Islam, G. and Storer, T. (2020) A case study of agile software development for large-scale safety-critical systems projects. *Reliability Engineering and System Safety*, 200, 106954. (doi: 10.1016/j.ress.2020.106954).

This is the author's final accepted version.

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

http://eprints.gla.ac.uk/213345/

Deposited on: 06 April 2020

Enlighten – Research publications by members of the University of Glasgow
http://eprints.gla.ac.uk

# A Case Study of Agile Software Development for Safety-Critical Systems Projects


## Abstract

This study explores the introduction of agile software development within an avionics company engaged in safety-critical system engineering. There is increasing pressure throughout the software industry for development efforts to adopt agile software development in order to respond more rapidly to changing requirements and make more frequent deliveries of systems to customers for review and integration. This pressure is also being experienced in safety-critical industries, where release cycles on typically large and complex systems may run to several years on projects spanning decades. However, safety-critical system developments are normally highly regulated, which may constrain the adoption of agile software development or require adaptation of selected methods or practices. To investigate this potential conflict, we conducted a series of interviews with practitioners in the company, exploring their experiences of adopting agile software development and the challenges encountered. The study also explores the opportunities for altering the existing software process in the company to better fit agile software development to the constraints of software development for safety-critical systems. We conclude by identifying immediate future research directions to better align the tempo of software development for safety-critical systems and agile software development.



*Keywords:*


## 1. Introduction

The software industry, as a whole, is witnessing a gradual transition from traditional plan-driven process models to agile software development (Chapman, 2016; Chapman et al., 2017; Glas and Ziemer, 2009; Paige et al., 2011; Wils et al., 2006). A 2018 survey of software industry practitioners found that 97% of respondents reported using agile methods (CollabNet VersionOne, 2019). In addition, the survey found that 78% of respondents reported that the teams in their organisation continued to use a mix of agile and plan based methods and practices. Advocates of agile software development contend that plan-driven software processes lack the flexibility to respond to rapidly changing business requirements (Beck and Andres, 2005; Beck et al., 2001; Schwaber and Beedle, 2001). Agile software development addresses this demand for flexibility by emphasising the organisation of work into small co-located teams, short development cycles punctuated by deliveries of software releases to customers for review and feedback, encouraging frequent informal communication amongst software team members and the exclusion of practices that do not demonstrably contribute value to the project customer, often including formal documentation (Black et al., 2009; Rayside et al., 2009). Such values are embodied in a number of agile methods, such as Feature Driven Development (Palmer, 2002), Extreme Programming (XP) (Beck and Andres, 2005) and Scrum (Schwaber and Beedle, 2001). Each agile method may also be characterised by a number of agile practices, such as daily standup in Scrum or pair programming in XP. Methods may also be customised by the addition of supplemental practices, or practices themselves may be customised to meet the demands of the project context.

Several researchers have argued that such characteristics of agile software development are best suited to small scale projects (Boehm, 2002; Boehm and Turner, 2003) with research suggesting that agile softare development is effective in these contexts (Paetsch et al., 2003). The application of agile software development in safety-critical systems engineering projects, comprising multiple teams, developing both hardware and software and spanning several years of delivery effort, is more contentious (Boehm, 2002; Boehm and Turner, 2003). Critics argue that such projects have very different characteristics and constraints that invalidate many of the assumptions underpinning agile software development. For example:

- Notander et al. (2013) argue that the imposition of safety-critical standards, accompanied by required processes, limits the ability of a software development team to reflect on and adapt their processes as they see fit to meet the project's goals. This conflicts with the desire within agile software development for teams to take responsibility for their own processes, selecting and composing practices that fit the demands of the project (Schwaber and Beedle, 2001).

- Stelzmann (2011) argued that agile software development is incompatible with projects that incorporate a significant amount of hardware engineering, due to the length of time and cost of building prototypes.

- (Boehm, 2002; Cohen et al., 2004; Lindvall et al., 2002; Misra et al., 2009; Siddique and Hussein, 2014) have argued that the demand for extensive supplemental documentation to demonstrate conformance with standards conflicts with the agile principle of prioritising the delivery of working software.

Nevertheless, there is growing interest in applying or adapting agile software development to safety-critical systems projects, driven by business demands for smaller and faster deliveries (Chapman, 2016; Chapman et al., 2017). Researchers have also begun exploring the use of agile software development in safety-critical systems development (Gary et al., 2011b). A number of case studies and experience reports in the academic literature have reported on this transition in diverse domains, including railways (Jonsson et al., 2012), medical science (McHugh et al., 2013) and most relevant to the present research, avionics (Wils et al., 2006; Chenu, 2012).

Many of these studies conclude that agile software development requires adaptation for application to safety-critical systems. For example, Notander et al. (2013) conclude that agile software development, while not incompatible with typical safety-critical standards, need to be modified for use on safety-critical system projects. The practice of adapting and customising methods and practices to suit local needs has been reported for other software domains (Fitzgerald et al., 2006; Wang and Wagner, 2016b; Conboy, 2009). However, there has been a very little reported in the literature of the experience of practitioners who have applied necessary adaptations to agile methods or practices in the context of safety critical system development. Therefore there are many open questions about the selection of particular adaptations and their efficacy in different contexts.

To continue to address this gap, we conducted a series of semi-structured interviews with software engineers working for a large avionics company in the United Kingdom (referred to as 'the company'). The company as a whole is engaged in a variety of projects for external customers, typically comprising both hardware and software development for safety critical systems. The purpose of the study was to learn about the company's experiences in the application of agile software development to safety-critical systems projects and to gain a deeper insight into the difficulties experienced. Therefore, the two research questions addressed within the context of the case study in this exploratory research were:

**RQ1** What agile methods and practices are being employed in the context of software development for safety-critical systems?

**RQ2** What are the challenges in employing agile methods and practices in the context of software development for safety-critical systems?

Addressing the first question provides an understanding of the use of agile software development within the company. Addressing the second question allows for an exploration of the impact of agile software development from the perspectives of the practitioners. We also seek to understand what challenges *they* encountered when employing different practices within agile methods, which practices were rejected and adapted, and the rationale for doing so. Due to the exploratory nature of the research, a case study approach was taken (Runeson and Höst, 2009). An initial interview with stakeholders at the company was conducted as a scoping exercise. Following this, a semi-structured interview instrument was developed following Wengraf's (2001) method to ensure traceability between research questions and data gathered. Findings from this stage were validated in a full-day workshop with wider group of participants. We present the full results of this investigation here.

*Contribution:* This paper significantly extends the existing evidence base for the application of agile software development within safety-critical systems engineering by investigating the challenges from the perspective of practitioners. We conducted four semi-structured interviews with employees of the company in a variety of roles in different software projects and with diverse experiences. The interview structure was based upon the information gathered during an initial exploratory conversation with two senior employees. The findings of the study were validated in a workshop with a wider number of participants drawn from across the company's software development function. The extent of the material generated from these interviews allowed us to gain significant insight. Specifically, we report on how some teams within the company have employed an agile software process (Scrum) within a Waterfall process for the wider systems engineering project. We elaborate on this integration by describing how the teams have made necessary customisations to Scrum to fit within this process. We describe the successes that the teams have experienced in employing and adapting individual agile practices, such as, planning poker, continuous integration, automated static analysis and code reviews, as well as, discussing where the use of agile software development has led to drawbacks. We also investigate practices that the teams have not employed, such as, pair programming and user stories, and discuss the rationale for this from the teams' perspective. Where appropriate, we relate these insights to the available literature. The work, therefore, provides a substantial case study based on evidence from industry of the real world challenges of employing agile software development for safety-critical systems and provide a foundation for future research in addressing these challenges.

This paper is structured as follows: Section 2 provides an overview of the relationship between agile methods and safety-critical system development. Section 3 describes the research method for this study including the design of the semi-structured interview instrument and validation of the findings in a review workshop with the company. Section 4 provides an overview of the company, and how it approaches systems engineering, giving an understanding of the context in which agile software development is employed. Section 5 summarises the use of agile software development, including specific practices, to date within the company, and how these have been fitted into the existing software development process. Section 6 discusses the challenges discovered from the interviews. Section 7 presents the conclusions drawn from the work, identifies a number of limitations and discusses future work.

## 2. Background

This section provides an introductory background to agile software development, characteristics of software development work for safety-critical systems engineering projects. The section also presents a review of related work concerning the application of agile methods to software development for safety-critical systems.

### *2.1. Agile Software Development*

Agile software development emerged in the late 1990s and is considered to be a response to the failure of existing plan based software development processes, such as Waterfall (Benington, 1983; Vijayasarathy and Butler, 2016; Wang et al., 2012) and the Rational Unified Process (Rational; Tanveer, 2015) to accommodate the highly volatile nature of requirements for software development projects. A common critique of these methods is that the lifecycle of software delivery is far slower than the pace of change in the problem domain (Schwaber and Beedle, 2001; Tanveer, 2015; Koronios et al., 2015; Abrahamsson et al., 2017). For example, a typical iteration in the Rational Unified Process is between six and twelve months, during which time, the requirements for the project or the technology available in the market place may have changed considerably. Proponents of an *agile* approach to software development (Beck et al., 2001; Abrahamsson et al., 2017), instead advocate for a process model that is based on continual review of progress and requirements through continued close collaboration with the customer. Schwaber and Beedle (2001); Abrahamsson et al. (2017) explain that this approach is derived from empirical process engineering, in which, rather than attempting to design a software process apriori, process engineers closely monitor and make small, frequent changes to the production process. As a consequence of this approach, a team practising agile software development will still begin work with a broad understanding of the long term objectives for their project, but will avoid detailed planning for all except the most immediate project activities.

Agile methods are a family of software process models that share this common agile philosophy. Examples of agile methods include Lean (Poppendieck and Poppendieck, 2003; Dingsøyr and Lassenius, 2016), Crystal (Cockburn, 2004), Feature Driven Development (Palmer, 2002), Extreme Programming (XP) (Beck and Andres, 2005) and Scrum (Schwaber and Beedle, 2001). A unifying characteristic of these process models is that they are *iterative* and *concurrent*. Software development takes place within short iterations of typically two or three weeks, but sometimes as short as a single day, punctuated by deliveries to a customer for immediate feedback and review. In further contrast to plan based methods, within each iteration, multiple software development activities may occur concurrently, including requirements analysis, design, implementation and testing. Each agile method is itself further characterised by a set of practices undertaken to support development work and manage the complexity of the concurrent software process. Examples include backlog grooming, planning poker, sprint planning daily standups and retrospectives from Scrum (Schwaber and Beedle, 2001); spike prototyping, automated unit testing and refactoring in extreme programming and value-chain mapping in Lean (Poppendieck and Poppendieck, 2003; Dingsøyr and Lassenius, 2016).

According to industry surveys, Scrum and XP are the most frequently reported methods employed by software teams for organising an agile software development process (Wang et al., 2012; CollabNet VersionOne, 2019). Schwaber and Beedle (2001) and Lei et al. (2017) state that the Scrum process works well for small teams of between three and nine members. Key roles within Scrum include the Scrum master, responsible for facilitating team activity and the product owner, responsible for managing the relationship between the customer and the team. The Scrum process comprises of short iterations called sprints, typically lasting 1-3 weeks. Each sprint begins with a planning meeting during which new requirements are transferred from the *product backlog* to the *sprint backlog*. The sprint begins once the requirements are agreed for the sprint backlog. Communication between team members is maintained through a daily meeting, called a *stand-up*, during which each team member briefly reports progress, plans and any issues that have arisen. At the end of a sprint, the team holds a *review meeting* during which progress is compared against the goals of the sprint.

The XP process, as described by Beck et al. (2001)and Wang et al. (2012) has a similar focus on short iterations punctuated by releases to the customer. Similar practices to Scrum are also advocated for project management, such as a daily stand-up meeting and release planning for an iteration. However, in contrast to Scrum, XP practices focus on the lower level activities associated with software engineering. For example, XP advocates the use of user stories developed in user story workshops for requirements gathering; test driven development for both new features and bug fixes; and refactoring as an explicit practice to maintain code quality. Other practices are also recommended to foster team communication through pair programming, for example. Schwaber and Beedle (2001) argue that the two methods are complementary and can co-exist in a single team with Scrum providing a *wrap around* for the practices within XP.

### *2.2. Software Development for Safety Critical Systems*

According to Knight (2002) "*Safety-critical systems are those systems whose failure could result in loss of life, significant property damage, or damage to the environment.*". Examples include nuclear systems, medical devices, air traffic control, avionics, railway control systems, and automotive control systems. Due to the involvement of physical risks, development of safety-critical system development is typically undertaken within respect to particular generic or domain specific standards or other regulatory constraints (Heeager and Nielsen, 2018). Such standards may impose considerable structure on the software development process including the selection and ordering of activities. Furthermore, standards may specify artifacts that must be produced during the development to show conformance. For example: DO-178C is a standard for development of airborne software. Similar standards exist for other domains, such as IEC 62304 for development of Medical devices, ISO 26262 for automotive and IEC 61513 for nuclear.

Regulatory standards can be classified by their scope i.e. generic vs. domain specific (Gruber et al., 2010; Notander et al., 2013). Notander et al. (2013) divide regulatory standards into two categories *(i) means-prescriptive*: in which the methods by which software development will proceed is either required or recommended and *(ii) objective-prescriptive*: that defines what objectives the resulting system artifacts must satisfy, without stating how the objectives are achieved. For example, the avionics standard DO-178C specifies 71 objectives in total, covering the full scope of the software life-cycle. The number of objectives that must be met is dependent upon the level of criticality of the system, a qualitative scale, ranging from Catastrophic through to Minor. Each of the objectives requires performing different activities, as a result of which, a number of artifacts are produced including documents. These artifacts are presented as proof of conformance at the certification stage. Demonstration of conformance means doing additional activities which also impacts the pace and cost of development (Wong et al., 2011). For example, the objective "*Source Code complies with low-level requirements"* can be demonstrated through the artifact "*Software Verification Results"*; and "*Assurance is obtained that software life cycle processes comply with approved plans"* is demonstrated through software quality assurance records.

According to Notander et al. (2013), means-prescriptive standards dictate traditional life cycles, making accommodation of agile software development much more difficult. On the other hand, objective-prescriptive standards, such as DO-178C may offer fewer restrictions.

### *2.3. Related Work*

Much of the published literature on the application of agile software development to safety-critical systems work is speculative, suggesting considerable uncertainty amongst practitioners concerning how best to proceed in applying and adapting agile software development in the context of safety-critical systems development. In a recent survey of the field, Heeager and Nielsen (2018) reviewed 51 papers published over two decades (2001 – 2018). Heeager and Nielsen found that of those papers, 10 were based on case studies and a further 5 were considered to be experience reports, such as Gary et al. (2011a). Another experience report not listed by Heeager and Nielsen (2018) is the work by Chenu (2012).

Relatively few studies have developed conclusions based on detailed interviews with practitioners. Of the existing research, McHugh et al. (2013) conducted interviews with practitioners working on the development of medical devices. Notander et al. (2013) interviewed five engineers at four different companies to understand the impact of increasing demands for flexibility on established safety-critical development. Siddique and Hussein (2014) interviewed 21 individuals, each in different companies in Norway to understand the practical choices made by software engineers in choosing a development method. Reporting on then on-going interview-based research, Stelzmann (2011) proposed a classification scheme for different safety-critical contexts in which agile software development is being considered or applied. Hajou et al. (2015) conducted 14 interviews with software developers in the pharmaceutical industry to understand the reasons for the lack of adoption of agile software develoment in that context. In particular, the authors concluded that the perceived risk of agility mitigated against its adoption.

A common theme in the work on applying agile software development in a safety-critical context has been the need for adaptation of agile methods and practices to fit within the constraints of safety standards. For example, McHugh et al. (2013) suggested that incorporating agile methods with existing plan-driven methods is the most favourable choice in the software organisation they studied. To facilitate this, McHugh et al. propose a hybrid V model which incorporates aspects of agile methods and activities from plan-driven methods.

A more extensive investigation of the integration of agile software development with safety-critical systems has been developed in the SafeScrum method (Stålhane et al., 2012). The original motivation for this work was the integration of the Scrum method with the IEC 61508, a high level standard for safety-critical systems. The key intuition in the approach is that safety requirements change far less frequently and are far more certain than product requirements. To accommodate this, the SafeScrum method (a) focuses only on software development within the overall system engineering process; and (b) maintains separate Scrum backlogs for functional and safety requirements.

Later work on SafeScrum extended the assessment of its compatibility with a variety of other safety standards, such as in the petrochemical industry (Myklebust et al., 2016). Other authors have also considered extensions to the original SafeScrum method, including the integration of change impact analysis into the agile change request lifecycle (Stålhane et al., 2014), safety analysis (Wang and Wagner, 2016a) and configuration management (Stålhane and Myklebust, 2015).

A limitation of much of the work on SafeScrum is the lack of case studies or experience reports, evaluating the method through industrial experience. However, Hanssen et al. (2016) undertook a two year case study of applying SafeScrum to the development of a fire detection system. As a consequence of the case study, the authors discovered the need to augment SafeScrum with an embedded quality assurance role within the development team. The duration of Hanssen et al.'s case study demonstrates the difficulty of conducting real world evaluations of methods for safety-critical systems. Equally, the work demonstrates the importance of doing so in order to identify necessary adaptations to theoretical process models.

## 3. Research Method

The company that is the focus of this study is a large multi-national that develops products in the avionics sector. The company is engaged in a number of projects concerning the design and development of safety-critical systems, comprising both hardware and software. As discussed above, the company had begun to experiment with the use of elements of the Scrum process and other agile practices. During this period, the researchers were invited to conduct interviews with a number of the company's employees who had been involved in this transition process. The purpose of this study was to explore and understand the application of agile software development to the development of software for safety-critical systems from the perspective of practitioners. The study sought to identify both: the benefits recognised by practitioners in using agile methods and practices in this context and the challenges and limitations experienced. We conducted a series of interviews with practitioners at the company.

Since this was an exploratory study, and the researchers did not have prior experience of the company's work, the first stage of the research process was an unstructured interview (Interview 0) with two senior employees of the company. One of these participants, who also participated in all the following interviews, was the team lead of a systems team, which was responsible for elaborating requirements and disseminating these to other teams within a larger project. The other participant was the Head of Software Engineering, who is responsible for the overall software development function of the company. The interview meeting continued for 90 minutes. This interview was conducted in person, with one of the researchers taking extensive notes during the interview. A memo was prepared summarising the answers to questions asked. This memo was validated by one of the interviewees during a follow-up discussion. The answers to this initial interview provided guidance to help scope the next stage of our research.

Following this stage, semi-structured interviews were used to gather data. This approach offers freedom of expression to the participants, and open-ended questions prompt discussion aiding the interviewer to explore a particular theme. Following McHugh et al. (2013), Wengraf's guidelines were used to construct the interview instrument (Wengraf, 2001). Figure 1 illustrates how Wengraf's method was applied to the design of the semi-structured interviews.

This is a top down approach beginning with a *Research Purpose* (RP), in this case: "*Learn about application of agile software development to software development for safety-critical systems and to gain a deeper insight into difficulties experienced when developing avionics systems using agile methods and practices."*. The RP is then refined as one or more *Central Research Questions* (CRQ) that encompass the broader aspects of the research purpose. In the current work, the RP is refined into two research questions stated in the introduction and included in the figure for completeness. Each CRQ is divided into a number of *Theory Questions* (TQ), specific propositions to be investigated during the conduct of the study. For example, CRQ1 is refined into two TQ, including "*TQ1.1 What agile methods are employed in practice?"*. To answer each TQ, a number of interview questions that will be presented to the participants are defined. The figure shows a sample of interview questions for TQ1, with the full interview instrument available for review (AUTHORS, 2018). This approach provides a traceable hierarchy and rationale behind every interview question.

Once an initial version of the interview instrument was prepared, it was validated by an independent academic expert who did not have any involvement in the research. The validator was contacted by email to arrange a teleconference during which all questions in the interview instrument were reviewed. The validator advised altering the order of questions to facilitate the interview process, but did not recommend changing the content of any questions. A series of mock interviews were also conducted with non-participants

| Research Purpose | Central Research Questions | Theory Questions | Example Interview Questions |
|---|---|---|---|
| Learn about the application of agile software development to software development for safety-critical systems and to gain a deeper insight into difficulties experienced when developing avionics systems using agile methods and practices. | 1. What aspects of agile methods and practices are being employed in the context of software development for safety-critical systems? | 1. What agile methods and practices are employed ? | Customer Involvement 6. Are multiple releases delivered to the customer during a project? |
| | | 2. What customizations have they made to the method and practices they are employing? | Requirements 9. How are requirements managed during elaboration/change/evolution? |
| | 2. What are the challenges in employing agile methods and practices in the context of software development for safety-critical systems? | 3. What benefits did they expect from agile software development? | Requirements 4. How does certification drive quality assurance practices? |
| | | 4. What benefits were they able and not able to achieve? | Requirements 10. How often are requirements reviewed? How is this done? |
| | | 5. What are the potential conflicts of agile software development with regulatory standard(s) (i.e. DO-178C) ? | Quality Assurance 4. How does certification drive quality assurance practices? |

Figure 1: Research question construction process following Wengraf's method (Wengraf, 2001) for the interview instrument used in the Avionics Company.

in the study to familiarise the researchers with the structure of the interview instrument and to test the timing and duration of the interviews.

Four interviews were conducted during four sessions. Our intention was to gather data from multiple perspectives within the company, creating a broader understanding of the research. Interviews were conducted with five practitioners (Participants P1-P5) with different experiences, expertise, and roles. These experiences included acting as a project manager, requirements engineer, software developer and a member of an integration team. The fifth participant, P5, is a systems team lead and participated in all the interviews. The first four interviewees were working on three different projects within the company. The first team had some experience of employing agile software development within their projects whereas the second software team was considering its use because they wanted to be able to deliver more frequent releases. In both cases, the participants interviewed had used an agile method and associated practices in their *previous* projects within the company. However, the third software team was reluctant to adopt agile software development and wanted to retain their existing plan based process, which resembled Waterfall (Benington, 1983). The third team felt that they worked effectively within this process and although aware of the use of agile software development elsewhere within the company, did not see the need to begin introducing an agile method or practices to their own software process. All the participants, including the ones with experience of agile software development within the company, worked on avionics related projects requiring D178-C certification. A summary of the interview participants is presented in Figure 2.

The approximate duration for each interview was 90 minutes. Interviews were transcribed and sent to the participants for validation, permitting participants to make additions or clarifications. After getting verbal permission from the participants, the transcripts were used for analysis. The transcripts from the interviews were then analysed to answer the theory questions. The analysis of the gathered data is also performed by using Wengraf (2001)'s guidelines, using a bottom-up approach to answer the questions at each level.

For the analysis, answers to the questions were gradually aggregated at each stage in the hierarchy. A table was created similar to Figure 1 for this purpose. Answers to every interview question from all participants were pasted in the Answer column next to the respective interview question. Answers to every group of IQ relating to each Theory Question were then merged to form a story. The group of Interview Questions relating to each Theory

| Participant and Role | Experience of Agile |
|---|---|
| P1: Lead software engineer | Using agile and practices within current team; experience of using agile on previous projects |
| P2: Lead software engineer | Experience of using agile software development in previous projects; Considering the use in current project |
| P3: Deputy lead software engineer | Using waterfall |
| P4: Lead software engineer | Using waterfall |
| P5: Systems team lead | |

Figure 2: Summary of Interview Participants

Question was deleted such that each Theory Question had a descriptive answer. The same process was repeated again to find answers to CRQs.

The descriptive answers to each CRQ were reviewed by the authors independently, and the issues reported in them were highlighted. The notes were compared afterwards in a meeting to discuss the discovered issues. Eleven challenges were identified during this data analysis. These results were presented to a group of people from the company for validation. The participants in the workshop validated all the challenges identified during the interviews, with the exception of one. In addition, the participants of the meeting raised three new challenges which were not discovered during semi-structured interviews. All fourteen of these challenges are discussed in Section 6. As a result, we also gained an understanding of the factors that directly or indirectly affect and contribute to the actual and perceived benefits of agile software development within the company. At the end, the findings from the interviews were mapped to findings in the literature. Note that where we use quotations below to illustrate a challenge it is sometimes necessary to anonymise some of the topics to preserve confidentiality. All the work described in this section took place between March 2017 and March 2018.

## 4. Overview of Software Development in the Company

This section draws on the analysis of the answers to the interview questions to develop a description of the structure and process for software development used by the company. The description below addresses Research

Question 1, as well as providing context for the discussion of challenges which were identified during the interviews and discussed in Section 6. Each theme discussed below was identified in the interviews as having an impact on the introduction of agile practices to the software teams. The Section begins with an overview of the a typical project team structure, organised to accommodate both hardware and software development processes. The section then describes the *overall* software development process within the company and where agile software development has been adopted within individual sub-teams. Next, the section describes the relationship between a typical project in the company and a complex network of project customers. The next section reviews the requirements management process, showing how requirements derived for the overall project are communicated to the software teams and sub-teams. Finally, the process of delivering and certification for products according to safety standards is described.

### *4.1. Project Team Structure*

The size of project teams within the company varies considerably, typically between 50 and 200 people. Within a project, a software development team (SDT) itself typically comprised of 20 to 35 people, with the rest of the project team working on different other components or functions within the project, including the systems integration team, hardware, firmware, software, safety, flight trials, configuration and the management team.

The SDT has its own organisational structure. The overall team has a small management unit, comprising a lead software engineer, deputy lead software engineer, program manager and coordinator. The lead software engineer and deputy lead software engineer share technical and managerial responsibilities for the overall project. These include the overall software lifecycle, comprising requirements, definition, design, software implementation, quality assurance, certification and delivery. The lead software engineer is also responsible for customer liaison and has sign-off authority for documentation and software changes. The lead software engineer is also responsible for assigning responsibilities to individual software sub-teams. The software program manager has responsibility for project planning within the software team and resource allocation. Finally, the software coordinator is responsible for maintaining documentation, for example, meeting minutes.

A software team is typically divided into a number of sub-teams, which specialises in a particular functional aspect of the software project and consists of either four or five people. Each sub-team has a sub-team leader, who is expected to be able to run a full lifecycle including high level design and requirements analysis within their area of expertise. The sub-team leads also act as *functional champions* because of their expertise in some area of functionality. The sub-team leaders typically have 15 to 30 years of experience. Other members of the team have different level of experience, from recent graduates to 20-30 years of experience.

### *4.2. Development Process*

Most of the projects within the company, including the participants' current projects, are planned to run for several years and are divided up into a number of *phases* with each phase intended to deliver further new functionality on the product, as agreed with the customer(s). The duration of a *phase* varies from project to project. In some projects, a phase is between four (4) and six (6) months and in others, a phase is between one (1) year and eighteen (18) months. Each phase is allocated a number of requirements to be implemented, agreed with the project customer. At the end of each successful phase, a delivery is made to the customer comprising (in the ideal case) the features of the requirements that were originally agreed upon.

A typical phase is illustrated in Figure 3. Requirements are created in the IBM DOORS documentation tool by the systems team and later exported into the IBM Rhapsody modelling tool used by the requirements analysis sub-team within the SDT. The requirements analysis team translates the requirements into a high level software architecture. During this process, the software team and systems team are in constant communication, due to the need to further negotiate and clarify the requirements. Once the requirements and architecture are agreed upon, they are allocated to different sub-teams by the requirements manager. Within each sub-team, the company allows some flexibility with regard to the software process, for example, with some sub-teams using a Waterfall software process within a single phase and others applying the Scrum method. Consequently, one participant (P2) called their software process "*water-scrum-fall*", as Scrum was inserted into the middle of the company's overall project lifecycle. Towards the end of a phase, different functions of the software are packaged into an integrated software release. The software is delivered to the integration team to develop an overall delivery release to the project customer.

There is a set practice of having a weekly technical and management meeting and a monthly software team meeting. Minutes and actions are captured at the meetings and distributed only to the relevant people. Other than

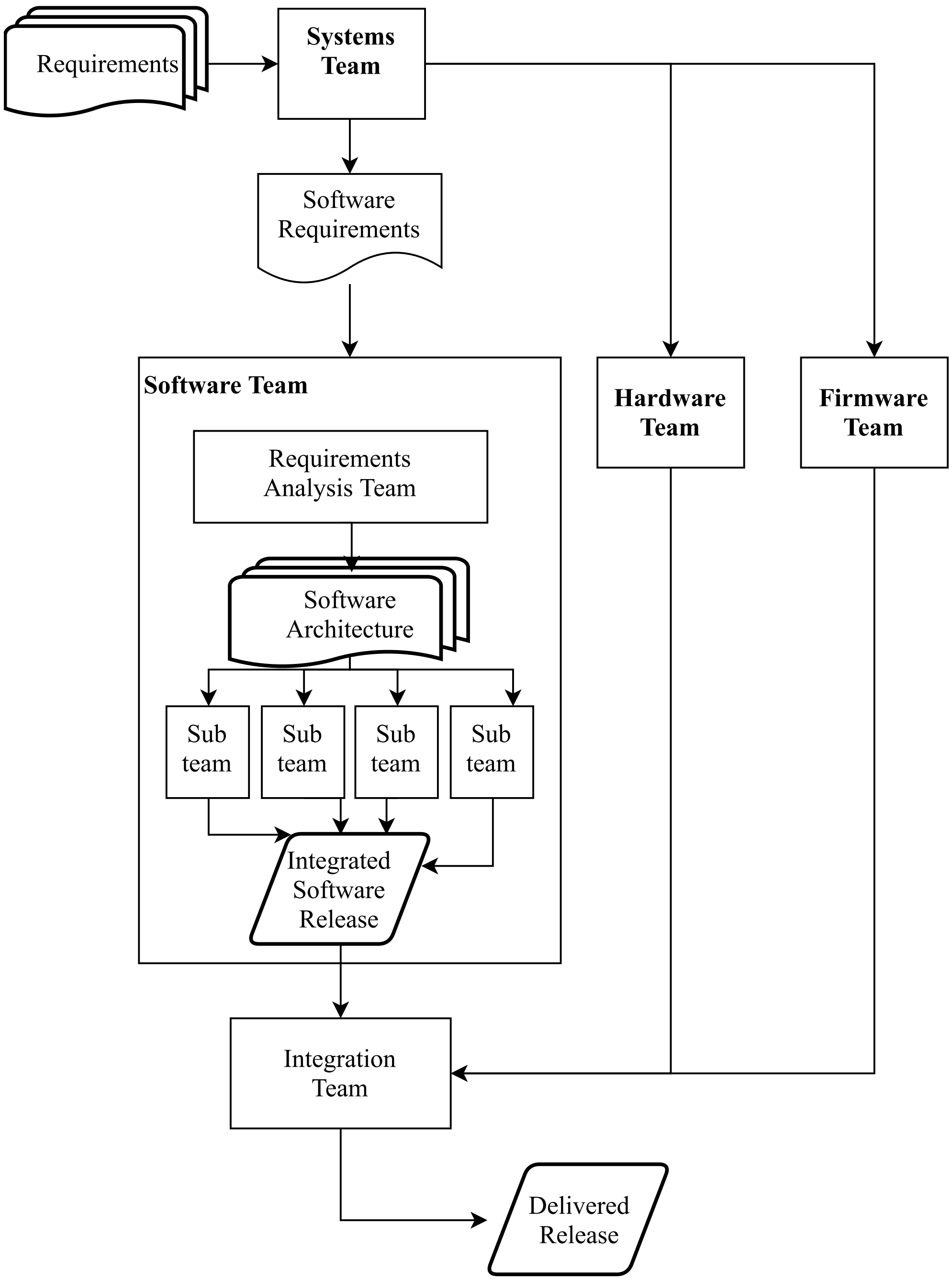


Figure 3: A typical phase of a project from the perspective of the Software Team

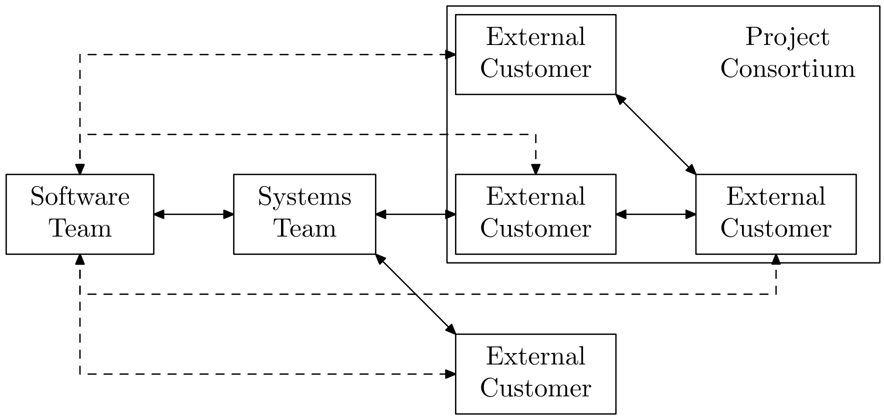


Figure 4: Layers of Customers. Solid arrows represent formal lines of communication. Dashed arrows represent informal or infrequent lines of communication.

the formal meetings, spoken/face-to-face communication is the main type of interaction that takes place between the software team and other teams. Within each sub-team, members are co-located and interviewees report that the culture within the company encourages workplace interaction.

### *4.3. Project Customers*

From the perspective of a project software team, the relationship with the project customer was viewed as complex, with the project actually having several ‘layers’ of customer (Figure 4). The systems team acts as the most immediate customer for the software team, providing the requirements specification (recall Figure 3). In turn, the systems team manages the relationship with the project’s immediate external customer. The systems team is therefore responsible for gathering requirements from the external customer. As the company may be part of a larger project consortium, the external customer may itself also have a further external customer who will have a significant influence on the direction of the project. Alternatively, the system under development may have several direct customers. In all these cases the software team may find themselves interacting less frequently with these stakeholders, or doing so through informal communication mechanisms, indicated by the dashed arrows in Figure 4.

One of the interview participants (P5) described this as “*a very complex stakeholder relationship in terms of lots of people with different views and influences.*” The customer has a certain delivery schedule which has the main influence over the overall schedule. The interview participants reported that in the past, the overall project management team decided the project schedule, but now the software team also give their input on tasks and schedule. Although the wider project management team sets the major milestones in agreement with the external customer, the software teams set their own milestones within these boundaries. This gives the team members a sense of ownership and responsibility. Agreed delivery dates are then passed onto the external customers. Normally, the software team would involve more people if there is a risk of missing the delivery date but if the schedule needs to be changed it is done after negotiation with the external customer. Final decision about changes to a schedule is made by the Software Function lead.

For the software team, the "*customer*" is primarily the project's systems team, who partitions and allocates requirements to teams within the project. Consequently, the systems team is usually one or two delivery phases ahead of a software team. For example, the systems team will be preparing requirements for the second or third phase while the software team is working on the first phase. The main involvement of a systems team is in the beginning (elaborating requirements) and at the end (completing integration) of each phase. A systems team does not participate in the feedback reviews regularly, but if there is a very complex task (a complex algorithm to be implemented, for example), they would get involved. The systems team also provides inputs for acceptance testing.

The interview participants reported that in the past, their software team has had ready access to the systems team, who can be approached on a needs basis. However, there is no pre-defined way of soliciting feedback from the respective systems team. Rather, it is mostly informal, whenever needed. Conversely, gate reviews and interim reviews are formally performed with the external customer (representatives). Normally it takes more than six weeks to get feedback on a delivery, as the customer requires this time to test the new features on the integrated system. Certification also delays delivery sometimes.

### *4.4. Requirements Management*

Requirements are analysed and refined at the start of each iteration. At the end of requirements analysis phase of each iteration, the requirements are reviewed by a panel which involves the software team lead and software engineers. Requirement specifications are delivered to the software teams in textual form with some supporting UML diagrams to help the engineers understand the requirements. Requirements are managed through the IBM requirements management application, DOORS. The interview participants reported that requirements analysis and decomposition is a challenge and depends on an engineer's familiarity and experience with the nature of task to be performed well. There is no typical number of requirements for a phase. The average number of requirements per iteration is unknown because it depends upon the amount of work required to meet a particular requirement, due to the unequal size of requirements.

One software team had experimented with converting requirements into more formal structured text. However, one participant (P1) reported that this turned out to be a "*disaster.*" According to P1, the customer reported their displeasure with the transformed requirements because they were less readable than the original.

The interview participants reported that requirements change was experienced in all projects. One participant estimated that 10% of the requirements changed throughout the software lifecycle. Changes were reported due to a variety of sources, including requests from customers, the discovery of conflicts between the architecture and requirements during implementation or the need for further requirements elaboration or additional scope. The need for a change in the requirements can be discovered at any stage from requirements analysis to delivery. Participants also reported that the discovery of requirements changes often necessitated rework or coordination with other teams in the project to assess impact, particularly the project's systems team. It was also observed that requirements tended to stabilize towards the end of the project.

### *4.5. Product Integration and Certification*

Integration and certification is performed iteratively, beginning within the software team, before an entire product release is provided to the customer. Certification occurs when a *formal release* is due to be delivered to the customer. Also, an integral part of the integration process is the preparation of supplementary documentation to support certification processes. This documentation includes requirements specifications, risk management plans, accomplishment summaries, release information and high level and subsystem design documents.

Software teams manage all their documentation and design models locally using the Serena Dimensions configuration management tool and generally only have visibility of other teams' documentation during the integration and certification process. Documentation is reviewed whenever a significant change is made as well as during the certification process. Documentation is formally reviewed during a lifecycle in the appropriate phase. For example, test reports will be reviewed in testing.

More recently, projects have used a practice of delivering *engineering releases* as well as the end of phase *formal* releases. Although these are releases that are provided to the customer, they are done so in order to generate feedback and do not undergo the whole certification process.

The participants reported that some visibility of progress is lost during the integration process. This happens because during the integration process there are many other ways of tracking progress, and it is possible that software team members do not update internal issue tracking (such as Jira) because this creates duplication of work. Moreover, if a problem arises in integration, it is recorded via a project wide defects recording tool, and the respective software team involves the people they need immediately in the task. Thus the benefits of internal progress tracking within the team are lost during integration.

## 5. Use of Agile Software Development

This section discusses the extent to which the company has so far used agile practices, building upon Section 4 to address Research Question 1. Each team has some flexibility in choice of software process, depending on the nature of the overall project, with the final selection of lifecycle being made by a team's lead software engineer. The company has developed a series of questions that guide for the selection of a software process. Historically, teams have typically employed Waterfall or an iterative process because of the duration of the projects.

Two of the interviewees had previously worked in software teams that employed agile methods. In their current projects one participant had also begun employing elements of Scrum, several months prior to the interviews. Several motivations for this were given during the course of the interviews:

- The need to speed up delivery times and produce a series of phased releases for the customer. The second team reported that this goal had not been reached yet, although the first team found employing aspects of agile methods had resulted in significant benefits. One participant (P5) commented that they wanted to be "*...giving the customer many* *more releases"*. Another participant (P3) with no experience of using agile software development, while expressing his expectation from its adoption, emphasized the need to deliver more frequently "*...we would be able to provide the customer with more frequent deliveries of the software"*.

- Improving communication within the software team. One interviewee (P1) reported that "*...we wanted more visibility in the project i.e. who is doing what?, how many tasks have been completed?, estimates, performance and list of completed jobs etc."* Tools like Jira Kanban boards were reported as helpful in this regard.

- Improving team member engagement with the coordination of the software project. Freedom to select one's own tasks has prompted a sense of responsibility among team members. While expressing benefits of using Scrum, a participant (P1) said "*The level of engagement of some of my engineers is much better... That is the massive difference, my teams are working much better."*

- Earlier discovery of problems. The interviewees reported that problems were often discovered late during integration, requiring more costly rework. One participant (P1) while talking about reasons of adopting agile software development commented "*...not letting things get too far before realising its gone wrong. It's that visibility thing. It's about knowing about problems sooner"*. Another participant (P3) who did not have any experience of using agile software development, while discussing the reasons seen for using agile in other projects, said "*...so we get feedback earlier."*

The Scrum method itself had been selected by these teams for this part of the process because it was perceived as the de facto industry standard, and within the scope of a sub-team, did not require senior management support to allow the experiment. At the time when interviews were being conducted, the organisation had not undertaken significant Scrum training for its personnel. Rather, individual teams had chosen to adopt agile methods and practices within their own parts of a wider project.

Each team has a scrum master responsible for coordination of activity. Project planning is organised into a series of *sprints* with associated planned releases, with each sprint typically lasting one or two weeks. The team creates a plan at the start of each sprint, using a Jira or Kanban issue board to track progress. The scrum master begins by calculating the available effort in terms of *story points* in the sprint based on team size and availability. The teams do a "*T-shirt size*" estimation of the tasks and record this on Jira boards. Numerical information is extracted with the help of a formula from T-shirt estimation and entered into a Microsoft Project plan for long term planning. Items from the backlog are then selected for completion and allocated to the sprint.

The interviewees reported that the first and second teams follow the daily stand-up ceremony to facilitate communication. In the second software team, the lead software engineer acts as the product owner, so the team also conducts customer demonstrations. However, the first software team does not have customer demonstrations because they do not have a product owner within the team. Customer demonstrations are also interpreted as something that induces a sense of failure or inability to finish the task on time, by the first software team we interviewed. According to the first software team lead (P1) "*...at times the teams don't feel failure, and I know that meeting (customer demonstration) helps with the feeling of failure, which would be nice sometimes...It helps with building reasonable pressure on the team member.*" The interviewee suggested that the first software team lead would rather have a 'mock' meeting with another internal team than the external customer because they have a very formal relationship with the customer with whom the team feels unable to discuss delays. Neither of the two software teams currently conduct retrospectives. It was stated that the first team does not see the value in it because they are already monitoring progress through Jira boards. Therefore they do not see the need of having a separate meeting for looking at previous performance. Separately, the second team reported that they had experimented with retrospectives. They reported finding the number of potential process improvements to consider to be overwhelming and so had abandoned them until additional Scrum training could be completed.

The team members are encouraged to communicate and help each other, and use this as a means of learning. Pair programming is viewed only as a form of mentoring in the organisation and people have different opinions about it. Pair programming is found to be ineffective and a time wasting activity by one of the participants because it has been observed that the weak member does not learn from it, and mostly, the stronger member takes the keyboard. According to the lead software engineer (P1) "*...someone always takes a back seat while the stronger member takes the keyboard.*" Others take pair programming purely as a way to "*help each other out.*"

Despite the perceived benefits, participants P1 and P2 also reported drawbacks of employing agile software development. First, the team has discovered that applying an agile philosophy to design is creating rework, because short term design decisions are later discovered to be incompatible with the overall product design, "*...this is because the teams think, since they are working agile, they are concerned (only) with the part they are working on.*" The Lead Software Engineer (P1) for the first team further suggested that the team needs some "*forward thinking*", for example, to anticipate the need for extension points in design. The first software lead engineer felt that the Waterfall process helps with this issue by encouraging a more holistic approach to design.

The third team was using a Waterfall process, rather than an agile method. When we asked them if they would consider applying agile software development in the future, several reasons were given for not doing so. First, the third team believed that agile methods and practices are not suitable for projects where requirements are uncertain or volatile. One of the Lead Software Engineers (P4) said "*I think one of the reasons we've not gone agile is experience. You know we're experienced with the lifecycles that we follow.*" This was a common reply from the team members who were reluctant to use agile software development. The project they worked on was for a new product, but they based their software process on that for a long standing (more than 20 years) project within the company, "*...it used fairly similar processes all the way through. So for us on the new product it made sense to stick with the non-risky strategy of going with what we'd done previously. We know that process works, we know, what we're going to get out of it (P4).*" In particular, the team believed that the requirements for the project were relatively well understood and stable, so the team was able to plan Waterfall phases of 9 - 12 months duration, "*So I guess it was a sort of macro-agile process... but the sprints were just incredibly long...But each of those was separate in a way (P4).*"

Second, the third team believed that applying an agile method only within the software team would create difficulties for coordination with the other teams in the project (hardware, firmware, integration etc.). A software team does not work in isolation as said by one of the participants (P4) "*...we do work very closely with the systems team to define our requirements, we work very closely with the hardware and firmware teams to integrate software, so those, you know would all need to be working to the same schedule, and the* *same set of sprints."* Consequently, there was concern that applying an agile method and practices within the software team would complicate this as different teams work at their own pace and the schedules between different teams often mismatch, "*...they may not be working to the same schedule as we are...that's not something we're very good at (P4)."*

More widely, the third team believed that the company as a whole lacks guidance on how to adopt agile software development when developing software for safety-critical systems and are uncertain about the suitability, "*...there's always been a fear of certification... and how an agile development would affect that?."* The Lead Software Engineer (P4) for the third team also pointed towards the need to change the mindset of the people saying "*...within the business there is a fear or a concern that doing something an agile way means doing it in a scrappy way,.. you know or doing it in a careless way."*

These concerns were also reflected in the experiences of the first two teams in employing agile software development. The participants from these two teams reported both internal and external obstacles, both in convincing team members of the benefits of change and in engaging with the project customer. They found that the application of an agile method was constrained by the customer's desire for a form of contract that encouraged a plan-driven software process. For example, the requirements phase is associated with a milestone in the contract for delivering a full requirements specification before design and implementation work proceeds. Further, the participants believed that the regulatory framework also dictated a plan driven process. Both software team leads argued that "*...regulatory standards do not let us choose our own method (P1)."* In addition, these regulatory standards require production of a lot of documentation.

As a consequence, both participants that had experience of agile software development picked up the parts of Scrum and agile practices that they thought were beneficial and could be applied without conflicting with regulatory standards of project contracts. These included the use of Kanban boards in Jira, sprints, daily stand-ups, sprint planning and product backlogs. Further, both teams anticipated employing more agile practices, such as the specification of requirements as user stories, in the future. Conversely, both software teams wanted to re-instate the Gate Reviews they were conducting while using Waterfall but they had not reached an agreement yet on how to do this within their agile software development process. Both these teams expected that employing agile software development would enable them to deliver smaller, incremental improvements of the overall system more frequently to the customer over the lifecycle of the project, compared with their existing plan-driven process.

## 6. Discussion of Challenges

This section discusses the challenges in implementing agile software development within the Company, building upon Section 4 and 5 to address Research Question 2. Figure 5 presents the key challenges elicited during the interviews. We discuss these challenges under the distinct themes of Pressure for Waterfall, Coordination amongst Stakeholders, Documentation Demands and Cultural Challenges, below. For each theme, we present and discuss extracts from our interview transcripts where relevant observations of interest are made. For each theme, we also identify relevant literature and discuss the implications of the findings.

### *6.1. Pressure for Waterfall (Challenges 1, 2, 3, 4, 5)*

Challenges 1, 2 and 3 reflect the difficulties of implementing agile software development in a wider software development culture where the Waterfall process has become embedded. All the participants except one said that regulatory standards are one of the main hurdles in use of agile software development. They anticipated that Waterfall imposed by standards would prevent the use of an agile method (Challenge 2). For example, "*Our standard says that we use waterfall... it doesn't say that we can pick our method (P1)*" Further, the participants stated that the company's internal standard, which conforms with DO-178C prevents the use of agile methods, and that customers are also wary of such an approach. However, the other participant (P4) argued that "*...No, I don't think there are any conflicts...I can't see really why it would be a problem.*"

Reflecting on the emphasis on Waterfall in the standards, the participants also reported that the use of Waterfall is often mandated by the (external) customer which restricts them from using agile software development (Challenge 3). Within the company, contractual agreements are the primary driving force of a project. Plans, milestones, term, and conditions of a project greatly impact the development lifecycle of a project, "*the customer is saying, we want you to use waterfall... because of the way we get a set of contractual requirements and we must complete all those contractual requirements, rather than create a set of requirements then cut dead at a certain point (P1).*" This perspective reflects the culture within safety-critical systems development of

| | **Challenge** |
|---|---|
| 1 | Agile software development advocates incremental design, but safety standards require upfront design as necessary input for hazard analysis. |
| 2 | Regulatory standards are perceived as mandating Waterfall and not permitting agile methods. |
| 3 | The prevalence of fixed price contracts for pre-agreed requirements in safety critical systems projects is not readily compatible with agile software development. |
| 4 | The actual time taken to complete the tasks always turns out to be more than it is estimated in the beginning, particularly due to integration complexity in safety-critical systems projects. |
| 5† | Requirements are difficult to modularise in safety-critical projects because the functionalities are so interdependent that it is very hard to separate them. |
| 6 | Software teams lose visibility during the integration phase. Agile methods lack guidance on integration with hardware. |
| 7 | There is a complex network of customers that obstructs agile ceremonies such as the Sprint Review |
| 8 | Face-to-face informal contacts dominate communication, causing project related information to be lost. |
| 9 | Software, hardware, firmware and other teams in safety-critical systems work function independently according to their own schedule causing plans to become mismatched. |
| 10 | Frequent releases increase overheads and costs, because they must be accompanied by supplemental documentation to achieve certification. |
| 11 | The Software team has no practical example to follow for applying agile methods and they lack the resources to experiment. |
| 12† | The teams need guidance on how to scale agile methods for use in large multi team context. |
| 13† | The organisational mindset require convincing about the benefits of agile software development. |
| 14‡ | Independent testing required by standards conflicts with the practice of developer created tests advocated by agile software development. |

Figure 5: Summary of challenges identified during the research. Unmarked challenges were discovered during the semi-structured interviews and confirmed in the validation workshop. Challenges marked † were discovered within the validation workshop. The challenge marked ‡ was discovered during the semi-structured interviews, but rejected during the validation workshop. All challenges (including 14) are reported for completeness.

defining the full requirements at the beginning of the project because of the need to understand the full features of the software, and how it will integrate with the hardware. As a consequence, most participants believed that use of agile software development in full was not practical because this would require a different relationship with the customer in which requirements were continually refined and renegotiated at the beginning of each sprint or release.

In the literature, VanderLeest and Buter (2009) argue that "*Contractual models in aerospace expect firm-fixed estimates of large complex projects with little room for change. The agile approach of using client-driven adaptive planning at the start of each iteration faces the hurdle of dealing with the potential contractual changes that result from such frequent planning.*" Limited support for subcontracting is a connected limitation of agile software development reported by Turk et al. (2014). Sub-contracted tasks are usually well defined, and the milestones are clearly laid out (Turk et al., 2014) which already gives a limited freedom to the development team and the remaining "*flexibility*" is constrained by regulatory standards.

There are mixed opinions about use of agile software development for software development for safety-critical systems in the literature. For example, VanderLeest and Buter (2009), Cawley et al. (2010) and Wils et al. (2006) all argue that DO-178C does not favour a particular software development lifecycle, but rather provides process guidelines and (in total 71) objectives for development of airborne software (Coe and Kulick, 2013). Wils et al. (2006) argued that a reasonable re-interpretation of agile principles would mean they are compatible with certification. In particular, Wils et al. contend that working software in this context comprises both the implementation and the documentation, because the documentation is necessary for the software to be certified as safe to enable use.

Conversely, Winningham et al. (2015) argue that agile methods and practices are not developed for safety-critical systems. In order to be used for safety-critical systems such as avionics, the software process has to conform to process standards i.e. DO-178C in the context of the current study (RTCA). Several authors have identified and discussed specific conflicts. Relevant to our work, agile principles discourage the development of detailed designs that anticipate future requirements prior to implementation work. Beck and Andres (2005), for example, allude to the 'you ain't gonna need it principle' and argue that the expectation of requirements change means that any effort dedicated to design for future implementation could well be wasted. However, Chapman (2016), Chapman et al. (2017), Cawley et al. (2010), Wils et al. (2006), Chenu (2009), Glas and Ziemer (2009), Boehm and Turner (2003) and Coe and Kulick (2013) all contend that this principle conflicts with most safety-critical standards that mandate the development of a sufficiently detailed design to act as input to certification processes. Changes to the design may invalidate the certification status of the product and require an extensive rework of assurance related artifacts.

One particular impact of this emphasis on Waterfall reported by participants is the extent of detailed requirements analysis, specification and design that takes place, before implementation work proceeds (Challenge 1). These processes are accompanied by gate reviews to evaluate the quality of work before permitting a project to proceed to the next phase. Our participants said, for example "*The design itself, we tend to come up with fairly stable architectural designs quite early on... Specifically because we don't want to be changing them all the time (P4).*"

This issue was explored further with the participants. During discussion, it emerged that several of the participants *preferred* to engage in substantial upfront design, regardless of the constraints imposed by the standard. This preference was justified by the scale of the system development and the need to accommodate future planned features within the existing design, "*I think you need to be forward thinking as to what your design needs to be (P1).*" One of the participants also stated that adopting an agile approach to design increased costs of this aspect of the work overall because the team did not design with future requirements in mind and so created substantial additional rework, "*What I guess was not anticipated was the amount of rework that agile is creating for me... I think you need to be forward thinking as to what your design needs to be (P1).*" The participant goes on to explain that this anticipatory design is necessary because of the interdependence between the different teams in the overall project. The team needs to be aware of the expectations of other teams on the software they are working on and anticipate this in the design.

Despite the preference for upfront design, all of the participants noted the tendency for the software projects to undergo substantial requirements and consequent design changes once implementation begins, with estimates ranging from between 10% and 20% although one participant estimated that *deviations* from the original plan could reach 80%. Our participants also reported that these changes could come from the customer or from the software process itself (such as the need for further elaboration) and occur throughout the software process.

VanderLeest and Buter (2009) quote findings from different studies suggesting that a typical project may experience 25% change in requirements, increasing to 35% for a a large project. These estimates suggest that there is considerable variability within 'safety-critical' projects as to the degree of certainty in the project requirements and plan, and thus the feasibilty of applying a plan-driven process. On the one hand, the extent of volatility in requirements for safety-critical systems suggests that adopting an agile method or practices would be appropriate for requirements engineering in this context. However, there is a need to understand how agile methods and practices can be adapted to accommodate the need for continual certification against standards. As discussed above, SafeScrum (Stålhane et al., 2012) is an indication of the interest in this area. There is also a need to extend agile methods and practices to mitigate changing requirements across software, hardware and other developments, as discussed concerning Challenge 6 below.

Related to this, one participant in the validation phase workshop identified a further challenge with the modularisation of requirements (Challenge 5), stating "*We get over 8000 pages of requirements and it becomes really difficult for us to isolate a sub-set of requirements from a big pool of requirements (Validation Workshop).*" The sheer amount of detail in the fully elaborated requirements document makes it difficult for the software team to allocate packages of functionality to the different sub-teams. Later design and implementation work reveals interdependencies between functions that were not anticipated during the requirements analysis phase. This requirements complexity would appear to be a significant challenge for the implementation of agile software development, since requirements cannot readily be divided into modular, manageable features.

As a result of these constraints, the participants reported that they feel the time and amount of work needed is nearly always underestimated, and that delays occurred due to the additional effort needed to better understand or implement altered requirements (Challenge 4). One participant (P3) commented, "*there is a high level of, what we call punt... in our system level requirements which then obviously impacts us downstream.*" The fixed price approach to contracts and the estimation process does not anticipate this cost of change. In particular, one participant noted that even though change occurs in project requirements or plans, due to requests by the customer, this does not always get integrated into the estimates for the overall plan "*...but a lot of this time, changes come in that are not considered (P3).*" However, in some cases, the participants did report being able to rely on historical data from previous projects to produce reliable work estimates, "*it was a fairly mature, although [it]'s a new [product], our ... product line is very mature, you know. So the requirements, eighty percent of them probably were very well understood at the beginning of the project (P4).*"

Similar challenges have been identified in the literature. For example, Wils et al. (2006) reported the finding of their study of implementing XP, conducted at Barco (a major Belgian avionics equipment supplier). The company employed XP in order to reduce time-to-market and respond quickly to change in requirements. However, during the study, it was found that the software project was dependent upon external factors that were hard to control, such as delays in automated testing and mismatched hardware development schedules.

Large systems engineering projects often depend on significant upfront design as a means of coordinating effort between different sub-teams working on software, firmware and hardware elements (Chapman, 2016; Chapman et al., 2017). In addition, requirements and design documentation serve as inputs for hazard analysis and other safety certification processes which begin while software implementation is still underway. For example, DO-178B/C requires early completion and approval of Plan for Software Aspects of Certifications (PSAC). Later changes are difficult because the PSAC has to be updated and re-approved (VanderLeest and Buter, 2009). Therefore, some level of detailed design documentation is required for this purpose.

However, many regulatory standards, such as DO-178C (RTCA) do not prevent changes to software design because having a rigid upfront system design that cannot be revisited and changed is unrealistic. The concern here then is how much upfront design do is needed and how much change to a design can be accommodated by safety analysis processes. Ge et al. (2010) demonstrate that design can be simple but detailed enough to allow preliminary hazard analysis. Ge et al. have used the term "*sufficient design*" to refer to the level of detail in the initial design without explaining the minimum level of detail needed to conduct preliminary hazard analysis. Critically, there is a need to develop a design process that copes with both evolution and satisfies the needs of existing hazard analysis techniques, or develop a hazard analysis technique that copes with evolutionary design.

Advocates of agile software development, such as Beck and Andres (2005), advise against undertaking detailed software design work prior to implementation, arguing that without sufficient information about the problem domain and associated constraints, any proposed designs will be subject to change once implementation begins. One potential direction to address this problem may be to extend the practice of system metaphor definition in the XP agile method to encompass the need for some *anticipatory* design desired by the participants.

Despite these challenges, the participants reported considerable experience experimenting with agile software development, making adaptations to fit their needs. Winningham et al. (2015) note that agile methods were not developed for safety-critical systems and that consequently, many practices within agile methods need to be compliant with standards, such as DO-178C (RTCA). Coe and Kulick (2013); Boehm (2002); Boehm and Turner (2003) suggest that methods such as Agile-Planned that combine elements of both philosophies show promise in this context. This selection and adaptation of elements was reported by the participants. As one of the participants (P1) described it "*We follow some bits of agile that are of interest to us..*" The participants reported that their teams participated in a variety of Scrum 'ceremonies' including sprint planning, daily standups, customer demonstrations and retrospectives, although all participants reported adaptations, or the non-use of a ceremony, which we examined further.

In particular, two of the participants reported conducting frequent retrospectives, reflecting the use of Scrum within their teams, whereas, the other two participants reported undertaking less frequent "*lessons learned*" within their projects, typically following the delivery of a release to the customer. When discussing the practicality of employing retrospectives, one participant (P2) noted the difficulty of making frequent change to their software process, due to the risk that a change to the process might be disruptive, "*we've got pretty fluent software development delivery system...we're being encouraged to stick to schedule...it would be unwise to inject too many silly ideas into how to change that at this point in time. So we also encourage people to, like, to sort of like story-board their ideas and just to put them to the side.*" Instead, the participants reported collecting ideas for changes to the software process (on a Trello board, for example) that could be reviewed at less frequent meetings. This practice shows the company adapting agile practices to match the tempo of a safety-critical project, and avoiding the risks of frequent small changes.

Several of the participants reported extensive use of quality assurance associated with agile software development, including automated static analysis, refactoring, automated unit testing, test driven development, code review and pair programming. Automated static analysis in particular was used extensively within the company. One participant (P2) confirmed that a key goal of employing static analysis was to achieve conformance with MISRA C standards "*it's for MISRA, I think level coding standards.*" In a follow up discussion, it was revealed that the company had found that the application of static analysis within a continuous integration pipeline had transferred well to an agile software development approach without the need for adaptation. In fact, the transition had led to enhanced benefit from the use of static analysis. The teams found that applying the tooling more frequently led to the production of reports with fewer but more meaningful warnings, "*As the delivery frequency increased...As the maturity of the product became higher, the easier it was to run static analysis as large swathes of code were unchanged from delivery to delivery. (P5)*"

In other cases, these practices were adapted to fit within the constraints of safety-critical system development when appropriate. In the case of pair programming, two of the participants were very emphatic that they did not practice pair programming despite all the participants reporting that informal mentoring of newer members of the company was strongly encouraged. One of the participants (P1) made the distinction between pair programming and mentoring, "*... I think that it's much better giving people a little bit of help and then dropping back and then reviewing their changes and giving them some feedback but making them do the task. Really to use the adage teach someone to fish so that the next time they can fish. There is always a ...when you do pair programming, there is always a stronger member and they will always take the keyboard... and that's not what you want.*" One participant (P2) suggested that the "*demographics*" of the company was partly a cause of this approach. Many employees have worked for the company for considerable periods of time and have become experts in particular domains of the development work. Therefore, the participants felt that these engineers would not benefit from pair programming with a younger graduate, but that the graduate would benefit from a mixture of demonstration and peer review. As one participant (P1) described it, "*I think it wastes budget. I don't think we get the value from that task.*"

A final challenge within this theme was identified during the semi-structured interviews concerning quality assurance practices within the company (Challenge 14). Safety-critical standards, such as DO-178C advocate or even require the use of independent teams to develop test procedures. However, agile methods, such as XP advocate the development of tests by the development team themselves, partly as a form of documentation of the application software (Beck and Andres, 2005). When we investigated this conflict with the participants, a complex picture emerged, with some participants contending that this conflict was "*an ongoing problem. No, I don't think we have eliminated it. (P1)*" However, different perspectives amongst the team and within the validation workshop ultimately led to this Challenge being rejected by the participants, because the established compromise described below was considered to be sufficient. However, we report the discussion from the interviews for completeness.

The issue emerged when one of the participant stated that the DO-178C standard they were working towards did not require complete independence, instead, the testing procedures are independently *witnessed*, "*We satisfy that by having all of our v & v witnessed or signed off by our QA people. So, all of our document reviews and things like that would have input from the QA department. All of testing is actually witnessed, you know we have someone sitting there writing things down, so that that gives us our independence. (P1)*" However, the participants also recognised that this situation is the result of a tension between the desire for independence of testing and the need to have domain expertise concerning the software under development in order to test it effectively, "*it's an interesting tension there, between needing to know exactly the details of the component you're testing. (P2)*". What emerged from the following discussion was that the deliberate physical distance of the QA team to ensure independence had made it very difficult for them to gain a sufficient understanding of the system to develop effective tests "*There was too much inherent knowledge that the guys in these teams have about the internals of the software. (P2)*". One possible avenue here, proposed by the participants was a compromise in which the QA team remained independent, but engaged in closer cooperative work with the development team, "*[if] we had got a v & v team in much earlier it would have worked a lot better.* "

### *6.2. Coordination amongst Stakeholders (Challenges 6, 7, 9)*

The participants reported several aspects of the software team's work specific to safety-critical software development connected with coordination with external (to the software team) stakeholders that presented challenges to the use of agile software development (Challenges 6, 7 and 9). Agile principles emphasise the close involvement of an identifiable customer as critical to a project success (Chapman, 2016; Chapman et al., 2017). Providing the team with ready access to the customer enables better communication, allowing uncertainties with regards to requirements and design to be resolved more quickly (Schwaber and Beedle, 2001). However, the projects reported by the interview participants experience a far more complex relationship with the project customers (Challenge 7). The participants described various customer structures, for example, "*joint systems team meeting ... happens on a sort of two monthly basis .... And that involves our direct customers and members of.. their direct customers (P3)*" and "*...there's certain customers could be viewed as being the END USER they are the end users. They are type of customers. But then there are people who are little bit closer like COMPANY, then we get little bit closer again.. which are the people who are involved as product owners (P1).*" From the perspective of a software team, the immediate customer is the project's systems team who allocates the requirements. The whole project may have several different customers, each with slightly different needs. These customers may, in turn, be procuring the product as a component to be integrated into one or more larger systems for their own customers. This network of stakeholders is characteristic of safety-critical systems projects, and so "*Agile use in these environments is restricted by the elements that define these environments*" (Hajou et al., 2014). However, from the participant's perspective, the influence of external customers is difficult to manage, because they only have direct access to the systems team in order to demonstrate their work and receive feedback "*For me, I would say that customer demonstration ... would be the demonstration of how things work when it gets to the TEST ENVIRONMENT (P2).*"

Chapman (2016) and Chapman et al. (2017) notes that requirements engineering in agile software development is dependent on close customer involvement in the project to the extent that the customer may be viewed as an additional member of the project team. However, as Chapman et al. (2017) notes, this may not be practical in the scenario described above, where there are many different types of customers with different perspectives on and commitments to the project, such as procurers, end users, industry regulators and independent auditors. Ensuring close involvement of a larger number of customers on an on-going basis is difficult due to practical considerations such as time availability. In addition, these customers may have very different views on the requirements for the project, but there is very little guidance available on decision making, where the customer relationship is inevitably more complex (Chapman, 2016). One possibility is the suggestion by Paige et al. (2011) to use a "*Stakeholder consortium*" to mitigate this problem. However, Chapman (2016) and Chapman et al. (2017) suggest that achieving consensus within the consortium may not be practical and that establishing *"rules of engagement"* and use of tools to automate communication and documentation can counter this problem.

The other teams within the overall project are all also effectively external stakeholders for the software team and coordination here also presents challenges. The different teams within the overall project have their own pace of completing tasks (Challenge 9). Deadlines and milestones are defined in the contracts for the whole project, but individual teams choose their own development lifecycles within this framework, creating a *"silo effect"* (VanderLeest and Buter, 2009). Members of the software teams interviewed report being unaware of the details of activities and current status of tasks in other teams. Participants also reported that schedules across teams often do not match. For example, *"they're working on their own bunch of things at their own priorities, with their own pace dictated by the number of resources that those have, and it's often when it gets to the point where the crunch is coming that we start to understand that we've, we're misaligned in terms of priority (P2)."* The teams also have their own interpretation of when tasks are considered complete, as one participant (P2) observed, *"when I say hardware guys I mean the guys who produce the actual circuits, and then the firmware guys who bring that to life so we can use it for software development. Their definition of what finished is, so that we can put the capability of software on it, tends to be separate from what we think the done thing is."*

In early work in the field of Global Software Engineering, Herbsleb and Mockus (2003) recognised the challenges of coordinating work across loosely coupled or distributed sub-teams. These challenges remain an active area for Software Engineering research, as illustrated by the recent study by Ebert et al. (2016). Turk et al. (2014) suggests frequent and informal communication to overcome the lack of visibility but informal and face to face communication poses a risk of important project related information getting lost. VanderLeest and Buter (2009) also emphasize the importance of tools to improve communication and coordination among teams. In our case study, one participant described a project where all the teams were compelled to strictly follow the same schedule using a single Microsoft Project plan. According to the lead software engineer interviewed, this approach worked well. However, it is unclear whether this approach can be imposed on all projects in the company.

The lack of visibility also causes problems at integration between software and hardware (Challenge 6), a challenge that Stelzmann argues is characteristic of safety-critical system developments (Stelzmann, 2011). Integration between hardware and software is often done towards the end of a project release, due to the components only being available at this stage. All participants agreed that this arrangement caused problems, "*typically when we get to integration. We'll find that something that the hardware is doing either isn't as we understood it to be, or it's not working (P2).*" Although the allocation of tasks and designs is well understood by the different teams at the start of the release, it was difficult for the team members to stay up to date with "*what is happening in other teams.*" One participant (P1) said "*Once we get into the integration phase, we found that the boards don't always stay up to date.*" Several interview participants suggested this was because different teams run their own development lifecycles, for example "*We've got software people working in the software plan and hardware people and firmware people working in the firmware plan. So it often becomes dissected. (P2).*" Due to the late-stage integration, it was suggested that a software team tends to focus predominantly on their own tasks, and so lose visibility of changes that are occurring elsewhere in the project. This phenomenon affected both the team that followed Waterfall and the team that had recently employed aspects of agile software development. One participant (P1) also reported that the benefits of employing a Kanban board in Jira had been lost once the project moved to an integration phase, as other tools were used for tracking progress on integration "*Once we get into the integration phase, we found that the boards don't always stay up to date... I believe, that the reason for that is we have got other methods of tracking our problems and the guys see it as duplication.*"

To partly address this challenge, one of the participants (P4) described how they had adapted their software process to incorporate a weekly *integration meeting* during the integration phase of the project, "*during our integration process, you know a lot of people had to work quite closely together so we were having weekly meetings. Once we got through that process they stopped becoming useful.*" As described above, this demonstrates how the company is employing the principles of agile, such as frequent informal communication, but adapting the specific practices to fit with the needs of safety-critical system development. The weekly integration meeting allowed issues to be aired and resolved frequently between the different sub-teams, in a similar way to a product planning meeting within a single team.

The difficulties in employing continuous integration are also reported in the literature. Jamissen (2012) argues that DO-178C does not conflict with the concept of continuous integration in agile software development, however, Ge et al. (2010) and Kaisti et al. (2013) note that continuous integration of embedded systems is challenging. Kaisti et al. (2013) report a scarcity of evidence on the use of continuous integration in embedded systems. According to Douglass (2016), most of the literature concerning agile software development is focused on software application development, not embedded systems. This lack of guidance on Hardware and Software co-development and integration is recognized by many researchers (Chapman, 2016; Chapman et al., 2017; Kaisti et al., 2013; Douglass, 2016). For example, in their study, Wils et al. (2006) found that the software-hardware integration phase inevitably slows down development efforts. This stage is also where the discovery of required changes can frequently arise and be the most problematic.

One proposal in the literature is to use simulators and emulators to help reduce problems at integration (Ard et al., 2014; Schooenderwoert and Morsicato, 2004; VanderLeest and Buter, 2009). A key challenge in this approach is to ensure that emulators, simulators or test equipment have the exact specification of the target equipment (Ard et al., 2014). While testing a system using emulators, changes made to software and hardware should also be kept in mind (Ard et al., 2014). All the interview participants told us that the equipment for testing is not updated and often its specification does not match the target hardware. This suggests that there is a challenge in maintaining up to date test harness implementations.

### *6.3. Documentation and Communication (Challenge 8, 10)*

Two related challenges were reported by participants concerning the use of agile documentation and communication practices. The Agile Manifesto (Beck et al., 2001) advocates the delivery of "*working software over comprehensive documentation.*" Several authors have argued that this principle makes agile software development incompatible with the development of software with certification requirements (Ramesh et al., 2010; Turk et al., 2005; Martins and Gorschek, 2016; Rayside et al., 2009). This conflict was reflected in the interviews, with one participant (P3) commenting that "*..the process documentation that we have at the moment doesn't adhere to agile sort of development process*" (Challenge 10). Critically, certification standards for safety-critical systems (DO-178C, for example) mandate the generation of documentation to demonstrate that both the delivered product and development process conform with standards and is safe to use. Certification is a very expensive and time consuming activity since it is performed on the complete system for delivery, as one participant (P1) described "*the standards require us sometimes on producing a hell of a lot of documentation... a lot of overhead in that respect.*" Certifying the system each time a change had been made would be prohibitively expensive, so the company normally only certifies the system for each "*formal delivery*" to the customer. Participants also identified the need for maintenance of documentation as a cause of delays in the project schedule, having an additional impact on Challenges 4 and 5 discussed above.

Despite this apparent conflict, there are a number of studies which demonstrate the use of agile software development in the development of formal specifications, for example, Rayside et al. (2009); Black et al. (2009). Several of these authors emphasise on the need to adapt agile methods and practices according to the need of safety-critical system development. For example, Rayside et al. (2009) argue that traditional and agile methods are separated by limitations of current technology rather than by fundamental intellectual differences. They believe that the use of a "*mixed interpreter that executes mixed programs, comprising both declarative specification statements and regular imperative statements*" (Rayside et al., 2009) can mitigate many of the problems. Black et al. (2009) suggest that if requirements can be expressed in a formal notation they can then be machine checked for inconsistencies, effectively extending the the automation of quality assurance processes to requirements documentation, in a similar manner to the Behaviour Driven Development practice (North, 2006).

The company in the current study has also adapted its practice with respect to certification to achieve more frequent deliveries. The participants reported having employed a practice of making non-certified intermediate deliveries available to the customer, called "*engineering deliveries or releases.*" One participant (P3) stated "*we have moved to the philosophy of ... there would be all engineering releases and at certain points in development we would take an engineering release and do the formalities on it.*" An advantage of this approach is that the customer is able to begin integrating the product into their own system development efforts earlier. A subsidiary benefit is that the engineering releases do not require the demonstration of quality assurance processes demanded by many safety-critical standards (Chapman, 2016; Chapman et al., 2017; Cawley et al., 2010; Boehm and Turner, 2003; Vuori, 2011). As one participant (P3) stated, "*certainly when we come to formal release if you like... that's where our testing level moves up.*" Another potential option to mitigate the costs of document production is the use of automated techniques, which can reduce delay (Chapman, 2016; Chapman et al., 2017). In addition, the approach implies that there is an *expectation* that an engineering release may eventually become a formal release, which as a consequence imposes the quality assurance standards for developing a formal release, but without the accompanying documentation to demonstrate it.

Similarly, agile software development advocates frequent face-to-face communication in small groups to ensure that critical information is circulated effectively. However, it is well understood that this approach does not necessarily scale effectively to larger multi-team projects with different lifecycles and cultures (Challenge 8). In particular, communication in agile software development is reliant on the retention of tacit knowledge, which can be difficult to recover in large-scale projects (Boehm, 2002; Glas and Ziemer, 2009; Ramesh et al., 2010). We discussed the challenge of managing communication in large scale projects with the participants and a number of different perspectives were identified. The participants reported that a mixture of approaches to documenting information were taken, with some teams relying predominantly on an informal approach, "*I would say that large majority of them are not recorded. There is very few... where in the meeting someone minutes the meeting. (P3)"*, whereas, others stated that formal documentation was used extensively for communiciation, either through email or design documents, "*know have a face to face chat and then email out the outcome of that discussion and any action points, what was agreed, and distribute that to the rest of the team (P4)."*

Several of the participants stated that an informal approach had led to mis-communications, with one participant (P2) suggesting for example, that the informal communications needed to be 'snooped' on to ensure the information wasn't lost "*we could get someone to snoop the conversations, and figure out how much we lost."* However, another participant (P4) reported that the project teams could often rely on the tacit knowledge of individual members because of the demographics of the company. "*I don't think we really suffered as a result of that. Because we had a good group of people and a lot of very experienced people. If it was a less mature project with you know, less experienced engineers then I think it would have been a problem."* These two different perspectives illustrate the need to not just adapt agile practices to safety-critical systems, but to adapt them to the specific context of the project.

There was some discussion about the impact that adopting agile software development had on this problem. One participant (P2) commented that "*I'm not, at the moment I should be at the ten o' clock stand-up in the roof lab. If someone doesn't come and tell me what happened or what I'm meant to do or any of the other information then that could be lost.*" However another participant (P1) described how agile software development had assisted in retaining some aspects of information that might otherwise be lost because the team became *more* disciplined about recording information in the project team's tracking tool "*Since we have employed the boards and they understand more about what's going on.*" Again, this suggests that there is potential for agile software development to be adapted to allow teams working on large scale, safety-critical systems projects to identify and maintain the documentation that is valuable to them.

### *6.4. Cultural Challenges (11, 12, 13)*

The final theme which emerged from the interviews was the need to change the culture within the company. Three particular challenges emerged in this context. First, the software teams in the company had no prior experience of using agile software development on a large scale and lacked guidance from elsewhere in the literature (Challenges 11 and 12). At the moment, software teams are using the Scrum method and other agile practices within individual software sub-teams, but expressed a strong desire for guidance on how to scale these for use in large multi-team context, "*if we can get...the other functions who work in those projects like firmware and hardware, if we can get them simply to follow the water-scrum-fall, that might be as good as what we can achieve (P2).*" However, there is relatively little guidance in the academic or practitioner literature on this, an issue also reported by Fitzgerald et al. (2013)and Cawley et al. (2010). However, there are studies which report the successful use of agile software development in safety critical systems (Fitzgerald et al., 2013; VanderLeest and Buter, 2009; Gary et al., 2011b; Cawley et al., 2010). A commonly reported point in the literature is that agile methods and practices have to be adapted according to the requirements of a project (Fitzgerald et al., 2013; VanderLeest and Buter, 2009; Gary et al., 2011b; Cawley et al., 2010).

The participants reported feeling confident about using Waterfall because they have plenty of practical examples from the past. The company finds it difficult to experiment with something new, given the safety-critical nature of their projects and with very little or no prior example to follow. Also, there is relatively less guidance available in the literature about the use of agile software development in safety-critical systems, particularly in the avionics industry (Ge et al., 2010; Paetsch et al., 2003; Wang and Wagner, 2016b; Carpenter and Dagnino, 2014; Heeager, 2014; Huang et al., 2012; Axelsson et al., 2016).

As a consequence, the company has a well documented and understood software development process, which is reflected in the organisational culture. The participants, therefore, identified the need to change the mindset of their colleagues (Challenge 13), as the Waterfall process has been in practice for years in the company. As one participant (P4) said "*it would be quite difficult to have an Agile process that spanned this whole organisation, without a fairly fundamental paradigm shift.*" Fitzgerald et al. (2013) also report this issue in their study. Fitzgerald et al. found that agile methods and practices are "*developer-centric*", therefore, they are typically easily accepted by the development team, whereas, management requires some convincing about the benefits of agile software development. One of the reasons behind the resistance by the management is the perception of "*short termism*" about agile software development (Fitzgerald et al., 2013). Management usually prefers an upfront complete plan, whereas, the agile philosophy advocates short term sprints and a "*plan as you go*" approach.

## 7. Conclusion

This study reports the results of a series of interviews and workshops in a large avionics company who are experimenting with the incorporation of agile software development into their software development process. The research yielded 13 challenges faced by the different software teams interviewed during the study concerning the application of agile software development for safety-critical systems. The challenges can be grouped into three categories: the influence of wider Waterfall like systems engineering processes on the practice of agile software development within a single team, the necessarily complex interactions with external stakeholders, including multiple customer roles, and the demand for documentation to meet required regulatory standards. We also found that cultural resistance within the company was a cross-cutting concern, limiting the use of elements of agile software development.

### *7.1. Limitations*

There are several limiting aspects to our study, which we discuss as potential avenues for future work. First, the sensitive nature of much of the work in the company necessarily limited our access to the details of projects. Our findings are primarily based on the perspectives given to us by our interview participants, and we were consequently unable to verify them through independent inspection of other sources of evidence, such as project software repositories and software process documentation. The interview participants were selected by the company, based on their availability and different perspectives and experiences of agile software development. Considerable effort was made by the researchers to establish the relationship with the company to allow the interviews to be conducted in the described form. We believe the arrangements reflect the constraints imposed on much of the research conducted in safety-critical contexts, given the often sensitive nature of such work. However, this does create threats to the validity of the work, which we have sought to mitigate by relating the findings to those available in the literature. We are also seeking to further generalise our findings by conducting interviews in other organisations engaged in similar work.

Second, we note that one of our findings during the interview stage of the research was not validated during the review workshop, concerning the conflict between agile software development to software quality assurance and that demanded by regulatory standards. This topic was included in the interview instrument because of the prevalence of the challenge in the literature. Specifically, Notander et al. (2013) reported that independent testing of complex systems, in accordance with the regulatory standard, DO-178C (RTCA) was very difficult due to the need for significant specialist knowledge about the test subject. It was anticipated that this challenge would also be identified by the participants, particularly given that agile software development advocates that testing should be conducted by the software team as part of the design and implementation process.

However, the issue was rejected during the validation workshop. According to the interview participants, they had great difficulty getting their system tested by an independent quality assurance team. The independent quality assurance team did not have the inherent knowledge of the system needed to develop effective tests. To mitigate this, the software team conducted trainings and workshops with the independent test teams but found these insufficient. So the software team performed the testing themselves while the independent quality assurance team *acted as witnesses to the test-*

*ing* and signed off the documentation at the end. This approach worked well for the software team and was perceived to satisfy the demands of the standard for independent testing whilst also enabling effective tests to be developed.

The rejection of this challenge was surprising to us because the standard DO-178C mandates an independent testing body. Later reviewing the interview material, we noted that during one of the interviews a lead software engineer agreed that the risk of bias in this approach was *"... a problem, it's an ongoing problem."* In reviewing this, it is possible that the participants do not view the approach to testing as problematic with respect to the standard, but are still concerned about the risk of bias, regardless. The issue highlights the risk in our research method of misinterpretation of findings. However, the validation step is applied to mitigate this.

### *7.2. Future Research Questions*

Despite the limitations described above, the research has identified several key themes in the challenges of applying agile software development to safety-critical systems engineering and provides a roadmap for addressing the challenges. Beyond these broad challenges, we have identified a set of immediate research questions to guide future efforts in this area, summarised in Figure 6. These questions are indicative of immediate research directions that can be undertaken in the short term within these broad themes.

Questions 1 and 2 address the theme of mitigating the pressure for Waterfall development processes for software engineering processes. Question 1 concerns the development of lightweight design review methods that accommodate more rapid changes in software design without compromising on design quality. We envisage leveraging existing agile methods and practices to facilitate this, such as continuous inspection techniques. Question 2 concerns the need for alternative approaches to the structuring of requirements specifications to better support decomposition of requirements in complex systems. In particular, we are investigating whether a feature driven approach to requirements engineering, embodying detailed specifications as user stories and scenarios provides for better decomposability. We also envisage enhancing traceability of requirements in complex systems through the adaptation of behaviour driven development techniques.

Question 3 and 4 address the theme of coordinating the stakeholder relationships (both internal and external) within complex systems engineering projects. In particular, software engineering has developed sophisticated

| Question | Challenge (Figure 5) | |
|---|---|---|
| 1 | 1 | Can lightweight gate reviews be used to achieve the same quality of the design? |
| 2 | 5 | How can requirements for complex systems be better structured and decomposed to enable agile development efforts? |
| 3 | 6 | How can continuous integration methods be extended to satisfy the heterogeneous nature of complex systems engineering projects in safety-critical environments? |
| 4 | 7 | How can agile customer management methods be adapted to the complex customer structure of safety critical systems? |
| 5 | 10 | To what extent can the maintenance of documentation be automated, or better integrated into the cost estimation process? |

Figure 6: Future work research questions

techniques for achieving continuous integration of software products. We envisage that these techniques can be extended further across the technology stack of firmware and hardware through networked deployments of software on hardware under development, or the development of realistic hardware simulators concurrently with hardware development efforts. Similarly, recent advances in software process development that enable abstraction of hardware, such as virtualisation, DevOps and Infrastructure as Code may be adapted to provide solutions to this integration challenge.

Separately, Question 4 concerns the adaptation of agile customer management techniques, through the product owner to complex systems projects. By convention, agile software development assume that all the interests of 'the customer' can be represented to the software team via the product owner, shielding the development team from the conflicts, tensions, and negotiations that may occur between different stakeholders. However, the size and complexity or large-scale systems engineering projects, together with the typically complex interplay between stakeholders (recall Figure 4) makes the allocation of this role to a single person impractical. Several authors have described proposals or experiences of scaling agile methods and practices particularly for scaling the role of the product owner. For example, Lowery and Evans (2007) reports on experiences of implementing a hierarchy of product owners in the BBC's iPlayer app. They found that a critical aspect of their approach was ensuring coordination between product owners and scrum masters in the different teams and placed significant emphasis on time in the product owners' schedules to accomplish this. The popular Scaled Agile Framework (Leffingwell, 2016) also advocates the use of a hierarchy within product ownership, between product managers who are responsible for the high level direction and product owners who are embedded in particular teams focused on specific aspects of functionality. There is a need to explore how these hierarchical approaches to managing the relationship with customers through the product owner can be adapted to both the heterogeneous nature of systems engineering projects which combine a variety of software and hardware elements; and the consortium arrangement of customers in systems engineering projects.

Question 5 concerns the automated generation of supplemental documentation, addressing the need to reduce friction in Software Engineering projects. Traceability remains a critical component of standards and regulations for safety critical environments. There will be an on-going need to produce evidence that system artifacts remain consistent with their requirements and design, such that any associated safety evaluations are reliable. To adapt agile software development to fit with this context, there is a need to develop mechanisms for automatically regenerating artifacts as changes occur, or better support their continuous maintenance alongside mainstream development efforts. A factor here will be to integrate documentation maintenance efforts into on-going software development task cost estimates, such that all necessary changes are continuously tracked. Similarly, there is a need to develop better methods for modelling and representing dependencies amongst software project artifacts, such that when changes occur the impact can be more efficiently assessed.

Crucially, this study has demonstrated that there is a need to adapt agile software development to fit within the constraints of software development for safety-critical systems and investigated the specific challenges in detail. In particular, there is a need to understand how agile software development can be scaled to fit large-scale, complex systems engineering efforts comprising multiple development efforts that include both software and hardware components on projects that may last many decades. Ultimately, these questions reflect the need to better align the *tempo* of safety-critical system developments and that assumed by agile software development. The agile philosophy is to accommodate the constant, rapid, concurrent change of software development projects, due to inevitable external pressures. The complexity created by this change is then mitigated through the disciplined application of a combination of tools and methods. Conversely, the philosophy in software development for safety-critical systems is to deliberately constrain options for (and pace of) change in order to maintain traceability of artifacts. Applying agile software development to safety-critical systems will, therefore, require the development of tools and methods that provide for the same standard of continuous traceability.

## References

## References

Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*, 2017.

Jason Ard, Kristine Davidsen, and Terril Hurst. Simulation-based embedded agile development. *IEEE Software*, 31(2):97–101, 2014.

PAPER AUTHORS. Interview questions material repository, 2018.

Jakob Axelsson, Efi Papatheocharous, Jaana Nyfjord, and Martin Törngren. Notes on agile and safety-critical development. *ACM SIGSOFT Software Engineering Notes*, 41(2):23–26, 2016.

Kent Beck and Cynthia Andres. *Extreme Programming Explained*. XP Series. Addison Wesley/Pearson Education, second edition, February 2005.

Kent Beck, Mike Beedle, Arie van Bennekum andw Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. The agile manifesto. Available at `http://agilemanifesto.org`, 2001.

Herbert D. Benington. Production of large computer programs. *Annals of the History of Computing*, 5(4):350–361, October 1983.

Sue Black, Paul P Boca, Jonathan P Bowen, Jason Gorman, and Mike Hinchey. Formal versus agile: Survival of the fittest. *Computer*, 42(9), 2009.

Barry Boehm. Get ready for agile methods, with care. *IEEE Computer*, 35 (1):64—69, January 2002.

Barry Boehm and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

Scott E. Carpenter and Aldo Dagnino. Is agile too fragile for space-based systems engineering? In *2014 IEEE International Conference on Space Mission Challenges for Information Technology*, pages 38–45, September 2014.

Oisín Cawley, Xiaofeng Wang, and Ita Richardson. Lean/agile software development methodologies in regulated environments - state of the art. In Pekka Abrahamsson and Nilay V. Oza, editors, *Lean Enterprise Software and Systems - First International Conference, LESS 2010, Helsinki, Finland, October 17-20, 2010. Proceedings*, volume 65 of *Lecture Notes in Business Information Processing*, pages 31–36. Springer, 2010.

Roderick Chapman. Industrial experience with agile in high-integrity software development. In *Safety Critical Systems Club*, 2016.

Roderick Chapman, Neil White, and Jim Woodcock. What can agile methods bring to high-integrity software development? *Commun. ACM*, 60(10): 38–41, September 2017. ISSN 0001-0782. doi: 10.1145/3133233. URL http://doi.acm.org/10.1145/3133233.

Emmanuel Chenu. Agility and lean for avionics. Paper Presented at Lean, Agile Approach to High Integrity Software, Paris, 2009. http://manu40k.free.fr/AgilityAndLeanForAvionics1.pdf, 2009.

Emmanuel Chenu. Agile & lean software development for avionic software. In *6th European Congress on Real Time Software and Systems*, pages 1–3, Toulouse, France, February 2012. Association Aéronautique Astronautique de France.

Alistair Cockburn. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional, October 2004.

David J Coe and Jeffrey H Kulick. A model-based agile process for do-178c certification. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, page 1, 2013.

David Cohen, Mikael Lindvall, and Patricia Costa. An introduction to agile methods. *Advances in computers*, 62(03):1–66, 2004.

CollabNet VersionOne. 13th annual state of agile report. https://www.stateofagile.com, May 2019.

Kieran Conboy. Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research*, 20(3):329–354, 2009.

Torgeir Dingsøyr and Casper Lassenius. Emerging themes in agile software development: Introduction to the special section on continuous value delivery. *Information and Software Technology*, 77:56–60, 2016.

Bruce Powel Douglass. *Agile Systems Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2016.

Christof Ebert, Marco Kuhrmann, and Rafael Prikladnicki. Global software engineering: Evolution and trends. In *11th IEEE International Conference on Global Software Engineering, ICGSE 2016, Orange County, CA, USA, August 2-5, 2016*, pages 144–153. IEEE Computer Society, 2016.

Brian Fitzgerald, Gerard Hartnett, and Kieran Conboy. Customising agile methods to software practices at intel shannon. *EJIS*, 15(2):200–213, 2006.

Brian Fitzgerald, Klaas-Jan Stol, Ryan O'Sullivan, and Donal O'Brien. Scaling agile methods to regulated environments: an industry case study. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 863–872. IEEE Computer Society, 2013.

Kevin Gary, Andinet Enquobahrie, Luis Ibáñez, Patrick Cheng, Ziv Yaniv, Kevin Cleary, Shylaja Kokoori, Benjamin Muffih, and John Heidenreich. Agile methods for open source safety-critical software. *Software Practice & Experience*, 41(9):945–962, 2011a.

Kevin Gary, Andinet Enquobahrie, Luis Ibanez, Patrick Cheng, Ziv Yaniv, Kevin Cleary, Shylaja Kokoori, Benjamin Muffih, and John Heidenreich. Agile methods for open source safety-critical software. *Software - Practice and Experience*, 41(9):945–962, 8 2011b.

Xiaocheng Ge, Richard F. Paige, and John A. McDermid. An iterative approach for development of safety-critical software and safety arguments. In Sallyann Freudenberg and Joseph Chao, editors, *2010 Agile Conference, AGILE 2010, Orlando, Florida, USA, August 9-13, 2010*, pages 35–43. IEEE Computer Society, 2010.

Martin Glas and Sven Ziemer. Challenges for agile development of large systems in the aviation industry. In Shail Arora and Gary T. Leavens, editors, *Companion to the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA*, pages 901–908. ACM, 2009.

Thomas Gruber, Egbert Althammer, and Erwin Schoitsch. Field test methods for a co-operative integrated traffic management system. In Erwin

Schoitsch, editor, *Computer Safety, Reliability, and Security*, pages 183–195, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

A. Hajou, Ronald S. Batenburg, and Slinger Jansen. An insight into the difficulties of software development projects in the pharmaceutical industry. *Lecture Notes in Software Engineering*, 3(4):267– 275, November 2015.

Ali Hajou, Ronald Batenburg, and Slinger Jansen. How the pharmaceutical industry and agile software development methods conflict: A systematic literature review. In Bernady O. Apduhan, Ana Maria A. C. Rocha, Sanjay Misra, David Taniar, Osvaldo Gervasi, and Beniamino Murgante, editors, *2014 14th International Conference on Computational Science and Its Applications, Guimaraes, Portugal, June 30 - July 3, 2014*, pages 40–48. IEEE Computer Society, 2014.

Geir Kjetil Hanssen, Børge Haugset, Tor Stålhane, Thor Myklebust, and Ingar Kulbrandstad. Quality assurance in scrum applied to safety critical software. In Helen Sharp and Tracy Hall, editors, *Agile Processes, in Software Engineering, and Extreme Programming - 17th International Conference, XP 2016, Edinburgh, UK, May 24-27, 2016, Proceedings*, volume 251 of *Lecture Notes in Business Information Processing*, pages 92–103. Springer, 2016.

Lise Tordrup Heeager. How can agile and documentation-driven methods be meshed in practice? In Giovanni Cantone and Michele Marchesi, editors, *Agile Processes in Software Engineering and Extreme Programming - 15th International Conference, XP 2014, Rome, Italy, May 26-30, 2014. Proceedings*, volume 179 of *Lecture Notes in Business Information Processing*, pages 62–77. Springer, 2014.

Lise Tordrup Heeager and Peter Axel Nielsen. A conceptual model of agile software development in a safety-critical context: A systematic literature review. *Information and Software Technology*, 2018.

James D. Herbsleb and Audris Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Trans. Software Eng.*, 29(6):481–494, 2003.

Phillip. M. Huang, Ann. G. Darrin, and Andrew. A. Knuth. Agile hardware and software system engineering for innovation. In *2012 IEEE Aerospace Conference*, pages 1–10, March 2012.

Hanne-Gro Jamissen. The challenges to the safety process when using agile development models. Master's thesis, Østfeld University College, 2012.

Henrik Jonsson, Stig Larsson, and Sasikumar Punnekkat. Agile practices in regulated railway software development. In *23rd IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Dallas, TX, USA, November 27-30, 2012*, pages 355–360. IEEE, 2012.

Matti Kaisti, Ville Rantala, Tapio Mujunen, Sami Hyrynsalmi, Kaisa Könnölä, Tuomas Mäkilä, and Teijo Lehtonen. Agile methods for embedded systems development - a literature review and a mapping study. *EURASIP J. Emb. Sys.*, 2013:15, 2013.

John C Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering*, pages 547–550. ACM, 2002.

Andy Koronios, Michael Lane, and Glen Van Der Vyver. Facilitators and inhibitors for the adoption of agile methods. In *Systems Analysis and Design: People, Processes, and Projects*, pages 43–62. Routledge, 2015.

Dean Leffingwell. *SAFe 4.0 Reference Guide: Scaled Agile Framework for Lean Software and Systems Engineering: Scaled Agile Framework for Lean Software and Systems Engineering*. Addison-Wesley Professional, first edition, 2016.

Howard Lei, Farnaz Ganjeizadeh, Pradeep Kumar Jayachandran, and Pinar Ozcan. A statistical analysis of the effects of scrum and kanban on software development projects. *Robotics and Computer-Integrated Manufacturing*, 43:59–67, 2017.

Mikael Lindvall, Victor R. Basili, Barry W. Boehm, Patricia Costa, Kathleen Coleman Dangle, Forrest Shull, Roseanne Tesoriero Tvedt, Laurie A. Williams, and Marvin V. Zelkowitz. Empirical findings in agile methods. In Don Wells and Laurie A. Williams, editors, *Extreme Programming and Agile Methods - XP/Agile Universe 2002, Second XP Universe and First Agile Universe Conference Chicago, IL, USA, August 4-7, 2002, Proceedings*, volume 2418 of *Lecture Notes in Computer Science*, pages 197–207. Springer, 2002.

Mike Lowery and Marcus Evans. Scaling product ownership. In Jutta Eckstein, Frank Maurer, Rachel Davies, Grigori Melnik, and Gary Pollice, editors, *AGILE 2007 Conference (AGILE 2007), 13-17 August 2007, Washington, DC, USA*, pages 328–333. IEEE Computer Society, 2007.

Luiz Eduardo Galvão Martins and Tony Gorschek. Requirements engineering for safety-critical systems: A systematic literature review. *Information & Software Technology*, 75:71–89, 2016.

Martin McHugh, Oisín Cawley, Fergal McCaffery, Ita Richardson, and Xiaofeng Wang. An agile v-model for medical device software development to overcome the challenges with plan-driven software development lifecycles. In John Knight and Craig E. Kuziemsky, editors, *Proceedings of the 5th International Workshop on Software Engineering in Health Care, SEHC 2013, San Francisco, California, USA, May 20-21, 2013*, pages 12–19. IEEE Computer Society, 2013.

Subhas Chandra Misra, Vinod Kumar, and Uma Kumar. Identifying some important success factors in adopting agile software development practices. *Journal of Systems and Software*, 82(11):1869–1890, 2009.

Thor Myklebust, Tor Stålhane, and Narve Lyngby. An agile development process for petrochemical safety conformant software. In *2016 Annual Reliability and Maintainability Symposium (RAMS)*, January 2016.

Dan North. Introducing behaviour driven development. *Better Software Magazine*, 2006.

Jesper Pedersen Notander, Martin Höst, and Per Runeson. Challenges in flexible safety-critical software development - an industrial qualitative survey. In Jens Heidrich, Markku Oivo, Andreas Jedlitschka, and Maria Teresa Baldassarre, editors, *Product-Focused Software Process Improvement - 14th International Conference, PROFES 2013, Paphos, Cyprus, June 12-14, 2013. Proceedings*, volume 7983 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2013.

Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises, 9-11 June 2003, Linz, Austria*, pages 308–313. IEEE Computer Society, 2003.

Richard F. Paige, Andy Galloway, Ramon Charalambous, Xiaocheng Ge, and Phillip J. Brooke. High-integrity agile processes for the development of safety critical software. *IJCCBS*, 2(2):181–216, 2011.

Stephen R. Palmer. *A Practical Guide to Feature-Driven Development*. Prentice Hall, February 2002.

Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional, May 2003.

Balasubramaniam Ramesh, Lan Cao, and Richard Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010.

Rational. The rational unified process. best practices for software development teams. white paper TP026B, The Rational Corporation, 2003.

Derek Rayside, Aleksandar Milicevic, Kuat Yessenov, Greg Dennis, and Daniel Jackson. Agile specifications. In Shail Arora and Gary T. Leavens, editors, *Companion to the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA*, pages 999–1006. ACM, 2009.

RTCA. *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*. RTCA, 1150 18th St, NW, Suite 910. Washington DC 20036-3816 USA., December 2011.

Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

Nancy Van Schooenderwoert and Ron Morsicato. Taming the embedded tiger - agile test techniques for embedded software. In *2004 Agile Development Conference (ADC 2004), 22-26 June 2004, Salt Lake City, UT, USA*, pages 120–126. IEEE Computer Society, 2004.

Ken Schwaber and Mike Beedle. *Agile Software Development with SCRUM*. Prentice Hall, 2001.

Lubna Siddique and Bassam A. Hussein. Practical insight about choice of methodology in large complex software projects in norway. In *2014 IEEE International Technology Management Conference*, June 2014.

Tor Stålhane and Thor Myklebust. The role of CM in agile development of safety-critical software. In Floor Koornneef and Coen van Gulijk, editors, *Computer Safety, Reliability, and Security - SAFECOMP 2015 Workshops, ASSURE, DECSoS, ISSE, ReSA4CI, and SASSUR, Delft, The Netherlands, September 22, 2015, Proceedings*, volume 9338 of *Lecture Notes in Computer Science*, pages 386–396. Springer, 2015.

Tor Stålhane, Thor Myklebust, and Geir Hanssen. The application of safe scrum to iec 61508 certifiable software. In *11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference 2012, 25-29 June 2012, Helsinki, Finland*. Curran Associates Inc., 2012.

Tor Stålhane, Geir Kjetil Hanssen, Thor Myklebust, and Børge Haugset. Agile change impact analysis of safety critical software. In Andrea Bondavalli, Andrea Ceccarelli, and Frank Ortmeier, editors, *Computer Safety, Reliability, and Security - SAFECOMP 2014 Workshops: ASCoMS, DECSoS, DEVVARTS, ISSE, ReSA4CI, SASSUR. Florence, Italy, September 8-9, 2014. Proceedings*, volume 8696 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2014.

Ernst Stelzmann. Contextualizing agile systems engineering. In *2011 IEEE International Systems Conference*, pages 163–167, April 2011.

M. Tanveer. Agile for large scale projects ? a hybrid approach. In *2015 National Software Engineering Conference (NSEC)*, pages 14–18, Dec 2015. doi: 10.1109/NSEC.2015.7396338.

Dan Turk, Robert B. France, and Bernhard Rumpe. Limitations of agile software processes. *CoRR*, abs/1409.6600, 2014.

Daniel E. Turk, Robert B. France, and Bernhard Rumpe. Assumptions underlying agile software-development processes. *J. Database Manag.*, 16(4): 62–87, 2005.

Steven H. VanderLeest and A. Buter. Escape the waterfall: Agile for aerospace. In *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, pages 6.D.3–1–6.D.3–16, October 2009.

L. R. Vijayasarathy and C. W. Butler. Choice of software development methodologies: Do organizational, project, and team characteristics matter? *IEEE Software*, 33(5):86–94, Sep. 2016.

Matti Vuori. Agile development of safety-critical software. *Tampere University of Technology. Department of Software Systems; 14*, 2011.

Xiaofeng Wang, Kieran Conboy, and Minna Pikkarainen. Assimilation of agile practices in use. *Information Systems Journal*, 22(6):435–455, 2012.

Yang Wang and Stefan Wagner. Towards applying a safety analysis and verification method based on STPA to agile software development. In *Proceedings of the International Workshop on Continuous Software Evolution and Delivery, CSED@ICSE 2016, Austin, Texas, USA, May 14-22, 2016*, pages 5–11. ACM, 2016a.

Yang Wang and Stefan Wagner. Towards applying a safety analysis and verification method based on STPA to agile software development. In *Proceedings of the International Workshop on Continuous Software Evolution and Delivery, CSED@ICSE 2016, Austin, Texas, USA, May 14-22, 2016*, pages 5–11. ACM, 2016b.

Tom Wengraf. *Qualitative Research Interviewing: Biographic Narrative and Semi-Structured Methods*. SAGE Publications, 2001. URL `https://books.google.co.uk/books?id=gj5rvAR1CYgC`.

Andrew Wils, Stefan Van Baelen, Tom Holvoet, and Karel De Vlaminck. Agility in the avionics software world. In Pekka Abrahamsson, Michele Marchesi, and Giancarlo Succi, editors, *Extreme Programming and Agile Processes in Software Engineering, 7th International Conference, XP 2006, Oulu, Finland, June 17-22, 2006, Proceedings*, volume 4044 of *Lecture Notes in Computer Science*, pages 123–132. Springer, 2006.

Jason D Winningham, David J Coe, and Jeffrey H Kulick. Agile systems integration process. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, page 149. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.

W. E. Wong, A. Demel, V. Debroy, and M. F. Siok. Safe software: Does it cost more to develop? In *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement*, pages 198–207, June 2011.