Meshing agile and plan-driven development in safety-critical software: A case study

Lise Tordrup Heeager
Department of Management
Aarhus University
Denmark
lith@mgmt.au.dk, https://orcid.org/0000-0003-4495-3084

Peter Axel Nielsen (Corresponding author)
Department of Computer Science
Aalborg University
Denmark
pan@cs.aau.dk, https://orcid.org/0000-0002-0282-7445

Abstract
Organizations developing safety-critical software are increasingly seeking to create better practices by meshing agile and plan-driven development processes. Significant differences between the agile and the plan-driven processes make meshing difficult, and very little empirical evidence on using agile processes for safety-critical software development exists. There are four areas of concern, in particular, for meshing the development of safety-critical software concerning: documentation, requirements, life cycle and testing. We report on a case study of a pharmaceutical organization in which a Scrum process was implemented to support agile software development in a plan-driven safety-critical project. The purpose was to answer the following research question: For safety-critical software, what can a software team do to mesh agile and plan-driven processes effectively? The main contribution of the paper is an elaborated understanding of meshing in the four areas of concern and how the conditions for safety-critical software influence them. We discuss how meshing within the four areas of concern is a contribution to existing research.

Keywords Agile software development · Plan-driven software development · Process · Safety-critical software · Meshing · Compliance · Scrum · FDA

1 Introduction

Safety-critical software products are increasing in numbers and complexity. They are found in several domains, and they affect the everyday life of people. A few examples of safety-critical products are digital insulin pumps, railway signalling, airbags, elevator controls, self-driving cars and flight controls (Abdelaziz et al. 2015; Rasmussen et al. 2009). Though the advantages of these devices are great, the safety risks involved cannot be ignored as failure may have severe consequences, such as loss of life. The safety risks have led to the institutionalization of approval procedures and certification, and as a consequence, the development of safety-critical products and software is highly regulated. For example, medical devices in the USA must be approved by the US Food and Drug Administration (FDA) (U. S. Department of Health 2010), and the EN 50128 is a European standard for the development of railway applications (Jonsson et al. 2012; Myklebust and Stålhane 2018). Similar agencies are authorized in other areas and other countries to approve and certify safety-critical products. To the untrained eye, there seems to be a generic waterfall model underlying these standards, leading to traditional plan-driven approaches to software development.

The plan-driven models do not work well in general software development when requirements change and when users, marketing executives, managers and developers inevitably acquire new knowledge during the stages of development. That is a significant shortcoming of the plan-driven model (Boehm and Turner 2005; Cockburn 2006; Kuhrman et al. 2017). Also, in the development of safety-critical software, changes are inevitable despite thorough analysis (Beznosov 2003), and requirements change or additional requirements emerge during development (Rasmussen et al. 2009). While several agile development processes have existed for some years (Cockburn 2006; Conboy 2009), a large number of software organizations have implemented agile processes such as Scrum (Schwaber and Beedle 2001). Within organizations which are developing safety-critical software, there is a marked interest in the agile processes, and several have implemented elements of the agile processes (McCaffery et al. 2016; Notander et al. 2013a).

Research indicates that agile processes can work well under regulatory requirements (McHugh et al. 2014b) and that safety-critical software can be developed using an agile process, (e.g. Abdelaziz et al. 2015; Van Schooenderwoert and Shoemaker 2018). The research literature shows that organizations use a variety of elements of agile processes while developing safety-critical software (Heeager and Nielsen, 2018), and surveys show how several organizations have experimented with agile development of safety-critical software (McCaffery et al. 2016; Notander et al. 2013a). However, to fulfil the regulatory requirements while using agile processes, the development processes need to be tailored for this specific purpose (e.g. Fitzgerald et al. 2013; Myklebust and Stålhane 2018).

In this article, we address a particular case company, and we document what a software team has done to mesh agile and plan-driven development processes. Here, we have taken *meshing* as the common term for mixing, integrating and coexisting. The case study software team developed software for a safety-critical medical device while being embedded within a much larger device project that was highly plan-driven. The product, as a whole, was required to be compliant with the FDA standards, and the case demonstrates both the difficulty with some aspects of meshing and the ease of other aspects.

The objective of the research was to provide empirical evidence and to analyse the case data on the following research question:

For safety-critical software, what can a software team do to mesh agile and plan-driven processes effectively?

In Section 2, the related research on meshing in safety-critical software development is presented, and the areas of concern in which the agile and the plan-driven processes need to mesh are discussed. The research design of the case study is described in Section 3. The case organization is presented and the software team's practices are described in Section 4. The case study analysis is presented in Section 5, supported by empirical evidence from qualitative interviews. The contributions derived from the findings are discussed in Section 6, and the conclusions are described in Section 7.

2 Agility in Safety-Critical Software Development

In this paper, we use the following definition of agility: “the continual readiness of an ISD [information systems development] method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment” (Conboy 2009, p. 340). Embracing change and customer value is central to the agile processes; any attempt to apply agile processes when developing safety-critical software must accommodate this, one way or the other.

It is often required in standards and approval procedures that development follows a documented and rigid process (Heeager and Nielsen 2018), and that this occurs at the outset is mostly in line with plan-driven software development; thus, for a long time, agility was considered at odds with safety-critical software that can cause serious injury to people, the environment or the economy. Several studies have engaged in the question of whether agile processes can be used when developing safety-critical software; the majority are studies comparing an agile process, such as Scrum or XP, with a certain regulatory standard (e.g. Beznosov and Kruchten 2004; Mehrfard et al. 2010; Wäyrynen et al. 2004; Özcan-Top and McCaffery 2019), but some scattered empirical studies also do exist (e.g. Bedoll 2003; Fitzgerald et al. 2013; Grenning 2001; Rottier and Rodrigues 2008). The literature shows that agile processes in their original form do not comply with regulatory requirements for safety-critical software development, and thus adaptations to the agile processes are needed. In line with this, some papers have suggested modified agile processes, for example, Stålhane et al. (2012) who proposed the Safe Scrum, (Boström et al. (2006) who proposed an extended XP and Hajou et al. (2015b) who proposed a method, called æ, which was tailored for the pharmaceutical industry. However, more empirical evidence of the challenges and the approach that can be taken is needed.

In practice, compliance between agile development and safety-critical development can be done by meshing; meshing will conceptually entail that the agile and plan-driven ideas and processes are mixed when they can coexist, balanced against each other when one is dominating the other and merged when they can integrate seamlessly. The mixing, balancing and merging of software processes, parts of processes and process components we refer to as *meshing*. A prominent example of meshing is the work of Boehm and Turner (2005) in which they develop and describe several dimensions all of which are important for software development in general.

In a recent systematic literature review, of 51 articles published in leading journals and conferences, four areas of concern for agile processes when developing safety-critical software were found (Heeager and Nielsen 2018):

- Light documentation: While the agile processes depend on face-to-face communication and informal coordination over documentation of knowledge, safety-critical software development relies on formalized processes based on documentation.
- Flexible requirements in user stories: Agile processes advocate changeability in requirements and a less formal specification. In safety-critical development, changes pose challenges to the documentation, and the documentation of the requirements must be done formally.
- Iterative and incremental life cycle: Agile processes suggest an iterative life cycle to facilitate learning and adoption. In safety-critical software, development stability is prioritized, and the development follows a sequential life cycle.
- Test-first processes: Agile processes rely on test-driven development and iterative automated testing. In safety-critical development, the testing is done as the final phases of the project, and instead of test-driven development, detailed test plans are conducted.

These are the four areas of concern within which we investigated the meshing of agile and plan-driven processes. For each area of concern, we added the method, the key experience and the recommendations of Van Schooenderwoert and Shoemaker (2018), who have applied agile principles to developing medical devices. Table 1 summarizes the challenges within each area of concern presented in the literature, as well as the suggestions for meshing the two processes.

Table 1 Challenges and meshing approach for each area of concern

Area of concern	Challenges	Meshing approach
The use of documentation	1. Providing the appropriate amount and the right kind of documentation for proving the safety of the software (Gary et al. 2011; Hajou et al. 2015a; Sidky and Arthur 2007; Wäyrynen et al. 2004) 2. Handling change management and maintaining traceability (Drobka et al. 2004; Kasauli et al. 2018; McHugh et al. 2014a; McHugh et al. 2014b; Notander et al. 2013a; Rottier and Rodrigues 2008; Sidky and Arthur 2007)	• Treat documentation as part of the product requested by the customer (McHugh et al. 2012) • Consider the purpose of the documentation and only provide what is required (Grenning 2001; Misra et al. 2010) • Produce the documentation before development, but cumulatively (Van Schooenderwoert and Shoemaker 2018) • Treat the documentation as a separate activity during each iteration (Stålhane et al. 2012) • Use sub-teams in charge of documentation (Paige et al. 2008)
Engineering requirements	1. Achieving flexible requirements and maintaining traceability (Bedoll 2003; Boström et al. 2006; Drobka et al. 2004; Ge et al. 2010; Notander et al. 2013a; Notander et al. 2013b; Rasmussen et al. 2009) 2. Documenting the requirements in user stories (Górski and Łukasiewicz 2013; Jonsson et al. 2012)	• Separate functional and safety requirements (in the beginning) (Ge et al. 2010; Stålhane et al. 2012; Van Schooenderwoert and Shoemaker 2018) • User stories should supplement the formal requirements (two types) (Boström et al. 2006; Demissie et al. 2016) • Modify user stories to include risks (Hajou et al. 2015b; Kasauli et al. 2018) • Use safety stories (Myklebust and Stålhane 2018)
Life cycle	1. Managing iterative changes (Beznosov 2003; Beznosov and Kruchten 2004; Jonsson et al. 2012; Notander et al. 2013a; Van Schooenderwoert and Shoemaker 2018) 2. Iterative validation of the software (Kasauli et al. 2018; Paige et al. 2008; Wang et al. 2017; Wang and Wagner 2018)	• Conduct the safety analysis iteratively, but validate the software incrementally (Kasauli et al. 2018) • Treat change management in documents iteratively and incrementally (Notander et al. 2013a; Van Schooenderwoert and Shoemaker 2018) • Use new validation techniques such as System-Theoretic Process Analysis (STPA) or a combination of STPA and Behavior-Driven Development (BDD) (Wang et al. 2017; Wang and Wagner 2018) • Use stepwise integration instead of continuous integration (Myklebust and Stålhane 2018)
Testing	1. Changing the practice from testing in the final stages to iterative testing (McCaffery et al. 2016; Rottier and Rodrigues 2008) 2. Roles related to testing conflicts in agile vs safety regulations (Jonsson et al. 2012) 3. Heavy iterative testing of the software increments (Kasauli et al. 2018; Rottier and Rodrigues 2008)	• Use longer iterations to include testing and validation (Rasmussen et al. 2009; Rottier and Rodrigues 2008) • Use automated, risk-based, continuous testing (Kasauli et al. 2018)

2.1 The Use of Documentation

The use of documentation of safety-critical software gets much attention in the research literature as it is considered a primary challenge for applying agile development (Heeager and Nielsen, 2018). Documentation is essential for providing what is called the *safety argument* in the development of safety-critical software, and agile processes do not provide enough documentation or provide the wrong kind of documentation for this purpose (Hajou et al. 2015a; Sidky and Arthur 2007; Wäyrynen et al. 2004). Because of the flexible nature of agile processes, it has been concluded however that the amount of documentation in safety-critical projects is not in itself an issue (e.g. Gary et al. 2011). Whatever is requested by the customer gets delivered by agile processes, and that can include documentation to prove the safety of the software (McHugh et al. 2012). In safety-critical development, the documentation is key in stakeholder communication, and it is thus important that the right documentation is produced for this purpose (Kasauli et al. 2018). The focus on documentation can however harm the agility and flexibility of the development (Jonsson et al. 2012), and the regulatory bodies are not likely to agree to less documentation on software requirements and design (Vogel 2006). While Hajou et al. (2015a) stated that the lack of documentation can be harmful to the quality of the software, Shafiq and Minhas (2014), on the other hand, suggested that agile processes might even result in higher quality software than the plan-driven processes, despite the lower focus on documentation. However, it remains open-ended in the existing research literature as to how flexible processes and the rigor of documentation can be reconciled. In the practice-based literature, there are recommendations on how to include safety documentation, for example, by producing documentation that is 'complete, consistent, and unambiguous' which also includes hazards and mitigations (Van Schooenderwoert and Shoemaker 2018, p. 21), though at the same time referring to the agile value of 'working software over comprehensive documentation'.

Furthermore, a few papers give only scattered examples of how practices have met the requirements for documentation. Grenning (2001) and Misra et al. (2010) suggested that it is important to consider the purpose of the documentation and determine which parts of the process must be documented. Stålhane et al. (2012) proposed a Safe Scrum method in which the documentation is treated as a separate activity in each iteration. In line with this, Paige et al. (2008) recommended using sub-teams in charge of documentation and suggested that tools are a way to support the activity.

In the development of safety-critical software, the software requirements must be documented before implementation and testing (McHugh et al. 2014a). Writing and updating the documentation of requirements, iteratively and incrementally, is a concern for an agile process. When documentation and requirements keep changing, it becomes increasingly challenging to ensure traceability amongst the different parts of the documentation in all stages of development as mandated by safety standards (Kasauli et al. 2018; McHugh et al. 2012; McHugh et al. 2014a; Notander et al. 2013a; Rottier and Rodrigues 2008). As safety-critical development is most often a very long (up to 30 years) process, changes are bound to happen and need to be handled (Drobka et al. 2004; Sidky and Arthur 2007).

By referring to an FDA standard, Van Schooenderwoert and Shoemaker (2018, p. 57) recommended that, for agile development of medical devices, the standard is open to the various forms of documentation. The documentation must, however, be produced before the software is developed, though the documentation can be produced cumulatively.

2.2 Engineering Requirements

For *requirements*, the agile and the plan-driven processes of safety-critical development differ in two ways, which pose a challenge when meshing.

Firstly, agile processes are open to continuous change of the requirements, whereas in safety-critical software development, changing the requirements is discouraged (Notander et al. 2013a; Notander et al. 2013b). The changes in the requirements can have severe consequences on the software architecture and may invalidate the safety argument (Drobka et al. 2004; Ge et al. 2010), which then need to be changed accordingly. Dealing with changing requirements in software development is generally a necessity (Bedoll 2003; Beznosov 2003; Lee and

Xia 2010), and as safety-critical development is often a very long-term activity, changes will happen over time as the project progresses (Rasmussen 2009). On the other hand, some studies have shown that changes to requirements may be less frequent in safety-critical software development, compared to the development of less critical software (Ge et al. 2010). The functional requirements may well change considerably over time, whereas the safety requirements remain quite stable. Thus, it is recommended to divide the product backlog into two: functional requirements (allowed to change more frequently) and safety requirements (kept as stable as possible) (Myklebust and Stålhane 2018; Stålhane et al. 2012). Others recommend that only functional requirements and hazards analysis are kept separate at the very beginning of a project, but then are eventually integrated later in the project (Van Schooenderwoert and Shoemaker 2018).

Secondly, agile methods generally rely on loosely structured requirements, such as user stories, and that conflicts, according to Górski and Łukasiewicz (2013), with regulatory requirements for safety-critical development. User stories written in plain business-like language, as suggested by the agile processes, cannot be used for validation as the regulatory standards for the development of safety-critical software require well-structured requirement engineering (Jonsson et al. 2012). The literature offers different strategies for solving this issue. Demissie et al. (2016) recommended using user stories as a supplement to the formal requirements, in the development of safety-critical software, as agile user stories provoke a discussion between the developers and the customer. Boström et al. (2006) advocated using two types of stories, one type for the safety requirements and another type for the functional requirements. Hajou et al. (2015b) and Kasauli et al. (2018) suggested combining the traditional user stories with risk assessments to fulfil the regulatory requirements. Myklebust and Stålhane (2016) suggested using what they call safety stories, which are user stories that include one or more safety requirements. These stories follow a pattern close to the pattern of user stories, but include safety requirements. Van Schooenderwoert and Shoemaker (2018), on the other hand, based on their experience, recommended that stories are quite adequate to capture the important aspects of requirements.

2.3 Life cycle

Learning and adapting are essential aspects of agile processes, which are supported by an iterative and incremental *project life cycle*, whereas in safety-critical software development, the project is structured in sequential phases, and the V-model is favoured by many (McHugh et al. 2014b). The V-model is a variation of the waterfall model, with a particular focus on testing and quality management (Hajou et al. 2015b), and it does not advocate iterations and increments (Ge et al. 2010). The V-model is often an idealized explanation for developing safety-critical software, as it produces the necessary quality documentation for regulatory approval (McHugh, McCaffery, and Coady 2014). Yet, a specific development life cycle is not prescribed by the regulatory process standards (Lin and Fan 2009; McHugh et al. 2012; McHugh et al. 2014b; Rottier and Rodrigues 2008). However, the regulations present their requirements according to the waterfall model or the V-model and require that, if a different life cycle model is followed, an argument of how this life cycle fulfils the requirements must be provided (Myklebust and Stålhane 2018).

Examples of how iterative safety-critical development can be adopted have been shown in various studies (Abdelaziz et al. 2015; Heeager 2012; Notander et al. 2013a). However, studies have also shown how iterative documentation and validation of an increment is challenging (Beznosov and Kruchten 2004; Jonsson et al. 2012; Paige et al. 2008). Beznosov (2003) concluded that using iterations in safety-critical development is similar to using iterations in non-safety-critical development, but more difficult. A main concern is that the documentation continuously needs to be created and updated; thus, change management is a significant concern with iterative development processes. Changes are introduced in most iterations, which may invalidate previous work on ensuring safety, when developing systems incrementally (Jonsson et al. 2012). It is also difficult to evaluate the quality of safety arguments when working iteratively; therefore, it is advised to conduct the safety analysis iteratively and validate the safety incrementally (Kasauli et al. 2018). Incorporating iterative verification techniques is challenging, as these activities are work intensive and reduce the use of agile software development (Paige et al. 2008). Myklebust and Stålhane (2018) suggested using stepwise integration instead of continuous integration. Notander et al. (2013a) and Van Schooenderwoert and Shoemaker (2018) suggested an iterative framework for safety-critical software development in which documentation is also treated iteratively and incrementally. Wang et al. (2017) and Wang and Wagner (2018) stated how traditional safety analysis and verification techniques are difficult to use with agile software development. Wang et al. (2017) proposed a process called S-Scrum, which uses a hazard analysis technique called System-Theoretic Process Analysis (STPA) and allows for an iterative validation, and Wang and Wagner (2018) proposed using a combination of STPA and Behavior-Driven Development (BDD).

2.4 Testing

Extensive testing is considered necessary by both the agile and the safety-critical processes (Górski and Łukasiewicz 2012). According to Kasauli et al. (2018), using agile processes results in better test cases, which improves the quality of the software. However, the testing strategies of the processes differ, and the reconciliation of the safety-critical plan-driven testing strategy with the agile strategy is not straightforward. Following the V-model, the testing is done in the final phases of the development, whereas in agile processes, testing is part of each iteration (McCaffery et al. 2016). Adopting iterative testing is difficult due to the high level of required validation and verification, quality of documentation and thorough testing of increments, resulting in heavy iterative testing. The literature, therefore, has suggested that longer iterations are necessary when developing safety-critical software (Rottier and Rodrigues 2008), while Kasauli et al. (2018) have proposed automated, risk-based continuous testing (including safety tests).

Some agile evangelists have advocated test-first processes as stemming from XP; yet, some have reported that it easily clashes when developing safety-critical software (Heeager and Nielsen 2009). In test-driven development, developers write the tests themselves, and this proves problematic, as some standards (such as EN 50128) require that the tester must be responsible for specifying the test and that the developer and tester must be separate persons (Jonsson et al. 2012).

A comparative analysis has shown that the test-driven development is compatible with various regulatory standards (Paige et al. 2008). There are, furthermore, empirical examples of safety-critical software development in which test-first processes have been implemented successfully (Drobka et al. 2004; Grenning 2001; Spence 2005; VanderLeest and Buter 2009). According to Gorski and Łukasiewicz (2013), this however requires adaptations to the agile testing processes.

Within the safety-critical domain, the software is often embedded in a device. The development of the hardware and mechanics of this device cannot be built incrementally in the same way or to the same degree as the software. Moreover, it is a challenge to carry out incremental testing due to dependencies on dedicated hardware (Paige et al. 2005; Wils et al. 2006).

3 Case Study Approach

The study follows the guidelines for conducting a case study in software engineering by Runeson and Höst (2009); thus, this methodological section is divided into three subsections: case study design and planning, data collection and data analysis.

3.1 Case Study Design and Planning

The purpose of this study was descriptive (Runeson and Höst 2009); thus, the purpose was to portray the situation as it unfolded (Gregor, 2006) and provide a deeper understanding of what a software team can do to mesh agile and plan-driven processes when developing safety-critical software. A case study approach was chosen, as this method is suited for understanding the complex real-world setting of developing software while meshing the agile and plan-driven processes. Case study research is highly accepted both in information systems research (Walsham 2006) and software engineering research (Runeson and Höst 2009). We used a holistic case study (Yin 2009) in which a large pharmaceutical company in Denmark was the case organization, and the unit of analysis was the practice of a software team developing software for an embedded safety-critical device. Thus, we focused on the meshed software practice in the context of a large, traditional pharmaceutical organization.

3.2 Data Collection

The case study was based on qualitative data gathered over almost two years by using different data collection techniques and by involving multiple sources (Runeson and Höst 2009). Collecting data over a longer period through multiple iterations allowed time to elaborate on the findings before collecting more data. By using three different data collection techniques (interviews, observations and archival study), we sought data source triangulation and gained both first-, second- and third-degree insights into the software practice (Runeson and Höst 2009). By involving multiple data sources with different roles (managers, developers and testers), we sought different interpretations of the phenomenon (Runeson and Höst 2009).

The data were collected in the final stage of the development project in the case organization, but included an historical view of the process. The time of study was appropriate, as the software team, at that point, had gained sufficient experience with using agile software processes for the development of the medical device.

The data collection was divided into two phases (see Table 2). The purpose of the first phase was to map the current software practice, as well as the interviewee's evaluation of the strengths and the weaknesses of the practice. The first phase was concluded with a seminar presentation with the purpose of validating the findings. The second phase took place almost a year after the first phase; over that intervening period, the case organization had reflected on the findings of the first phase and had changed and improved the practice accordingly. Therefore, we found it relevant and insightful to conduct another round of data collection. Within each phase, the interviews were done in two iterations, approximately two months apart.

Table 2 Overview of the data collection

Phase	Focus	Data	#	Participants
1	Initial mapping and evaluation of the software practice	Seven interviews	1	The process manager
			2	A software architect
			3	A software tester
			4–7	Four software developers
		Eight interviews	8	The process manager
			9	The software architect
			10	A software tester
			11–15	Five software developers
		Observations of practice		
		Archival study		
		Seminar presentation and validation		
2	Mapping and evaluation of changes and improvements of the software practice	Seven interviews	16	The product owner proxy
			17	A project group coordinator
			18	A Scrum consultant
			19	A software tester
			20–22	Three software developers
		Five interviews	23	A process manager
			24	A software architect
			25	A software tester
			26–27	Two software developers
		Observations of meetings		
		Archival data		
		Report and validation		

Interviews are an important data collection technique in case studies (Runeson and Höst 2009) and were the primary data source; we collected a total of 27 qualitative interviews. In the first phase, a total of 15 qualitative interviews were conducted with the managers, developers and testers. In the second phase, 12 interviews were conducted with the managers, software developers, testers and a consultant specializing in Scrum who had been affiliated with the software team for six months. The interviews were all based on semi-structured qualitative interview guides with open questions inviting a broad range of answers (the interview guides are included in Appendix A). In the first phase, the questions focused on mapping the current practice and identifying both agile and plan-driven elements. The focus of the second phase was on the changes and improvements that had been incorporated in the software practice between the first and second phases. Approximately half of the interviewees participated both in the first phase and in the second phase. All interviews were audio-recorded and transcribed.

Several observations of the software practice were conducted to provide details of how meetings were performed (e.g. the daily Scrum meetings and the planning meetings), and to provide a richer, first-hand view. The observations had a low degree of interaction by the researcher and a low awareness of being observed by the participants (Runeson and Höst 2009). The observations were documented by note taking in structured forms developed based on the purpose of the observation. The notes were supplemented with photos when allowed.

The data collection was also supported by archival data, for example, a software development handbook created by the organization and examples of documents provided for the FDA. The purpose of the archival data was to gain a deeper insight into documents supporting the practice.

3.3 Data Analysis

After each round of data collection, the data were analysed to create an initial understanding of the software practice. The data analysis used open coding to identify, respectively, the agile and the plan-driven elements of the software practice, as well as the challenges and advantages. This analysis resulted in mappings of how the practice was conducted at the two points of data collection. At the end of each phase, a report was prepared and used for validation.

In the final analysis, a more thorough analysis was conducted; the qualitative data from both phases were analysed and coded in Nvivo. The coding was based on the four areas of concern from the literature, which were created as predefined codes. All quotes relating to one of these areas were identified and assigned to the code. If a quote mentioned two areas, the quote was assigned to its primary area. Simultaneously, the data were coded using open coding; this was done both to open up the possibility of other areas emerging and to have the codes describe the quotes more carefully. No other areas emerged, so all open codes and their underlying quotes could be assigned to one of the predefined codes. Based on the open codes, two to three categories emerged within each of the four areas. The process is visualized in Table 3 with all open codes and categories.

Table 3 Focus through areas of concern, open codes and categories of open codes

Areas of concern	**Open codes**	**Categories of codes**
Documentation	Plan focuses on documentation Early documentation Focus on documentation	Compliance documentation
	Difficult to create flexibility in documentation Strict process of changing documentation Depending on requirements at higher levels	Fast and slow changeability
Requirements	Difficulties understanding requirements Problems with details in requirements Problems with requirements in general	Requirements uncertainty
	Rewriting of requirements is necessary Difficult to convince management of rewriting New requirements are improved	Requirements changeability
Life cycle	Reluctance towards Scrum and iterations Sprints are advantageous Iteration length	Changing practice
	Stopped using iterations	Milestone alignment
	Interruptions of iterations Slices do not fit	Hardware, software and mechanics co-development
Testing	Developers lacking testing skills Strict testing procedures Use specialized roles with testers and developers Reluctance towards test-driven development	Practice and competence
	Prioritizes new functionality over tests No time for test in iterations Unit test is often postponed	Management priorities and progress

4 Case Description: The Software Team and Its Practice

This case study concerned a sizeable pharmaceutical company within which a project, involving the development of a safety-critical clinical device with embedded software, took place. This project was the first and, by far, the largest and most complicated software project in the company. Due to a high level of safety-criticality, the medical device and development hereof had to be compliant with a large number of regulations, specifically those related to regulatory standards of the US FDA. The project consisted of several project groups, each with its area of focus, such as mechanics, hardware and software. In total, over 100 managers, engineers and developers were working on this project. The unit of analysis was the meshed software practice, that is, what the software team had done to implement elements from the agile method Scrum, while still ensuring compliance with the FDA and fitting in to a larger project that was genuinely plan-driven. Fig. 1. depicts this relationship between the organization serving as the context of the project in charge of developing the medical device and how the software team was (one) part of this project. Everything outside the software team was treated as context.

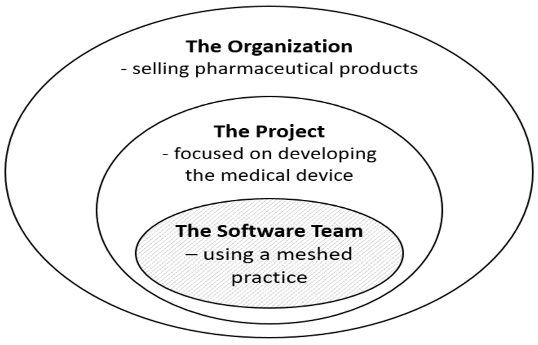
The Organization
- selling pharmaceutical products
The Project
- focused on developing the medical device
The Software Team
– using a meshed practice

Fig. 1 The unit of analysis and its context

At the time of the study, the project had run for several years and had come into the last stages of refining the product. The project followed a linear V-model consisting of phases and milestones. This model was widely used in the organization and was deeply rooted in its practice of developing new medical products. The milestones at the end of each phase were used to control and evaluate the progress of the project. To reach a milestone and move to the next phase, each project group had to provide documentation proving that its goal had been reached. During the time of data collection, the developers often talked about how to achieve the next milestone, how the project was being held back, and which project group was to blame for the stagnation.

The requirements for the product were, to a large degree, formed at the beginning of the project and were mostly based on a user survey. In addition, a hazard analysis was performed (as required by regulatory standards) to highlight the risks for the users of the device. The identified hazards resulted in requirements for the device and the software to mitigate the risks. The requirements were frozen and written into different documents, such as the device specification and the hazard analysis. The project and the development project thus relied on several documents arranged in a strict hierarchy. The documents placed at the higher levels had been through a strict and time-consuming process to be approved by several stakeholders at the beginning of the project. These documents did not change or became difficult to change. In particular, the hazard analysis was placed at the higher levels. The documents at the lower levels were easier to change as a less strict approval process was required. The detailed design of the software was, for example, documented at the lower levels. The software requirements specification was placed at the middle of the hierarchy.

The company had had a long and successful history of developing medical devices, but at that point had very limited experience in the area of software development. Most software processes for this particular project had to be established as part of the project. In the initial stage of the project, the software team experienced a high degree of complexity and uncertainty in the tasks and requirements. To overcome these challenges, the software team decided to implement the agile method, Scrum, and attempt to achieve a mesh between plan-driven processes and agile processes. This decision was made by the software team (primarily by the software team manager) and did not involve other teams in the project. Scrum was chosen as it was expected to be simple and easy to implement, and it was expected to support the software team with coping adaptively with the project's uncertainties.

During the time of data collection, the software team expanded drastically to 30 developers, testers, and managers. The software team consisted of the software project manager, a software architect, the process manager, a coordinator responsible for the coordination with the larger system engineering project, developers and testers. The software developers who were hired during this period had a strong software background and experience in agile software development, and the developers who were initially on the team had a traditional background in engineering. The management of the software team acknowledged that the original developers were less competent in agile processes. Moreover, several skilled coaches and educators were involved in educating the developers; for example, a consultant facilitating the Scrum practice was associated with the software team for six months. In this way, there was a planned compensation for the relative inexperience of the original developers.

The software team faced several problems adapting to Scrum due to internal issues within the software team and external issues caused by the software team's context. However, the software team did implement elements of

Scrum and moved towards a much more agile software development, that is, a mesh of agile and plan-driven processes.

The developers sought to plan their development tasks in iterations containing programming, unit testing and internal delivery of the increments, as prescribed by Scrum. The software team experimented with iteration length, but in the end, they found two weeks most fitting for them. The iterations were initiated with a planning meeting where a sprint backlog was created, and the tasks included were estimated. The software requirements specification served as the software backlog; however, tasks were neither created nor prioritized by customers, neither in the sense of potential patients (users of the medical device) nor in the sense of a customer proxy from within the project. The tasks chosen for the sprint were written on post-it notes and placed on the Scrum task board for visualizing the status of the iteration. The task board was supplemented with burndown charts visualizing the progress of the iteration. Daily stand-up meetings were used for coordination amongst the developers within the iteration. In that way, the developers relied very little on written documentation for knowledge sharing and coordination within the software team.

Several tasks were not always completed by the end of an iteration. First, the developers tended to underestimate the tasks, mainly because they experienced problems breaking them down to a manageable size. Second, many of the developers were specialists and were not able to solve every task on the board on their own. When this issue was not taken into sufficient consideration during planning, the tasks had to wait for a few developers to address them, and therefore they became bottlenecks. Third, for security reasons, all tasks were to be peer-reviewed with a colleague, and the developers were required to have 100% condition-decision coverage on unit tests before completion. Several tasks were not reviewed and tested by the end of an iteration, as the developers had to direct more attention to implementing new functionality than to finalizing the current task.

The developers were in charge of creating the unit test cases. The unit tests ran automatically every day to ensure that errors were not introduced into the software. At the end of an iteration, the software increments were integrated into the existing configuration, and after the integration test, the increments were sent to the system engineering group system-level testing. When errors occurred at this point, these were formally reported back to the software team, and a formal procedure for correcting and documenting the correction of the error had to be followed. Not all software increments were properly unit tested before being integrated, and several errors were found during the system test, creating a slow testing-debugging-correction process. Each iteration was concluded with an evaluation of both the process and the current increments.

5 Results

The case study analysis was based on the four areas of concern identified in the literature (cf. Section 2) and showed not only how the software team moved towards meshed development practices but also how the meshed practices were primarily plan-driven due to the regulatory requirements and to the large systems engineering project of which the software team was only one part.

The analysis of the case data showed that the software team had found it hardest to mesh within the areas of documentation and requirements. Within the area of *documentation,* the developers struggled with an overhead of documentation only written to prove compliance and with time-consuming procedures when changes were necessary. Uncertainty of *requirements* arose mainly due to the lack of customer contact (which is not unusual for safety-critical products) and due to slow the changeability of requirements. The software team found it difficult to implement an agile *testing* practice. The developers lacked the needed competences and the management lacked in prioritising testing. The software team was able to achieve a mesh within the area of *life cycle* by embedding an iterative software practice within the overall project that was linear and strictly plan-driven. Changing from a linear practice to an iterative one, however, met with resistance, and challenges in alignment with the linear project model and other project groups occurred.

Table 4 provides an overview of the results in terms of the categories identified in the case study and a short empirically based explanation of each category. The detailed results are described in the following subsections. We took the results as descriptions of meshed practices for developing safety-critical software in particular.

Table 4 The identified categories of meshed practices

Areas of concern	Categories	Explanations from practice
Documentation	Compliance documentation	Much documentation is needed and only produced to prove compliance.
	Fast and slow changeability	While some documents can change fast and often, others require a long, strict procedure. In addition, most documents were written early on in the process.
Requirements	Requirement uncertainty	Requirements uncertainty is higher at the beginning of the project, at the point of which the requirements were specified. Uncertainties about requirements are a greater issue when no immediate customer is present.
	Requirements changeability	The requirements hierarchy lowered the flexibility of the requirements.
Life cycle	Changing practice	Many developers were reluctant to change their practice.
	Milestone alignment	Safety-critical projects are often controlled by milestones; developing embedded software in iterations require alignment with these.
	Hardware, software and mechanics co-development	With an embedded device, coordination with hardware and mechanics is required.
Testing	Practice and competence	Changing the testing practice from plan-driven to agile requires different skills.
	Management priorities and progress	Implementing agile testing requires prioritizing by management.

5.1 The Use of Documentation

The software team was less able to mesh their use of documentation than anticipated and remained plan-driven within this area. The software team did adopt an agile strategy for daily coordination within the software team. However, a large number of documents were written for the sole purpose of complying with the FDA standard. Handling the documents in a light and iterative manner which could better accommodate change proved highly difficult for the software team, and according to the project and software managers, this was due to the strict approval process for documentation. Two categories of meshed documentation in safety-critical development emerged: compliance documentation and fast and slow changeability.

5.1.1 Compliance Documentation

The project had to provide a substantial amount of documentation to show compliance with the FDA standard for safety-critical devices, as the understanding was that all aspects of the product had to be documented. The design, the code, the code reviews, the safety hazard, and the strategies for mitigating the hazards were all documented. Thus, the software team spent much time writing documentation and having the documentation approved. Most of this was written at the beginning of the project (e.g., the requirement specifications). Some of the documentation had already been written and approved before the software development started.

The analysis showed how most of the documents were written with the sole purpose of validating the safety of the software product. The documents were not used for knowledge sharing and coordination within and amongst teams. The documentation was read by only a few people. In its place, the software team used face-to-face communication as their primary way to coordinate and share knowledge, as suggested by the agile processes. The daily Scrum meeting and the planning meeting at the beginning of each sprint provided ample opportunities for the software team.

> *'I think they [stand-up meetings] are great. That is where we, well, normally at a stand-up meeting in the morning you briefly tell what did I do yesterday, what am I going to do today and what is blocking me right now. That is not exactly how it works here, we are discussing more at our stand-up meetings, but that shows that we work very closely together, so we need much planning to make the day work'. (Interview 16)*

To further support knowledge sharing, the software team additionally used several 'information radiators' (an agile tactic to pass along information quickly and quietly), for example, Scrum boards displaying the status and progress of the sprint. These Scrum boards provided an opportunity for instantly receiving relevant information with minimum effort. This knowledge sharing tactic was perceived as being useful in practice by the software developers, whereas much documentation was not found useful despite the safety-critical issues addressed in the documentation. The safety-critical issues in need of coordination and knowledge sharing were addressed through the light-weight Scrum meetings, boards, and a few ad hoc meetings where developers discussed their problem-solving after the morning meeting.

This way of adapting to the requirements of the FDA thus required the software team to spend a great deal of time producing documentation required by the overall project that did not support or improve their software practice. On the other hand, they found a way to coordinate in a more agile manner within the team.

5.1.2 Fast and Slow Changeability

Due to the strict documentation hierarchy, the changeability of most documents was very slow and some were even almost impossible to change. The software team especially struggled with slow changeability of the software requirements specification. This problem was exacerbated by the fact that several documents had been written in the early stages of the project, both due to requirements by the FDA standards that certain documents were to be produced before the product was developed (e.g. a safety hazard analysis) but also due to the overall project model which focused on milestones and written evidence for progress.

The V-model is an often used as a tactic in safety-critical projects (though not a requirement), and the company having had a long history in the pharmaceutical industry had established and adapted a V-model to be used across all its projects. A major part of this model is the focus on delivering the documents needed for the final regulative approval of the product. The milestones in the V-model thus forced the software team to focus on writing documentation and describing the software up front to progress through the phases. The software team was therefore limited in developing the software and the documents iteratively.

> *'. . . due to our previous milestone, suddenly we had to have a whole bunch of documents ready, which did affect the progress of the product, to have these documents ready'. (Interview 24)*

The software team often found a need to change and update some of the documents, but also found the process of changing these documents extremely difficult. The software team sought a much more flexible process in handling the documentation to fit better with its more flexible way of working.

> *'The main challenge is to introduce flexibility in the artefacts [i.e. documents], that we have to produce.' (Interview 23)*

To improve flexibility, the software team dealt with lower-level documentation iteratively by writing this in increments, improving the low-level specifications as new information came up or experience was gained. They were only partly successful in doing so because of the management focus and perceived regulatory requirements for full upfront documentation, as described by the project model. The tactic of treating the low-level specifications as output from sprints enabled a faster changeability and a smoother mesh with agile development. However, changing the high-level documents was much more complex and time-consuming and led to slow changeability, and the key explanation for this in this case was that the slow changeability was caused by the regulatory requirements.

5.2 Requirements Engineering

The analysis identified two main categories within the area of requirements engineering: requirements uncertainty and requirements changeability. Requirements uncertainty arose due to poorly written requirements specifications and because no customer was affiliated. This was partly connected to changeability, as requirements were then bound to change.

5.2.1 Requirements Uncertainty

Due to the safety-critical regulations, most requirements were gathered and documented early. The early requirements were superficially described and, at times, difficult to understand. The high-level requirements were based on a thorough analysis. The requirements were uneven in detail and specificity, and later some were found to be too specific or even conflicting. According to the process manager, these issues occurred because the marketing group responsible for the initial requirements was not specialized in software requirements and because the requirements had been specified too early in the project. The software team faced severe problems because of these requirements.

> *'We have to start over with the product specification and restructure it and move many of the requirements, because many of these are very specific and need to be specified in the lower-level documents'. (Interview 4)*

This issue was complicated because the software team was not able to consult a customer representative or a customer proxy. Also, they found it difficult to understand the requirements. The lack of a customer representative also impacted negatively on the Scrum practice in other ways. Each sprint was initiated with a planning meeting and a sprint review meeting, but as no customer participated at the meetings, the developers found it difficult to define a common goal and commit to achieving the goal.

> *'We say we run a Scrum model, but who is our product owner? That has been one of the challenges since day one and had been a challenge long before I came. I asked for the product owner when I landed, but no one could give me an answer; they could provide a political product owner, but that is not what it is all about. It has to be someone who can make day-to-day decisions about the product'. (Interview 18)*

To address these issues, the software team attempted to engage a representative from the marketing group to fill the role of a product owner and help resolve the issues and uncertainties of the requirements. This, however, did not turn out as expected, the main reason being that management would not grant the resources needed for an internal product owner proxy. With a background in traditional plan-driven development of safety-critical products, the management did not understand the needs of the software team to be more agile. The role of the product owner was eventually assigned to the software architect within the software team. However, he did not have the necessary knowledge nor the authority to implement this tactic.

> *I write use cases right now, but I am also working with the software. I have to design the software; that means if I have misunderstood something when I write use cases, you find the mistake in the software because I do both. I would rather get the use cases from the business or at least have them help write the use cases, so we can avoid the misunderstandings, that I am sure are in there'. (Interview 9)*

5.2.2 Requirements Changeability

The marketing group was responsible for the high-level requirements that went through an approval process before being submitted to the software team. The software team was responsible for the detailed design of each software requirement. Thus, changing ill-described requirements at the higher levels of the requirements was challenging and time-consuming. Nonetheless, the software process manager recommended a rewriting of these requirements in the later stages of the project because of the problems experienced by the software team. The software team had to push for the rewriting of these requirements as the management did not realize how poor the requirements were and what consequences this had on the design.

> *'The software team acknowledged the problems of the high-level requirements, a long time ago, but the rest of the project did not want to acknowledge the problems; so it was difficult to push the changes through; but as far as I have heard, people are generally happy that they were rewritten'. (Interview 23)*

Many better-suited requirements came from the rewriting based on the experiences and learning in the project. To increase agility in the handling of the lower-level requirements, the software team transformed the software requirements into an internally and less formal product backlog with prioritized and estimated tasks, suiting the Scrum process. These tasks were referred to as user stories by the developers, even though the tasks were not written by users and were not in the form of user stories as proposed by the agile processes.

5.3 Life Cycle

The software team implemented a highly iterative software practice encapsulated in the linear phases of the overall project model. Meshing towards an iterative software practice showed how challenging it was, how aligning iterations with the milestones of the project model challenged the software team and how the co-development with other project teams interrupted the workflow of the software team.

5.3.1 Changing Practice

The overall project model was an adapted combination of the V-model and a stage-gate model, which is a linear model consisting of phases separated by gates, in which the process of the project is assessed before moving to the next phase. The V-model is recommended but not required by the FDA standard for developing safety-critical products and has been used by the pharmaceutical company for several years. The purpose of the gates is to ensure the production of appropriate documentation for regulative approval and to ensure the progress of the project. This model has been useful in the past when developing hardware and mechanics, which are much less iterative than software development. Therefore, the management of the project did not want to change the overall project model even though this model limited the agility of the software practice.

With the linear model, the software team lacked flexibility and the possibility of accommodating change. It was therefore decided to mesh Scrum with the software practice and thus divide the software development into sprints. This was not easy for the software developers, as they found it difficult to break the tasks into sprints. Most of the developers were experienced in safety-critical development. They were specialists and used to focusing on a specific parts of a development. The lack of generalist knowledge affected the tasks chosen for each iteration; instead of choosing the tasks with the highest priority, tasks were chosen according to the competences of the developers.

Gradually, most of the developers grew more accustomed to the Scrum practice and the sprints. A few developers were reallocated to other projects, and new developers with experience in agile development joined the software team. The software team gradually found agile development advantageous. They came to see short sprints as particularly useful as that gave the developers a fluid work rhythm and dedicated focus. This forced the team to break the tasks into even smaller units, which resulted in transparency and had a positive influence on the ability of the developers to estimate the tasks.

> *'Using two-week sprints, we had to be very precise breaking down the tasks and be very precise defining the criteria for when the tasks are done. That made it much easier to see what the tasks contained. We had to figure that out, and then it was easier to talk in a subgroup when we would be done'. (Interview 24)*

Gradually learning to become agile is likely a relevant lesson that can be learned from all kinds of software development. Here, it came about in the context of the development of safety-critical software, and as such, it was specific to the case and the fact that there was a mix of competences present as some developers were safety-critical specialists and some were already trained in agile methods but without the safety-critical knowledge.

5.3.2 Milestone Alignment

As the iterations were encapsulated in the linear project model, the software team still had to work towards the goal of each of the linear phases, and the iterations thus had to fit with the long-term milestones. For example, the developers had to specify the design of the software and get it approved to pass one of the milestones.

> *'. . . due to our previous milestone, suddenly we had to have a whole bunch of documents ready, which did affect the progress of the product, to have these documents ready'. (Interview 24)*

The milestones therefore profoundly influenced the content of each iteration and lowered its agility. It was difficult for the software team to develop a whole increment of the software because the requirements for a milestone were created based on traditional, linear thinking, and that, in turn, was caused by the perceived need to comply with the regulatory standards, according to the informants.

5.3.3 Hardware, Software and Mechanics Co-development

As the software was encapsulated in the medical product, the software team had to collaborate and co-develop with several of the other project teams. The software team was, for example, working in close cooperation with the engineers developing the hardware and the mechanics, such as providing the hardware team with test scripts to test the hardware. The requests for test scripts often came as interruptions within the software iterations and thus affected the outcome of the iterations.

> *'I don't think that, in half a year I have been employed here, there has been an iteration without any interruptions from other places. Some of them were small, but there have also been larger changes'. (Interview 9)*

The Scrum Master did try to enforce the rule of not interrupting the developers when in a sprint; however, several of these interruptions were unavoidable as serious problems affecting the safety-critical issues of the product as a whole had to be solved. Some of these problems arose due to severe errors in the existing software discovered at the system test at a release; others came from changes in the requirements by other project teams.

5.4 Testing

In the analysis of how the software team meshed the practice with the area of testing, two categories emerged: practice and competence, and management priorities and progress. Testing is a key concern in development of safety-critical software, and in this case, the developers needed to improve their competences in unit testing. Furthermore, the management needed to allocate resources to prioritize testing.

5.4.1 Practice and Competence

Even though the software team expressed an interest in implementing an agile testing practice, many unit tests were postponed until later stages of the project, and each increment was not fully tested at the end of an iteration before the software was released for the system test. Errors found during the later system tests were by the regulative standards required to be handled more formally than those found during unit tests and integration tests, and this slowed the progress.

'We started late on the tests, and there are several reasons why. Again, the knowledge about testing in the original software team was low'. (Interview 8)

Several of the developers did not initially have the necessary competences to conduct proper unit tests, and many of them prioritized the implementation of new functionality instead of finishing and testing the implemented functionality. Thus, the implementation of a test-driven practice required a significant change in the mindset of the developers. By the time the developers had gained more experience and had grown more attuned to agile software testing, much of the software had not been tested, and many errors still remained in the previous increments. This led to problems in the continuous integration of the software, and the software team spent too much time in correcting these errors.

'You have finished the code, but the release is scheduled soon, so the time for testing is shortened. The argument often is that the code you want to finish is important for the iteration'. (Interview 9)

5.4.2 Management Priorities and Progress

The management did not support the implementation of test-driven development. This management focus made improvements towards an agile test-driven practice challenging for the software manager. Testing was planned for the final stages in the plan-driven life cycle of the project, and very little time and focus were applied to testing. Management frequently wanted new functionality in the running prototype to present the latest version for the upper management and the marketing staff, as well as other stakeholders. As a result, no resources were allocated to support the introduction of better testing practices, and no resources were allocated to increase developer agile testing competences.

'It is difficult to get understanding at the upper levels that testing is important'. (Interview 1)

The process manager in the software team spent much effort on convincing management that testing was just as essential as functionality, but was not successful in doing so until very late in the project.

6 Discussion

Four areas of concern were identified in the review of the literature on agility in safety-critical software development, and these were instrumental in enabling us to focus on the categories of the codes found in the case study data, ultimately leading to the explanation displayed in Table 4 which resulted from the case analysis. In this section, we discuss the four areas of concern and the findings vis-à-vis the extant research.

6.1 Use of Documentation

The amount of documentation required for the validation of safety-critical software is a primary barrier in the existing literature (cf. Section 2) (McHugh et al. 2014a; Notander et al. 2013a). The case analysis corroborated this but in a slightly different way. The software developers spent much time and effort in writing and approving documents to satisfy the validation of the product, but they worked on the documentation with the sole purpose of being *compliant* with the standards. Other studies suggested that it should be possible to minimize the overhead of producing documentation (Grenning 2001; Stålhane et al. 2012). In the case analysis, it was found that the software developers considered documentation as a part of the delivery for each iteration, as also suggested by Wils et al. (2006), and moved towards being more agile with respect to documentation of compliance. Despite the fact that they focused on keeping the documentation to a minimum, they found this very difficult, as their main priority was to comply with the regulations of the FDA. Thus, even though they spent much time scrutinizing the requirements stated by the FDA, they found it difficult to decrease the focus on documentation.

The software developers did not depend much on the documentation during sprints, and they found this advantageous. Thus, the case study showed that splitting the purpose of documentation into (1) compliance and (2) knowledge sharing in the software team may be vital to striking a balance between compliance and agility.

The case analysis further showed that changeability can sometimes be very *fast*, and when requiring approval, it is often very *slow*. Handling documents flexibly and, at the same time, producing the compliance documentation for validation require deliberate tactics for being fast and slow at the same time. Being deliberately slow will be advantageous when, for example, standards such as those of the FDA stipulate that software requirements should be explicitly documented before implementation and testing (McHugh et al. 2014a; Van Schooenderwoert and Shoemaker 2018). In this case study, it was found that several documents were written and formally approved early on and that slowed their changeability dramatically, but existing research had nothing to offer to explain the differences of such documents and that they, therefore, must be handled differently.

6.2 Engineering of Requirements

Well-structured requirement engineering is needed according to the regulatory standards for safety-critical software development (cf. Section 2) (Jonsson et al. 2012; Vogel 2006). Moreover, according to Beznosov (2003), user stories written in simple business-like language cannot be used for validation. However, the case study revealed that, to comply with the agile process of Scrum and implement the product and sprint backlogs, the software developers needed to deal with requirements *uncertainty,* and they found it is easier to do with less formal specifications.

In the case study, it was found that the software developers acknowledged that they specified the requirements in two different ways, depending on degrees of uncertainty. For the software team in the case study, it is likely that following the recommendation made by Rottier and Rodrigues (2008) to separate functional requirements (less formal) and safety requirements (more formal) would have been advantageous in addition to considering the degree of uncertainty.

Adapting and changing requirements is a central feature and advantage of the agile processes (Lin and Fan 2009), and at the same time, changes in requirements are found to be problematic and potentially harmful to the safety argument (cf. Section 2) (McHugh et al. 2013; McHugh et al. 2014b). The case analysis indicated that, despite the requirements being based on an in-depth upfront analysis, it was also necessary to *change* the safety-critical software requirements. Requirements related to safety hazards are less likely to change as compared with the functional requirements (cf. Section 2), and this was largely corroborated in the case study. Thus, the categorization of the requirements into safety requirements and functional requirements allowed the developers to apply the requirements differently (Abdelaziz et al. 2015; Stålhane et al. 2012). In the case study, their requirements were specified in a hierarchy, that is, the higher the requirements were placed in the hierarchy, the more difficult it became to change. A more adaptable process of changing and modifying the functional requirements would be made more productive by placing the safety requirements at the higher levels and the functional requirements at the lower levels.

6.3 Life Cycle

The V-model is a fundamental life cycle model used in much of the literature to ensure validation and verification of produced artefacts (cf. Section 2). Because the V-model produces the necessary deliverables required when seeking regulatory approval (McHugh et al. 2014b), it is considered theoretically suitable for safety-critical software development. The case study showed a similar application of the V-model, but for the overall project, it was used along with a stage-gate model to ensure controlled progress, thus leading to a linear project life cycle with *milestones*. The agile software development was embedded in the overall linear project life cycle. The analysis showed that the milestones limited the flexibility of the iterations and thus considerably affected the agility of the software process. The software developers did not resolve this problem. On the contrary, they adjusted the content of the iterations to conform to the objective of the next milestone. There are similar findings in a case study of software development in general (Karlström and Runeson 2006). The literature on agile development of safety-critical software also does not address this issue and thus does not offer solutions.

Most safety-critical projects include embedded software, and the literature on agile safety-critical development does not address the challenge of coordination between project groups (Demissie et al. 2016). For embedded systems, software and hardware must be co-developed (Ronkainen and Abrahamsson 2003), and that was also the situation in the case study where the *co-development* involved software, hardware and mechanics. Within agile software development, it is recommended to adhere to an iterative development strategy (Cockburn 2006), whereas a linear process is considered to be suitable for the development of hardware and mechanics (Wysocki 2011). The software developers in the case study were required to balance the development of new functionality and test scripts. When the hardware and mechanics groups required software test scripts, they often interrupted the software team's sprints to develop these test scripts instead of adding them as tasks to the product backlog. The case analysis revealed that the coordination between sub-projects was addressed by asking the hardware and mechanics groups to submit their requests for support to the Scrum Master, who then would determine whether the issue needed to be solved right away (thus overruling the timebox) or if it could wait for the next sprint. The case study also showed repeated examples of requests from the hardware and mechanics groups that deliberately ignored this and kept disrupting the developers. Reasonable co-development is therefore difficult to achieve as software teams and hardware teams can work according to very different strategies.

6.4 Testing

The literature points to three challenges when meshing testing processes. First, to adopt an iterative testing strategy requires significant changes in the work practice (McCaffery et al. 2016; Rottier and Rodrigues 2008), and as already mentioned, developers find it difficult to change their working practice. The case study indicated that this is even more relevant when looking at testing practices. Changing to agile testing practices requires considerable new skills and competences, and this is not lessened when testing safety-critical software. The research literature, as well as the practice-oriented literature, addresses agile testing procedures (e.g. Droba et al. 2004; VanderLeest and Buter 2009; Van Schooenderwoert and Showmaker 2018), but the difficulties of acquiring the new and very needed skills and competences are not mentioned. In this case study, the management, in the initial phases of the project, realized that the software team did not have the appropriate competencies in agile software development and testing. They, therefore, hired additional experienced developers and agile consultants who could coach agile software development. But, they never managed to deliver fully developed and tested increments at the end of each iteration. The second challenge identified in the literature regards the roles related to testing (Jonsson et al. 2012). In the case study company, much testing was done by a separate testing team to comply with the FDA regulations; however, this team was even less agile than the software developers. The third challenge is how to handle heavy, iterative testing of the software increments (Kasauli et al. 2018; Rottier and Rodrigues 2008). This challenge was also shown in the case study. Although the company had successfully automated the tests, it struggled with many bugs in the software which made continuous integration very difficult.

7 Conclusion

The main objective of this study was to answer the question: *For safety-critical software, what can a software team do to mesh agile and plan-driven processes effectively?* The answer to the question is vital to understanding the possibilities and difficulties in agile safety-critical software development. From a literature review, we explained four areas of concern of particular importance when developing safety-critical software. We then reported from a case study of a pharmaceutical company in which an agile software team operated in an otherwise plan-driven environment. The contribution is an elaborated understanding of the four areas of concern and how they extend to the categories reported in Section 5 and summarized in Table 4 as a conceptual framework:

- Use of documentation
  - Compliance documentation

    - Fast and slow changeability
- Requirements engineering
    - Requirements uncertainty
    - Requirements changeability
- Life cycle
    - Changing practice
    - Milestone alignment
    - Hard, software and mechanics co-development
- Testing
    - Practice and competence
    - Management priorities and progress

In the discussion, the findings were compared with current research to explain the novelty of the findings. In this research, we followed the case study approach as advocated for software engineering research (Runeson and Höst 2009). The case study approach has limitations that are generic to any case study, namely, that it seeks insight from a single case. Within this single case, we have exercised several measures to increase the validity as there were multiple qualitative interviews, the data collection was longitudinal, explicit analysis and validation activities were conducted, and triangulation with document studies and participant observation was undertaken. The selection of the pharmaceutical company as the case organization was deliberate and served the purpose well as its products (and the processes leading to the products) were under the strict regulation of the FDA and, thus, highly relevant. There are two immediate and future research directions that can be taken to overcome the built-in limitations of our study. First, embarking on other qualitative studies of the subject in other case organizations where the challenges may well differ can very likely broaden the insights gained from, if not directly confirm, the findings of our study. Second, our findings can form the basis of a survey amongst developers of safety-critical software.

References

Abdelaziz, A., El-Tahir, Y., & Osman, R. (2015). *Adaptive Software Development for developing safety critical software.* International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), Khartoum, Sudan.

Bedoll, R. (2003). *A Tail of Two Projects: How 'Agile' Methods Succeeded after 'Traditional' Methods Had Failed in a Critical System-Development Project.* Conference on Extreme Programming and Agile Methods.

Beznosov, K. (2003). *Extreme Security Engineering: On Employing XP Practices to Achieve 'Good Enough Security' without Defining It X.* The First ACM Workshop on Business Driven Security Engineering, BizSec, Fairfax, VA, USA.

Beznosov, K., & Kruchten, P. (2004). *Towards agile security assurance*. Proceedings of the 2004 Workshop on New Security Paradigms, Nova Scotia. Canada.

Boehm, B., & Turner, R. (2005). *Balancing agility and discipline: A guide for the perplexed*. Boston, USA: Addison-Wesley Professional.

Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., & Kruchten, P. (2006). *Extending XP practices to support security requirements engineering.* Proceedings of the 2006 international workshop on Software engineering for secure systems, Shanghai, China.

Cockburn, A. (2006). *Agile software development: The Cooperative game*. Boston, USA: Addison-Wesley Professional.

Conboy, K. (2009). Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research, 20*(3), 329-354.

Demissie, S., Keenan, F., & McCaffery, F. (2016). *Investigating the Suitability of Using Agile for Medical Embedded Software Development.* International Conference on Software Process Improvement and Capability Determination (SPICE), Dublin, Ireland.

Drobka, J., Noftz, D., & Raghu, R. (2004). Piloting XP on four mission-critical projects. *IEEE Software, 21*(6), 70-+. doi:Doi 10.1109/Ms.2004.47

Fitzgerald, B., Stol, K.-J., O'Sullivan, R., & O'Brien, D. (2013). *Scaling agile methods to regulated environments: An industry case study.* Proceedings of the 2013 International Conference on Software Engineering, San Francisco, USA.

Gary, K., Enquobahrie, A., Ibanez, L., Cheng, P., Yaniv, Z., Cleary, K., Kokoori, S., Muffih, B., & Heidenreich, J. (2011). Agile Methods for Open Source Safety-Critical Software. *Software - Practice and Experience*, *41*(9), 945-962.

Ge, X., Paige, R. F., & McDermid, J. A. (2010). *An iterative approach for development of safety-critical software and safety arguments.* The Agile Conference (AGILE), Orlando, Florida.

Górski, J., & Łukasiewicz, K. (2012). Assessment of risks introduced to safety critical software by agile practices-a software engineer's perspective. *Computer Science, 13*(4), 165-182.

Górski, J., & Łukasiewicz, K. (2013). *Towards Agile Development of Critical Software.* The International Workshop on Software Engineering for Resilient Systems.

Gregor, S. (2006). The nature of theory in information systems. *MIS Quarterly, 30*(3), 611-642.

Grenning, J. (2001). Launching extreme programming at a process-intensive company. *IEEE Software, 18*(6), 27-+.

Hajou, A., Batenburg, R., & Jansen, S. (2015a). An Insight into the Difficulties of Software Development Projects in the Pharmaceutical Industry. *Lecture Notes on Software Engineering, 3*(4), 267.

Hajou, A., Batenburg, R., & Jansen, S. (2015b). Method æ, the Agile Software Development Method Tailored for the Pharmaceutical Industry. *Lecture Notes on Software Engineering, 3*(4), 251.

Heeager, L. (2012). Introducing Agile Practices in a Documentation-Driven Software Development Practice: A Case Study. *Journal of Information Technology Case and Application Research, 14*(1), 3-24.

Heeager, L., & Nielsen, P. A. (2009). *Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study.* The Australasian Conference on Information Systems, Melbourne, Australien.

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*, 22-39.

Jonsson, H., Larsson, S., & Punnekkat, S. (2012). *Agile Practices in Regulated Railway Software Development.* The 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW).

Kasauli, R., Knauss, E., Kanagwa, B., Nilsson, A., & Calikli, G. (2018). *Safety-Critical Systems and Agile Development: A Mapping Study.* The 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).

Karlström, D., & Runeson, P. (2006). Integrating Agile Software Development into Stage-Gate Managed Product Development. *Empirical Software Engineering*, *11*, 203-225.

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.* Proceedings of the 2017 International Conference on Software and System Process.

Lee, G., & Xia, W. D. (2010). Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility. *MIS Quarterly, 34*(1), 87-114.

Lin, W., & Fan, X. (2009). *Software Development Practice for FDA-Compliant Medical Devices.* International Joint Conference on Computational Sciences and Optimization (CSO), Hainan, Sanya, China.

McCaffery, F., Trektere, K., & Ozcan-Top, O. (2016). *Agile–Is it Suitable for Medical Device Software Development?* International Conference on Software Process Improvement and Capability Determination (SPICE), Dublin, Ireland.

McHugh, M., Cawley, O., McCaffery, F., Richardson, I., & Wang, X. (2013). *An Agile V-Model for Medical Device Software Development to Overcome the Challenges with Plan-Driven Software Development Lifecycles.* 5th International Workshop onSoftware Engineering in Health Care (SEHC), San Francisco, USA.

McHugh, M., McCaffery, F., & Casey, V. (2012). *Barriers to Adopting Agile Practices When Developing Medical Device Software.* International Conference on Software Process Improvement and Capability Determination, Plam de Mallorca, Spain.

McHugh, M., McCaffery, F., & Casey, V. (2014a). Adopting Agile Practices When Developing Software for Use in the Medical Domain. *Journal of Software-Evolution and Process, 26*(5), 504-512.

McHugh, M., McCaffery, F., & Coady, G. (2014b). *An Agile Implementation within a Medical Device Software Organisation.* International Conference on Software Process Improvement and Capability Determination, Vilnius, Lithuania.

Mehrfard, H., Pirzadeh, H., & Hamou-Lhadj, A. (2010). Investigating the Capability of Agile Processes to Support Life-Science Regulations: The Case of XP and FDA Regulations With a Focus on Human Factor Requirements. In *Software Engineering Research, Management and Applications 2010* (pp. 241-255): Springer.

Misra, S., Kumar, V., & Kumar, U. (2010). Identifying Some Critical Changes Required in Adopting Agile Practices in Traditional Software Development Projects. *International Journal of Quality & Reliability Management, 27*(4), 451-474.

Myklebust, T., & Stålhane, T. (2018). *The Agile Safety Case*: Springer.

Myklebust, T., & Stålhane, T. (2016). *Safety Stories–A New Concept in Agile Development.* International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2016).

Notander, J. P., Höst, M., & Runeson, P. (2013a). *Challenges in flexible safety-critical software development–an industrial qualitative survey.* International Conference on Product Focused Software Process Improvement, paphos, Cyprus.

Notander, J. P., Runeson, P., & Höst, M. (2013b). *A model-based framework for flexible safety-critical software development: a design study.* The 28th Annual ACM Symposium on Applied Computing, Coimbra, Portugal.

Paige, R. F., Charalambous, R., Ge, X., & Brooke, P. J. (2008). *Towards agile engineering of high-integrity systems.* International Conference on Computer Safety, Reliability, and Security, Newcastle upon Tyne, UK.

Paige, R. F., Chivers, H., McDermid, J. A., & Stephenson, Z. R. (2005). *High-integrity extreme programming.* The ACM symposium on Applied computing, Santa Fe, New Mexico.

Rasmussen, R., Hughes, T., Jenks, J., & Skach, J. (2009). *Adopting agile in an FDA regulated environment.* Agile Conference (AGILE), Chigaco, USA.

Ronkainen, J., & Abrahamsson, P. (2003). *Software development under stringent hardware constraints: Do agile methods have a chance?* International Conference on Extreme Programming and Agile Processes in Software Engineering, New Orleans, LA, USA.

Rottier, P. A., & Rodrigues, V. (2008). *Agile development in a medical device company.* Agile Conference (AGILE), Toronto, Ontario Canada.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering, 14*(2), 131.

Schwaber, K., & Beedle, M. (2001). *Agile software development with Scrum*. Upper Saddle River, New Jersey, USA: Prentice Hall.

Shafiq, S., & Minhas, N. M. (2014). Integrating Formal Methods in XP—A Conceptual Solution. *Journal of Software Engineering and Applications, 2014*.

Sidky, A., & Arthur, J. (2007). *Determining the applicability of agile practices to mission and life-critical systems.* Paper presented at the Software Engineering Workshop (SEW), Columbia, USA.

Spence, J. (2005). *There has to be a better way![software development].* Paper presented at the Agile Development Conference (ADC'05), Denver, USA.

Stålhane, T., Myklebust, T., & Hanssen, G. (2012). *The application of Safe Scrum to IEC 61508 certifiable software.* Paper presented at the European Safety and Reliability Conference (ESREL), Helsinki, Finland.

U. S. Department of Health and Human Services. (2010). FDA U.S. Food and Drug Administration. In: U.S. Department of Health and Human Services.

Van Schooenderwoert, N., & Shoemaker, B. (2018). *Agile Methods for Safety-Critical Systems: A Primer Using Medical Device Example*. CreateSpace Publishing.

VanderLeest, S. H., & Buter, A. (2009). *Escape the Waterfall: Agile for Aerospace.* The 28th Digital Avionics Systems Conference, Orlanda, USA.

Vogel, D. (2006). Agile Methods: Most Are not Ready for Prime Time in Medical Device Software Design and Development. *DesignFax Online,* 1-6.

Walsham, G. (2006). Doing interpretive research. *European Journal of Information Systems, 15*(3), 320-330.

Wang, Y., Ramadani, J., & Wagner, S. (2017). *An Exploratory Study on Applying a Scrum Development Process for Safety-Critical Systems.* International Conference on Product-Focused Software Process Improvement.

Wang, Y., & Wagner, S. (2018). *Combining STPA and BDD for Safety Analysis and Verification in Agile Development: A controlled experiment.* International Conference on Agile Software Development.

Wils, A., Van Baelen, S., Holvoet, T., & De Vlaminck, K. (2006). *Agility in the avionics software world.* International Conference on Extreme Programming and Agile Processes in Software Engineering, Oulu, Finland.

Wysocki, R. K. (2011). *Effective Project Management: Traditional, Agile, Extreme*. John Wiley & Sons.

Wäyrynen, J., Bodén, M., & Boström, G. (2004). Security Engineering and eXtreme Programming: An Impossible Marriage? *XP/Agile Universe 2004*, LNCS 3134, 117-128.

Yin, R. K. (2009). *Case study research: Design and methods* (Vol. 5). USA: Sage Publications Inc.

Özcan-Top, Ö., & McCaffery, F. (2019). To what extent the medical device software regulations can be achieved with agile software development methods? XP—DSDM—Scrum. *The Journal of Supercomputing*, 1-34.

Appendix A: Interview Guides

Interview Guide, Phase 1

Themes	Questions
Regulations	What are the regulative requirements? How do they affect your work?
Agile	Why did you implement an agile method? What is your evaluation of the agile method?
Process	Please walk me through the elements of your process? What is the evaluation of each of these elements?
	What are the strengths and weaknesses of this process? How do you evaluate this process? How do you coordinate within the software team? How do you coordinate with people outside the software team?
Documentation	Which documents do you use? Who is responsible for handling the documents? What purpose do the documents serve?
Requirements	How do you handle the requirements for the software? Where do the requirements come from? What are the challenges? What are the advantages?
Life cycle	How do you work in iterations? How are they planned? Evaluated? How many interruptions are introduced during the iterations? What is your evaluation of working in iterations?
Test	How do you test the software? Who is responsible? What is your evaluation of the process of testing?

Interview Guide, Phase 2

Themes	Questions
Changes	In what way have your work tasks changed since last time? What are the greatest changes that has happened since last time? What is your assessment of these changes?
Organizing of people	What changes have been made to the way people are organized? What are the advantages and disadvantages of these changes?
Process	Which changes have you implemented in your process? What are the advantages and disadvantages of these changes? Which role do the regulatory requirements play?
	What are the strengths and weaknesses of this process? How do you evaluate this process?
Agile	Which changes have been made to: The planning meetings? The stand-up meetings? The demo and retrospectives? The roles: scrum master, product owner? What are the advantages and disadvantages of these changes?
Regulations	Which boundaries do the regulatory requirements give? How do the regulatory requirements affect the way you run Scrum?
Documentation	Writing documents, which role have that played since last time? Which documents have you been working on? Who was responsible? What purpose have those played?
Requirements	Have you made any changes to how the requirements are handled? Changes to the requirements specifications? What are the advantages and disadvantages of these changes?
Life cycle	Have you made any changes to the iterations? The length? How are they planned? Evaluated? Do you still experience several interruptions in each iteration? What is your evaluation of working in iterations?
Test	Have you made any changes to how the tests are handled? When is the software tested? Who is responsible? What are the advantages and disadvantages of these changes?