2005.09.013
Verdict Medical Devices. (2010). *Small Businesses have Big Impact on Device Market - Verdict Medical Devices*. https://www.medicaldevice-network.com/features/feature73715/
Wang, Y., Ramadani, J., & Wagner, S. (2017). An exploratory study on applying a scrum development process for safety-critical systems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10611 LNCS*, 324–340. https://doi.org/10.1007/978-3-319-69926-4_23
Wears, R. L. (2012). Rethinking healthcare as a safety-critical industry. *Work*, *41*(SUPPL.1), 4560–4563. https://doi.org/10.3233/WOR-2012-0037-4560
West, D. (2011). Water-Scrum-Fall Is the Reality of Agile for Most. *For Application Development & Delivery Professionals*, 2011–2012. http://www.storycology.com/uploads/1/1/4/9/11495720/water-scrum-fall.pdf
World Health Organization. (2010). *Medical Devices: Managing the Mismatch - An outcome of the Priority Medical Devices project*. World Health Organization.
Zühlke. (2021). *Discover Our Projects | Zühlke*. https://www.zuehlke.com/en/our-projects

Appendix B. Interview Questions

These questions were discussed during the one-on-one interviews with practitioners.

1. Could you tell me a bit more about your work? (professional experience, specialization)
2. How long is your experience with software development methods in safety-critical industries?
3. In case you are working with software development, which role do you take?
4. How familiar are you with agile software development methodologies in safety-critical industries?
    1. Hints: how long, how many projects (comparison)
5. How is your experience with SMEs using methodologies in safety-critical industries? (separate this between consultants consulting SMEs and employees at SMEs)
    1. Hints: years of experience, number of clients
6. According to your opinion, what are the most important requirements to start with agile development in a safety-critical environment?
    1. Functional or non-functional?
7. Which standards are relevant for safety-critical software development?
8. In your view, what are the challenges when starting to implement a software development method in the safety-critical industries?
    1. What did you experience?
9. What information do you think is needed to determine whether a software development method suffices?
10. In your opinion, how to deal with the challenge to derive reliable and consistent conclusions?
11. What suggestions from your practice would you provide to improve software development in safety-critical industries?
12. In your past experience, which pitfalls did you notice when implementing or using agile software development methods in safety-critical industries?
    1. Hint: which projects, how does it feel, how does it look, how to recognize it coming
13. Which methods are you familiar with in safety-critical software development? (do they know SafeScrum?)

Appendix C1. Sprint Review 1

Date: 02-09-2020 10:00
Attendees: Frank (Product Owner), Arthur (RAMS), Remco (Scrum Master)
Build and release was ran at the start of this meeting.

1. Review tasks with unresolved quality issues
During the sprint review, in user story 20 we found that the *scripts* folder is excluded from being linted. A new issue has been raised, and the scripts folder will be added to the linter settings.

2. Review or demonstrate resolved tasks from the sprint backlog
User Story #19 Setup bundler + ESlint integration
For the demo, a build action via GitHub Actions is ran. The tools ESLint and build are automatically ran. This results in a build folder that can be accessed with a static html server. Documentation is written in the wiki of the GitHub project: pages TypeScript, Git, GitHub Actions, npm, Prettier, Tailwind CSS.

A comment is made by Arthur that for the development setup, hot reloading is missing. Hot reloading in development is the practice of automatically regeneration of compiled code, so changes in code are reflected in the local testing environment. The developers in this team are used to a development experience where tasks such as building, linting, configuration are generally pre-configured by the framework they are developing with, React. Recreating this experience from the ground up is found cumbersome, since all configuration must be done manually. A preference is voiced for using *create-react-app*, the utility to automatically start a new development setup.

The product owner approves this user story.

User Story #20 Configure unit test framework
For the demo, a build action is ran via GitHub Actions. This results in a test folder, created by the tool jest. A report file is generated where the number of tests, the percentage of passing and failed tests and the coverage is displayed. Documentation can be found in the wiki at the page “Testing”. Upon checking, specific information about individual tests is missing in the report, possibly due to missing environment variables. This will be resolved in upcoming sprint.

A discussion about the scope of testing emerges. Certain parts of the code can be imitated, thereby intercepting the values that go through that part of the code. This practice is called “mocking”. When a part of the code is mocked, the initial function of that code is lost. Therefore, the decision is made to mock as little as possible and use the proper setup and teardown methods for a test: *beforeEach* and *afterEach* functions.

The product owner approves the user story.

Unresolved: User Story #18 Configure day of operation
This user story has been started on during this sprint but was unfinished. It will be brought back to the product backlog and planned for the next sprint.

Unresolved: User Story #8 Task list
This user story has been started on during this sprint but was unfinished. It will be brought back to the product backlog and planned for the next sprint.

Appendix C2. Sprint 1 Retrospective

Date: 02-09-2020 11:00 - 11:57
Attendees: Frank (Product Owner), Arthur (RAMS), Remco (Scrum Master)

What went well	• Using the wiki as location for documentation works well. It maintains a history log with every person, version and changes made. • GitHub Actions work well for quality assurance purposes. Static code analysis can be performed automatically after every push. • Avoidance of using dependencies is easier than expected. Is it necessary though? Dependencies help by external maintenance; configuration can be checked by developers.

Improvements	• It is unclear whether SOUP can be used within development.
	  ○ This is part of the research that has to be conducted.
	• Setup of the architecture is better done as a pre-sprint before starting development (sprint 0).
	  ○ In future development, sprint 0 will be added for setup.
	• Documentation in pull requests could be clearer. Now it is a list of comments starting from the original pull request, to comments for improvements, to acceptance.
	  ○ An automation could export this to a full document.

Appendix D1. Sprint 2 Review

Date: 23-09-2020 15:00
Attendees: Frank (Product Owner), Arthur (RAMS), Remco (Scrum Master)
Build and release was ran at the start of this meeting.

1. Review tasks with unresolved quality issues
During the sprint review, in user story 30 we found that unit tests fail. This is because the user story sets up a service worker. The service worker is excluded from unit tests and will be added to end-to-end testing, since it is part of the whole application. The product owner approves this story.

2. Review or demonstrate resolved tasks from the sprint backlog
User Story #32 Extra information in test report
In order to make the test reports in line with regulations, each test is reported on and the full test report shows a percentage of tests that succeeded and failed. A demo is ran.

The product owner approves this user story.

User Story #31 Router, settings, reset data
As a demo, the application is ran. The router allows for navigating through the application. Settings page can be accessed. Reset data functionality clears the data that the application stores.

The product owner approves the user story.

Unresolved: User Story #18 Configure day of operation
The code has not been reviewed, but only quality assurance has been performed on this user story. Therefore, it has been reverted and the code review will take place in the next sprint.

Unresolved: User Story #8 Care plan
This user story has been started on during this sprint but was unfinished. It will be brought back to the product backlog and planned for the next sprint.

Appendix D2. Sprint 2 Retrospective

Date: 23-09-2020 15:57
Attendees: Frank (Product Owner), Arthur (RAMS), Remco (Scrum Master)

What went well	• Testing using the automated process with GitHub Actions went well. Reports are well-structured. Testing-first approach leads to more acceptable code on first implementation, leaving less comments during the code review. • Quality assurance by testing and linting is performed automatically as well. • The development setup is simple, using React as framework and libraries supporting development.
Improvements	• A commit history is important to show traceability. Until now, commits were squashed before merging them, thereby losing the granularity. • A checklist or template for a code review would be convenient, to standardize this part. • Is it possible to combine quality assurance with code review, since quality assurance is automated through GitHub Actions?

Appendix E. Software Requirements

Functional requirements

- FR1. The application shall show the patient a list of daily tasks they need to perform.
- FR2. The application shall notify the patient on set times to perform tasks.
- FR3. The application shall allow the patient to register a performed task.
- FR4. The application shall allow the patient to register:
    - the amount of steps they took in one day;
    - the amount of glasses of water they drank;
    - the time they sat at the side of the bed.
- FR5. The application shall allow the patient to adjust wrongly entered information about their registered tasks.
- FR6. The application shall show the progress of the daily tasks to the patient.
- FR7. The application shall inform the patient when a daily task is completed.
- FR8. The application shall give the patient information about the tasks in the form of pre-recorded videos, text and/or audio.
- FR9. The application shall show a summary of daily tasks for a past period of time.
- FR10. The application shall show patients general information about things they can do to improve the recovery process.
- FR10. The application shall allow the patient to configure the date of surgery.

Appendix F. Defined Responsibilities

Except for the 5 meetings below, each task can be worked on asynchronously and the work to be done is a team effort.

User Stories
The product owner is responsible for the user stories. Most of the time the product owner will draft the user stories as they represent the wishes of all stakeholders.
Example:
>As a [stakeholder], I want to [achieve some goal], so I can [reason]
or
>As a [type of user], I want to [perform some task] so that I can [reach some goal]

Safety Stories
While user stories define “how” in the application, safety stories are about the “what”. They maintain safety in the system and enable discussion in the team, leading to a safety culture. The product owner is responsible for setting up these stories, while the RAMS engineer relates these to legislation and standards. When legislation or standards are unclear, the assessor should be involved.
Example:
>To satisfy [a safety standard requirement] the system must [achieve or avoid something]
or
>To keep [function] safe, the system must [achieve or avoid something]

Hazard Stories
Hazard stories are created by the product owner, after which they are discussed with the whole team. They are not meant to implement functionality, but support discussion and decisions.
Example:
>As a result of [cause][cause event] which will lead to [accident event] (if [accident condition])

Product Backlog
The product owner is responsible of maintaining the (product) backlog (list of user and safety stories). Every team member can edit the backlog, but the product owner is responsible for the prioritization of the user stories.
The RAMS engineer is responsible to ensure the system becomes a safe system and is in this sense responsible for safety validation of the stories. The RAMS engineer achieves this by relating the stories to applicable standards and creating safety stories for requirements from standards.

Sprint Backlog
The entire team is responsible for composing the sprint backlog during the sprint planning meeting. In the end the scrum master has to make sure this happens (in an orderly manner).

Task
The entire team is responsible for converting user stories into one or more tasks and determining the definition of done. Optionally, part of this can happen before the sprint planning meeting. It's good to prepare in advance (examples are wireframes, design artefacts or a concept technical plan). There are further elaborated during the sprint planning meeting, when a story is selected for implementation.

Sprint planning meeting (max. 1-3 hours)
The scrum master is responsible for planning, preparing and facilitating this meeting. The product owner is responsible for setting the goal of the upcoming sprint and a prioritized product backlog. The entire team is responsible for selecting the stories from the backlog for the upcoming sprint, which will become the sprint backlog.

Daily standup meeting (max. 15 minutes)
The scrum master is responsible for planning, preparing and facilitating this meeting. For each team member his/her responsibility is to update the team about the sprint progress and plan ahead together for the upcoming 24 hours. Safety must be on top of the agenda.

Sprint review (max. 1-3 hours)
The scrum master is responsible for planning, preparing and facilitating this meeting. The product owner is responsible for updating the product backlog. The entire team is responsible for sharing the final sprint results in the form of a demo. All team members are responsible for approving or

Sprint retrospective (max. 1-2 hours)
The scrum master is responsible for planning, preparing and facilitating this meeting. Every team member is responsible for sharing feedback and providing possible process improvements.

Appendix G. User Story Template

GitHub metadata
Name: Product backlog story
About: Product backlog is based on the system requirements specification. Add labels where applicable.
Title:
Labels: backlog
Assignees:

Template
Description: *description of functionality, related to requirements from the SRS*
SRS ref: *stable reference to SRS*
Type: *functional or safety*
Importance: *product owner: scale 1-10*
Tasks:

- *Optional: task breakdown. Should be conducted in a sprint planning meeting*

Risk: *high/medium/low, based on the risk analysis*
Complexity: *high/medium/low, how hard to implement*
Estimate: *ideal working hours*
Demo:

- *How to demonstrate story / for safety stories: assessment*

Definition of done:

- *What needs to be in place in order to consider story done. This includes V&V and documentation.*

Notes:
Any additional notes.