Information and Software Technology 103 (2018) 22–39

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A conceptual model of agile software development in a safety-critical context: A systematic literature review

Lise Tordrup Heeager[a], Peter Axel Nielsen[b,*]

[a] Department of Management, Aarhus University, Denmark
[b] Department of Computer Science, Aalborg University, Denmark

ARTICLE INFO

Keywords:
Agile software development
Agile processes
Software development
Safety-critical software systems
Systematic literature review
Interpretive literature review

ABSTRACT

Context: Safety-critical software systems are increasingly being used in new application areas, such as personal medical devices, traffic control, and detection of pathogens. A current research debate is regarding whether safety-critical systems are better developed with traditional waterfall processes or agile processes that are purportedly faster and promise to lead to better products.
Objective: To identify the issues and disputes in agile development of safety-critical software and the key qualities as found in the extant research literature.
Method: We conducted a systematic literature review as an interpretive study following a research design to search, assess, extract, group, and understand the results of the found studies.
Results: There are key issues and propositions that we elicit from the literature and combine into a conceptual model for understanding the foundational challenges of agile software development of safety-critical systems. The conceptual model consists of four problematic practice areas and five relationships, which we find to be even more important than the problematic areas. From this review, we suggest that there are important research gaps that need to be investigated.
Conclusions: We suggest that future research should have a primary focus on the relationships in the resulting conceptual model and specifically on the dynamics of the field as a whole, on incremental versus iterative development, and on how to create value with minimal but sufficient effort.

1. Introduction

We use safety-critical information technology (IT) systems in abundance and safety-critical embedded software systems are rising in number and complexity. For example, when diabetes patients use a digital insulin pump to control and fine-tune the level of blood glucose, it is a safety-critical system [41]. The need for diabetes treatment with insulin is rising dramatically, and many more patients will be treated with digitised insulin injections and use digitised blood glucose measurements. The benefits of these new devices are high, but the safety risks cannot be taken lightly. This is just one type of safety-critical IT system, and there are many other key applications that are highly critical for safety. These are, for example, found in the aerospace and medical sectors and in the transportation, energy, and process industries. Examples are flight control, radiation therapy, self-driving cars, airbags, railway control, and development of fuels (S1; S39; S refers to the reviewed literature at the end of the paper).

The severity of potential safety fails has led to the institutionalisation of approval procedures and certification. In the USA, the US Food and Drug Administration (FDA) has to approve medical devices, such as the digital insulin pump [84]. Other countries have similar agencies to approve and certify safety-critical products. There are several ISO standards [42], process models, and process maturity models (e.g., [17,43]) that are part of stipulating how safety-critical systems must be developed. For example, EN 50,128 is a European standard for the development of railway applications (S22). Due to the concern regarding risks and the well-being and lives of the users, the products and their development processes are subject to legislation, public interest, and the concerns of patients, consumers, and citizens, which become the responsibilities of the producers. The development of these safety-critical products is highly regulated; thus, the system developers are mandated by law to comply with an appropriate standard (S43).

To achieve approval, the software must be verified and validated. While verification ensures that the software is built in the right way so that it is safe, validation ensures that the right system is built (S7). The

* Corresponding author.
E-mail address: pan@cs.aau.dk (P.A. Nielsen).

https://doi.org/10.1016/j.infsof.2018.06.004
Received 17 July 2017; Received in revised form 21 May 2018; Accepted 13 June 2018
Available online 03 July 2018
0950-5849/

FDA, for example, requires several practices and documentation for verification and validation of the software (S32). They have high demands for the development process; thus, most of the FDA requirements are directly related to the process activities (e.g., requirement analysis, design, implementation, etc.). In addition, the FDA expects a sufficient level of auditability within the software process itself, meaning that parts of the development lifecycle must be tracked for external auditors to assess whether the system can be approved (S31).

The traditional answer to the challenges of developing safety-critical software has been to establish a formal development process moulded over experience from project management and from quality management [12,78,94]. The general process model is the waterfall model where the underlying idea is rational (i.e., first we think and then we do). For software development, this means that, first, we develop the requirements, and when these are fixed, we move on to design the software. When the software design is completed, we program the software, after which the quality of the software is assessed through elaborate and systematic testing. The primary reason for this process model is that it allows for thinking carefully about the system features and how they will be used in advance. In addition, we can reason about the properties of the system and the risks caused by the system and how these must be mitigated.

A fundamental problem exists with traditional waterfall development processes. The waterfall processes work well in circumstances in which requirements (and risks) are stable and well understood in advance and where little learning of additional or changed requirements is expected during the development process. Alternatively, if requirements are expected to change because users, marketing, managers, and developers are learning during the stages of development, then the waterfall processes are inappropriate [11,18,23]. Agile development processes [5,18,21] have been designed to alleviate this problem. The key features of agile development processes are that they support the management of change and embrace change and that change in requirements should be taken as a positive development. It further means that there should be clear methods of reacting to and learning from changes.

Research on agile software development of safety-critical software products has been published since 2001, and there are numerous research publications on the topic. However, there is no clear accumulation of knowledge on the topic; therefore, we suggest that a literature review is needed. The research question defining the focus of the literature review is: *How can the use of agile software development be increased in a safety-critical context?*

The purpose of the literature review is to improve the knowledge of how agile software development can be improved, increased and advanced when developing safety-critical software products and uncovering areas of interest and further research. This follows calls for more literature reviews in general [47,87] to close a gap in our understanding of the field. The purpose of this literature review is developmental [83], as we aim to build a new conceptualisation towards a coherent theory of what characterises agile development of safety-critical software.

The remaining paper is organised as follows. Section 2 presents the research method of the literature review. Section 3 describes the analysis results, which are discussed in Section 4. Finally, conclusions on the literature review are drawn in Section 5.

2. Research method

The research on the topic of agile development of safety-critical software is not entirely in its infancy, as the first research was published in 2001. While the research literature is vast, there is little overview of the body of knowledge. There are very sparse literature reviews related to our research area. Cawley et al. [16] concerns lean and agile software development in regulated environments, whereas [58] and [36] focus on agile software development of the medical devices and in the pharmaceutical industry. While Hajou et al. [36] conclude that the two are incompatible, Cawley et al. [16] conclude that adoption of agile development in these areas is difficult and McHugh et al. [58] argue that tailoring of the agile methods are needed and propose a mixed method. These conclusions do not reveal the challenges and possibilities of using agile software development of safety-critical products. Thus, a review of the literature is needed to understand the state of knowledge and the future direction of this stream of research.

With an empirically motivated research question, the primary goal is to investigate agile software development of safety-critical software products by gathering and synthesising analyses, evidence and results in the literature [70] on this matter. To this end, we take initial inspiration in what in software engineering research has become known as a systematic mapping with the purpose of structuring the research area [70]. To this we add the understanding of how to conduct interpretive literature reviews (e.g., [74,83,87]). It is not uncommon that there is overlap between the two types of reviews and that the methods are used in combination [48].

Templier and Paré [83] suggested that there are four types of literature reviews: narrative, developmental, cumulative, and aggregative. Narrative reviews assemble and summarise extant literature on a specific topic, providing a comprehensive understanding of the current state of knowledge in the area. The aim of developmental reviews is to provide new conceptualisations, based on previous research. The developmental review involves a systematic search of the literature to be reviewed, while the narrative review addresses an illustrative sample. Cumulative reviews synthesise extant literature (as narrative reviews) but further aim to compile empirical evidence to map bodies of literature and to draw overall conclusions regarding particular topics. Aggregative reviews test research hypotheses or propositions. By collating and pooling prior empirical data, they provide validations of pre-specified theoretical models and propositions. The immaturity of the existing literature on agile development of safety-critical software leads us to suggest that our literature review can be a narrative review or a development review, while it is much too early for a cumulative or aggregative review. The review is systematic, as we establish a detailed search procedure [87] without assuming that the research is accumulative [47]. In our review process, we follow the four phases suggested in [3]:

Phase 1: Systematically identifying and extracting a sample of papers,
Phase 2: Organising and preparing the analysis,
Phase 3: Coding and analysing the content,
Phase 4: Writing and reporting the findings.

These four phases are consistent with other recent review procedures (e.g., [74,83,87]).

2.1. Phase 1: systematically identifying and extracting a sample of papers

To guide the literature search, the scope of the review is defined as the possibilities and challenges of agile software development of safety-critical software products. Our intention is to cover the relevant literature and use the review to provide a conceptual model in answer to the research question. With this literature review we seek to provide an overview and to relate and possibly unite existing results.

Through preliminary searches we discovered that the research on agile software development in a safety-critical context has primarily been published in conferences (this was later confirmed, see Fig. 3). It was thus necessary to search in several digital libraries and search engines. These included the ACM Digital Library, IEEE Xplore Digital Library, AIS Electronic Library, Google Scholar, Scopus, and Web of Science. The search has been extensive, as we initially used search terms interactively and always starting with the following expression: (agile OR agility OR Scrum OR XP) AND (safety software OR safety-

critical software OR regulated software). The concept of agile development is not new. Since the agile manifesto was developed in 2001 [5], the agile methods and practices that we discuss today started to spread; thus, we limited the search to papers after 2001.

When the search expression was applied to Google Scholar, the result lists more than 100 K results, however increasingly irrelevant as we move down the search results. The search resulted in many potentially relevant papers (500 +). For our review we included in the initial list all references that were possibly relevant and we were explicitly inclusive. When searching and making initial sense of the literature, it was important to search widely. The initial search was restricted to titles and abstracts, but subsequently, a full-text search was included. After using search terms, we followed the backward snowball method to the references in the identified papers. With a forward snowball method, we followed the digital library lists of papers quoting the identified papers [88].

The most important criteria were relevance to the topic. The papers must focus on both the development of software for safety-critical systems and agile processes (irrespective of the terms used in the papers). The papers should address all three elements: agile development, safety-critical development, and software. We also included papers in which the case study was safety criticality also when the paper was not based on the safety literature. We included not only research but also theoretical analyses as these can reveal compatibilities between agile methods and standards for safety-critical systems and development processes. Practitioner experience reports were also included as these can reflect practical issues directly.

From the initial set of papers (500 +) many were excluded immediately based on the title, see Fig. 1. Many papers that showed up in the search could easily be excluded due to the lack of relevance to the topic or papers using other than the English language. Several of the papers only addressed either the safety-critical domain or agile software development, not both, or did not address software development. From this process, we identified 86 papers within the defined scope. From the 86 papers, we eliminated 21 papers after carefully reading the abstracts, as the foci of the papers were outside the scope of our literature review. Another 14 were eliminated after reading the full papers. This gave a final list of 51 papers, which is just above the recommended limit of 50 papers [3]. The final list confirms the initial assumption that most of the papers were published in conference proceedings and supports the reasons for the wide initial search process. The conferences are primarily under the three professional organisations for IT-related research: Association for Computing Machinery (ACM), Institute of Electrical and Electronics Engineers (IEEE), and Association of Information Systems (AIS). All the included papers had been subjected to peer review.

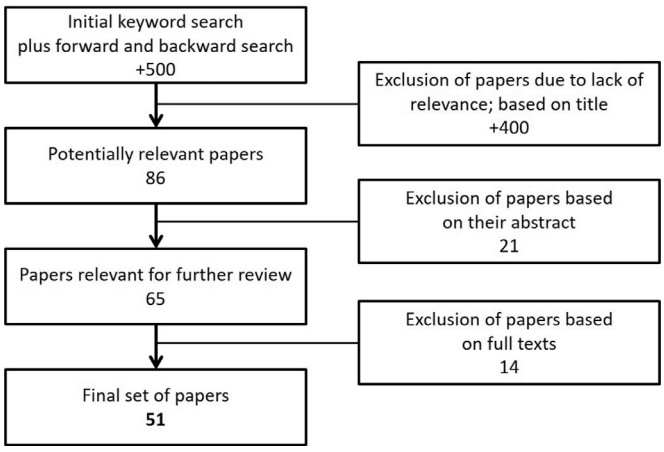

Fig. 1. Selection process of the papers.

Table 1
Initial coding scheme in the preparatory analysis.

Descriptive analysis	Content analysis
Research method	Definitions and characteristics of agile development
Publication type	Definitions and characteristics of safety-critical software
Agile method	Challenges
Specific domain	Possibilities
Embedded software	
Outcome	

2.2. Phase 2: organising and preparing the analysis

Bandara et al. [3] provided guidance on the use of tools to support the review process. We follow their advice and treat the literature review as a qualitative study in which the dataset consists of the identified literature. The tool we used to support the review is NVivo, which is a well-established qualitative data analysis tool with many embedded features to support qualitative data analysis. It has been used extensively in Phases 2 and 3 to support systematic capturing, coding, and analysing the literature. The 51 selected papers in the final set were treated as a qualitative dataset and loaded into NVivo. To prepare the detailed analysis, an initial coding was performed based on the research interest and research question. The list of the 51 included studies (S1-S51) can be found at the end of the paper.

The initial coding scheme in Table 1 was used as a preparation for the more detailed and systematic analysis to follow in the next phase with the purpose of initial categorisation of the papers. For all reviewed papers, it was important to understand the field through an initial mapping of which research methods are applied which type of publications are dominating, which agile methods are used, and which specific safety domains are studied. In addition, it is critical to understand how many papers concern embedded software and, finally, the outcome of each study. These aspects of the papers are summarised in a table in Appendix A and are described in Section 3.1. To understand how the included studies define agile development and safety criticality, these characteristics were defined. In Section 3.2, we discuss the definitions of agile and safety-critical development based on the included studies and secondary studies. The codes concerning challenges and possibilities of increasing agile development of safety-critical software are key to the research question and the analysis. This is described in Section 3.3 (problem areas) and Section 3.4 (relationships between the problem areas).

2.3. Phase 3: coding and analysing the content

The detailed analysis was inductive, as the reading and coding focused on extracting the literature to date. From this coding, themes were derived as the analysis evolved. The approach to coding followed the principles of grounded theory for literature studies [91]. The coding process consists of three steps: 1) open coding, 2) axial coding, and 3) selective coding (exemplified in Fig. 2). The open coding was performed to identify and build a set of concepts and insight based on the paper excerpts. Examples of open codes are ‘incremental documentation’ and ‘quality of requirements’. The second step in the analysis is the axial coding that was performed to identify categories and sub-categories. The analysis showed that most of the codes focused on four categories: requirements, lifecycle, documentation, or testing. In the third step, selective coding were used for integration and refinement of the categories that were identified. In this step, the problem areas and the relations were extracted. If a quote only had one code, it was assigned to that problem area (see Quote 1 in Fig. 2). If a quote had been assigned more than one code, an evaluation was made. If the quote contained two separate problem areas, it was assigned to both problem areas (see Quote 4 in Fig. 2). If the quote dealt with the relation of the two problem areas, it was assigned to a relation between the problem areas (see

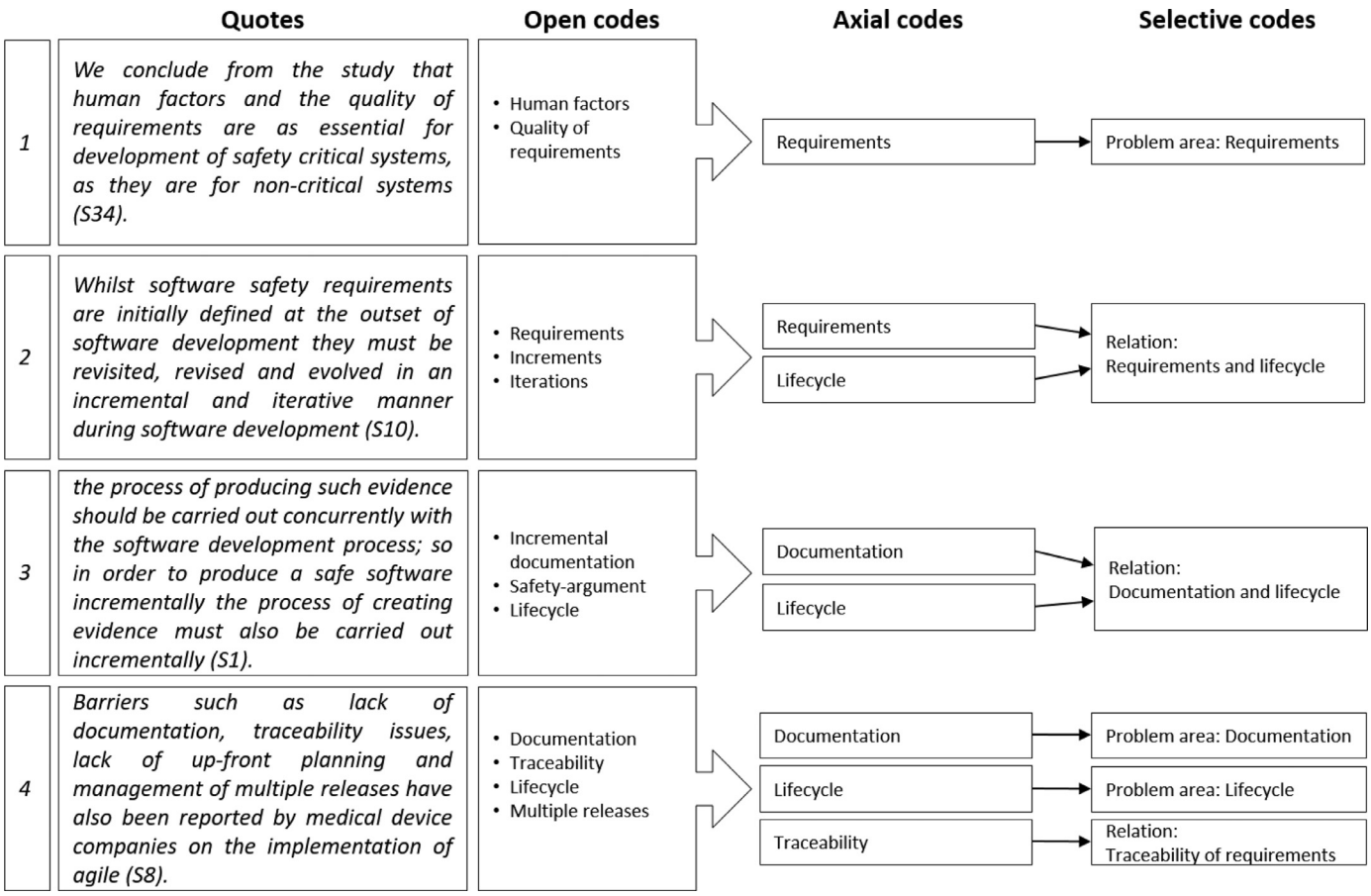

Fig. 2. The coding process exemplified (adapted from [15]).

Quotes 2 and 3 in Fig. 2).

3. Agile development of safety-critical software

The findings from this analysis of the literature initially cover the publication frequencies, then the key concepts of agile development and safety criticality are elicited, and the major part of the findings are found in the challenges in Sections 3.3 and 3.4. The categorisation of the research results is based on an emerging conceptual model clustering the challenges into four problem areas and five relationships between these areas.

3.1. Categorisation of the publications

In this section, the 51 papers are categorised and described based on the following characteristics: distribution over time, the applied research method, the type of publication, the applied agile method, the specific safety domain, whether the software is embedded, and the outcome of the paper. An overview of the papers and the categorisation can be found in Appendix A. The papers are distributed in time as depicted in Fig. 3. This provides a first overview of a research area, which is still active, but it is also likely that an intermediary peak was reached during 2012–2013.

Analysing the focus of the papers shows how the papers from 2001 to 2011 had two primary purposes: (i) providing a practical example of a case company using parts of the agile processes for developing safety-critical software, or (ii) conducting a theoretical analysis of compatibility between agile processes and a variety of safety-critical regulatory standards. While the main purposes of the first type of paper was to show the possibilities and motivate further research within the topic, the latter focused more on the challenges and incompatibilities. From 2012, we see a shift in the focus of the papers, which focus more on case studies used to evaluate both the advantages and challenges of agile development of safety-critical software. The analysis also shows an increase in the number of papers that suggest an adapted agile method for safety-critical development. In total, 12 of the 51 papers deal with this issue and eight of these are published after 2013.

We base the categorisation of the research method on the types of methods provided in the paper by Kjeldskov and Graham [49] who based their categorisation of research methods on the novel paper by Wynekoop and Conger [92]. This list of research methods has recently been used for categorisation by de Sousa Santos et al. [24]. We found it necessary to add three types of studies: experience reports, theoretical studies, and design studies. Fig. 4 shows a bar chart of the research methods. Most of the papers are based on empirical data from a natural setting. The research method used most is the case study (10 papers), describing a specific practice using agile software development in a safety-critical context. The case study method is not applied to the field until 2008, peaking in 2012 and 2013. The review also included experience reports (5 papers), which give examples of a specific practice from an insider practitioner perspective, and field experiments (3 papers), which are to modify a natural setting and evaluate the changes (e.g., experiments with introduction of agile practices). Five papers are based on surveys gathering a larger number of practitioner opinions and experiences of the topic. The review also includes a larger number of papers with the purpose of designing and evaluating an agile method for a safety-critical environment. These were categorised as design studies or laboratory experiments. While the design studies (9 of 11 papers) evaluated the proposed method or model in a natural setting, the laboratory experiments (2 papers) evaluated the proposed method or model in a simulated environment (student projects). The number of design studies peaked in 2013 (4 papers). The remaining papers were

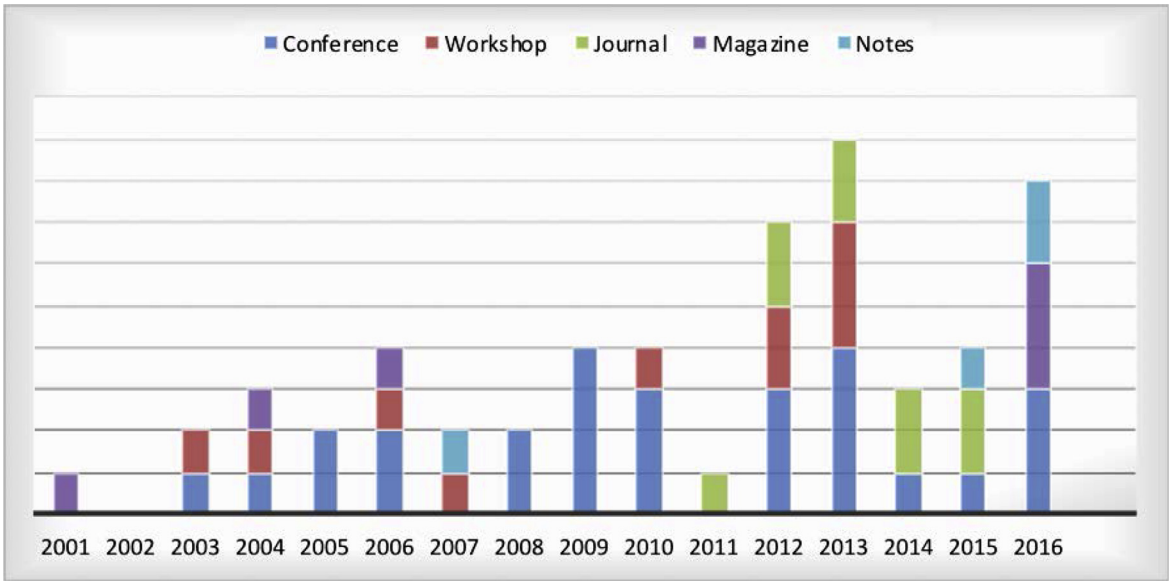

Fig. 3. Publication type divided pr. year.

theoretical (8 papers) offering a comparison of two theories/models (e.g., a comparison between requirements of a safety standard and practices of an agile method), and normative writing (9 papers) offering suggestions of challenges based on experiences without referring to a specific case. The papers offering a theoretical analysis were all published before 2013. The normative writings were published while the field was young (2003–2006) and again in 2015–2016. This indicates that the introduction of agile software development in terms of the agile manifesto was grounds for many considerations and discussions regarding whether this methodology could be applied in a safety-critical context. After some years, the changes in agile software development on the safety-critical environment were discussed.

The categorisation of publication type based on outlets shows a slight increase in the number of journal papers published on the topic since 2012, showing that the research area has grown more mature. The papers are mainly published at the Agile Conference (AGILE), the International Conference on Software Process Improvement and Capability Determination (SPICE), and in the *Journal of Information Technology Case and Application Research* (*JITCAR*), *Journal of Software Engineering and Applications* (*JSEA*), and *Journal of Software: Evolution and Process* (*JSEP*).

The majority (30 papers) do not focus on a specific agile method but refer to agile development in general. Ten papers focus on XP, eight on Scrum, and three use a combination of XP and Scrum. Understanding and categorising the papers according to the specific domain addressed shows that 23 of the papers focus on the medical domain, six on avionics, three on robotics, and five on other miscellaneous domains, whereas 14 of the papers do not specify a specific domain but deal with safety-critical development in general. See the distribution of agile methods and specific domains in Fig. 5.

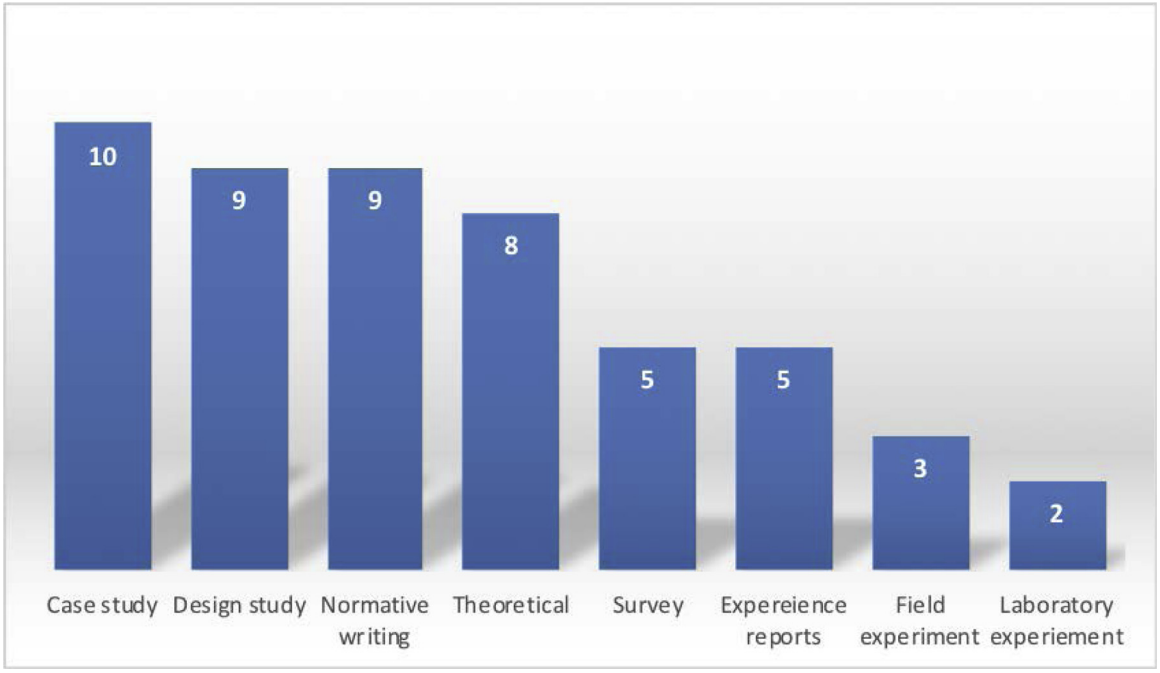

Fig. 4. Research methods.

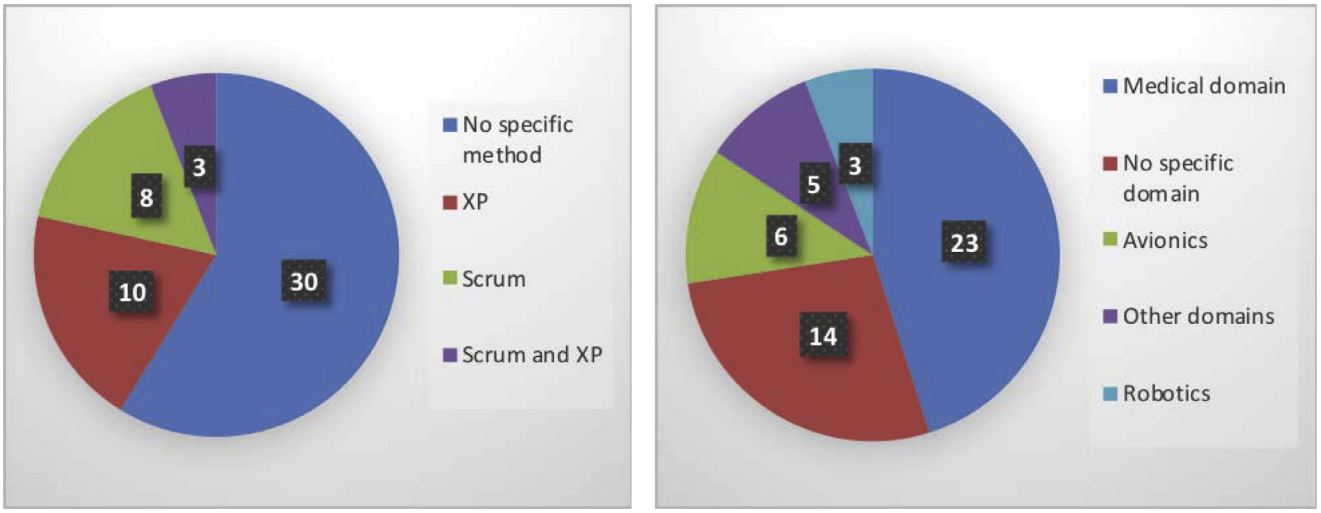

Fig. 5. The agile methods in the papers (left diagram) and the specific domain in the papers (right diagram).

The initial analysis also showed that 10 of the 51 papers are concerned with embedded software development. Most of these papers only report on a case of embedded software in a safety-critical device and do not discuss the implications this brings (S7; S11; S17; S20; S21; S39; S47). The three remaining papers deal explicitly with the issue of embedded safety-critical software development (S6; S8; S49).

3.2. Key concepts of agile software development and safety criticality

We define 'agile software development' as: *"the continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity) through its collective components and relationships with its environment"* ([21], p. 340). The reviewed literature is genuinely in accordance with this understanding of agile software development. The understanding of agile development in the reviewed research is predominantly explained by referring to the agile manifesto [5], which is contained in the theoretical foundation by [21]. Others refer to agile development processes as explained in one or more agile methods, such as XP [4] or Scrum [75].

A safety-critical software system can be defined as a system in which failure may "*result in injury to people, damage to the environment or extensive economic losses*" ([78], p. 287). These types of systems are most often found within the domains of the aerospace and medical sectors (S1). In a textbook on software engineering [78], the key issue is that safe systems are developed through an explicit safety engineering process that includes specification of properties, verification and validation processes, and is based on evidence of defined and dependable processes. Cockburn [18] was among the first to acknowledge that agile methods for critical software would require more elaborate processes, which [18] called ceremony. In the Crystal family of methods, "criticality is an important dimension, and safety criticality is understood as degrees of criticality starting in the low end with loss of comfort, then discretionary money, essential money, and life at the high end of the scale" ([18], p. 152). Most of the reviewed research does not distinguish degrees of criticality. If a reviewed paper has a definition of safety criticality, it is close to Sommerville's definition, but most papers do not define it.

The analysis shows that several papers concluded that safety-critical systems can be developed using an agile method (S1), that agile practices do not contradict regulatory requirements (S30), and that it is worthwhile to attempt adopting agile methods in safety-critical software development due to substantial compatibility (S37). Surveys of the industry show that agile methods or at least agile practices have been introduced in safety-critical software development for several years (S24; S34). The literature has reported on several case studies of organisations using a variety of agile practices when developing safety-critical software, documenting that it is possible to increase the role played by agile software development of safety-critical products. Some of the analysed papers even propose that agile methods are better suited for safety-critical software development than the traditional, plan-driven methods and that using agile methods is advantageous (e.g., S39). However, most of the papers also conclude that to fulfil the regulatory requirements using agile methods, the agile methods need to be tailored (e.g., S12) and that there are several challenges in tailoring agile methods to safety-critical development (e.g., S34). Due to these challenges, Górski and Łukasiewicz (S15) argued that agile methods should be regarded as complementary to plan-driven practices instead of as a replacement.

Two concepts are central to the discussion of increasing the use of agile software development in safety-critical contexts: iterative development and incremental development. The reviewed literature does not define these concepts but uses them indiscriminately. To define these concepts, we draw on classical software literature. Iterative development refers to an approach consisting of several cycles [10] and a strategy that focuses on the rework of pieces of the system [18]. Agile methods rely highly on rapid iterations to identify needed changes and handle them in the next iteration. Each iteration consists of the highest priority set of requirements, which is determined through negotiations between the developers and the customer [10]. Incremental development refers to an approach in which the entire system is not delivered at once [10]. It is defined as an approach in which the system is developed as "*a series of versions (increments), with each version adding functionality to the previous version*" ([78], p. 30). The system increment is integrated as it is developed [18]. The system can be developed incrementally and presented to customers for comment without the increment being delivered [78]. The idea is to evolve the system based on customer feedback through a series of versions until an adequate system has been developed. Incremental development can be used both in the plan-driven process and in the agile domain. In a plan-driven process, the increments are identified up front, while the later increments depend on progress and customer feedback in an agile approach [78]. An incremental model combines elements of a linear sequential model with an iterative model, and each increment is developed in accordance with the overall development model [71]. Incremental development is the simpler of the two methods to learn because cutting the project into subprojects is not as tricky as deciding when to stop improving the product [18].

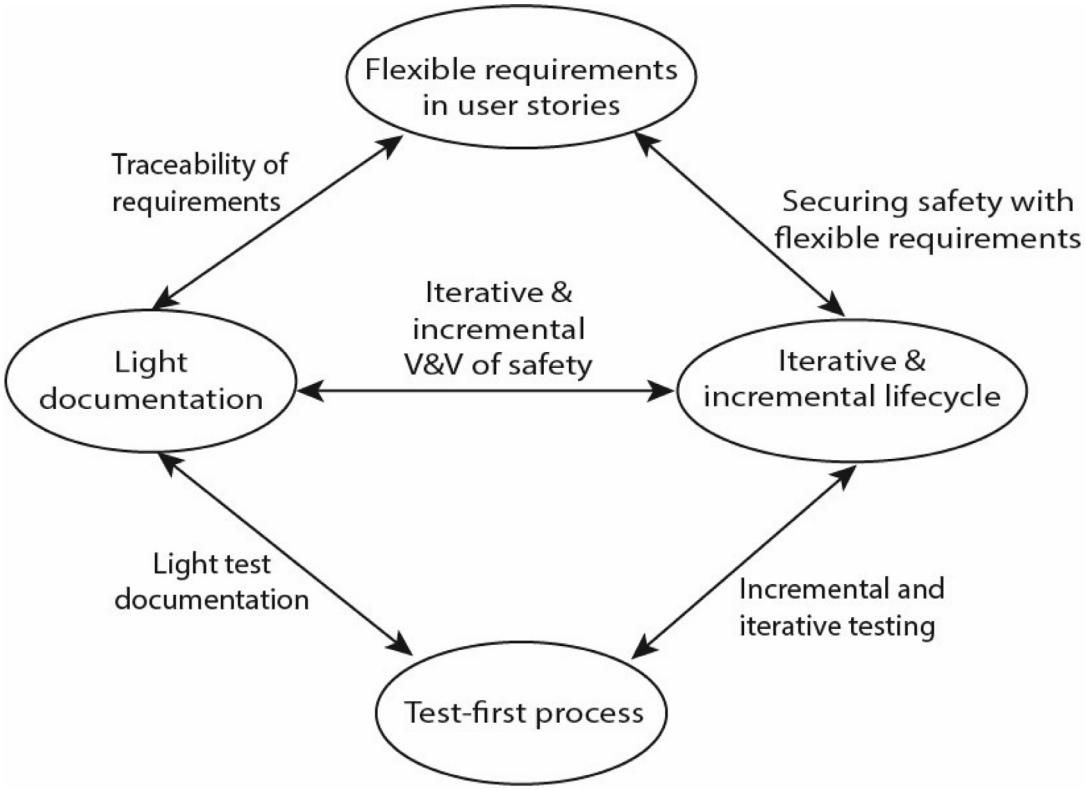

Fig. 6. Problem areas and relationships of agile software development in a safety-critical context.

3.3. Problem areas of agile software development and safety criticality

The analysis of the challenges of agile software development in a safety-critical context showed that the literature focuses on four problematic practice areas understood as issues to deal with the following:

1) Light documentation (35 papers).
2) Flexible requirements written in user stories (33 papers).
3) Iterative and incremental lifecycles (37 papers).
4) Test-first process (32 papers).

Five relationships between these four problem areas are analysed in Section 3.4. Fig. 6 presents a conceptual model that depicts the four problem areas and the five relationships. Based on the coding of the literature, the figure shows the number of papers that identify or address the challenge presented in a problem area and/or a relationship. Appendix B shows a table summarising the problem areas and/or relationships that each of the reviewed papers address.

3.3.1. Problem area: light documentation

A main challenge for increasing the use of agile software development for safety-critical products is the focus on documentation. How documentation can become key in agile development and not an outcast has been discussed in several papers, and these papers point directly to documentation as a main obstacle (e.g., S25; S34; S41; S45). It is argued that agile processes focus on 'working software over comprehensive documentation' [5]. This does not mean that agile methods cast aside all documentation [2], and some agile methods acknowledge that documents can be very useful. Maintaining documentation is not essential in agile software development; instead, documentation must be guided by the principles of 'just enough' or 'barely sufficient' [18]. Moreover, XP advocates that created artefacts or documents must all produce value [4], while Scrum advocates that documents and models should not be optional but be used by developers to structure their thinking [75].

From the standpoint of safety-critical software development, documentation is essential, as the documentation serves as proof that all processes have been adhered to and that the software is safe (S40; S51). The regulatory process standards and requirements regard documentation as a cornerstone to achieving high quality (S21). Compliance with regulations and the fear of regulatory inspections foster heavy documentation of the development (S18). Regulatory agencies responsible for inspection of the software will not agree to less documentation of software requirements and designs (S48), as the limited and sometimes absent focus on documentation in agile methods is insufficient to determine the quality of safety-critical systems (S50). Thus, the heavy focus on documentation in plan-driven development processes decreases the use of agile software development by increasing costs and lowering flexibility (S22).

Some researchers have argued that, due to the flexibility of agile processes, the amount of documentation is not a problem (e.g., S13). Furthermore, agile processes strive to deliver what is requested by the customer, which includes documentation to prove the safety in the case of safety-critical software (S25). To keep the documentation at a minimum, it is important to consider the purpose of the documentation (S17) and determine which knowledge needs to be codified and which knowledge may remain tacit (S33). It has also been found that using tools can support the process of handling a large amount of documentation (S38). Using a documentation sub-team can help separate the documentation from the development and ensure that the developers can focus on the development of the software (S38).

Other characteristics of safety-critical software development point to a need for a larger focus on documentation. The development of safety-critical systems often takes several years, and it is very likely that the project will suffer from multiple personnel replacements. The safety-critical products have a very long lifespan (up to 30 years), and during this time, a third party needs to continuously maintain, upgrade, and improve the software (S11; S42). Hence, safety-critical projects need a focus on documentation, as sufficient documentation needs to be developed to provide verification of the software safety. Documentation

must not provide unreasonable overhead and should be handled in a light manner.

3.3.2. Problem area: flexible requirements written in user stories

The agile and traditional processes used for safety-critical software development differ in two ways in relation to requirement management. First, while agile processes encourage constant change of the requirements (S34; [4]), safety-critical development processes discourage requirement changes due to the increasing costs of the redesign of the software, testing, and documentation of the requirements (S35). Second, agile processes break with the traditional ideas of requirements and, instead, rely primarily on loosely structured requirements, such as user stories written by the customer in a plain business-like language [19].

Securing safety is a requirement management problem, and good requirements that are documented in a complete requirement specification are therefore crucial (S34). When changes happen in a safety-critical project, the changes can have severe consequences on the software architecture (S11) and may weaken the essential proof needed for validation and verification of the safety of the software (S14). However, dealing with changing requirements is a necessity in software development [51]. Even if the original analysis is thorough and even if the requirement specifications are very detailed and several approval signatures are obtained, after some time, changes will happen (S2). These issues are also highly probable in the development of security software (S3). The development of a complex medical device takes several years. During this time, it is very likely that requirements will change, or additional requirements will emerge (S39). Some studies show that changes may be less common compared to the development of less critical software (S14). Requirements can be divided into two categories: safety requirements and functional requirements. While the safety requirements are quite stable, the functional requirements change considerably over time (S45). Extending the original Scrum process by adding a safety-product backlog that holds and handles the safety requirements is thus proposed. This is used as an addition to the typical functional product backlog and serves to separate the frequently changed functional requirements from the more stable safety requirements (S1; S45).

Both XP and Scrum utilise user stories as requirement containers. User stories invite changes to a greater extent than traditional requirements, which too often are taken to symbolise something mandatory [4]. In Scrum, for example, the user stories are collected in the product backlog and prioritised by the product owner [75]. In XP, early estimation of each user story is central, as it assists the interaction between the on-site customer and the developers (S8), and user stories assist in identifying the most valuable and potential stories [4]. To provide an overview of progress and the scope of the project, the stories are to be displayed on large boards placed in the work area, called information radiators [18].

Agile user stories are also recommended in safety-critical software development, as they provoke a discussion between the developers and the customer (S8). User stories are also included in one of the tailored agile methods proposed for safety-critical software development, called method æ. The method is a substitute for the V-model and is a hybrid consisting of both planned and iterative phases, focusing on risk-oriented decision making. The method suggests the use of stories (referred to as an æ story – pronounced 'a nice story'). The stories are a combination of the agile user stories and a traditional requirement specification, including more elements (such as risk assessment) than an agile user story but are limited to a minimum in size (S19). However, as the regulatory standards for safety-critical software development require well-structured requirement engineering (S22; S48), user stories written in plain business-like language cannot be used for validation (S3). Instead, these informal requirements in the user stories must be translated into a formal specification [9]. The use of simple paper cards as suggested by the initial versions of XP is also unlikely to be accepted, while paper cards can be used to create formal visibility documents, and tools must be used to hold the requirements as well (S22). Rottier and Rodrigues (S40) provided an example of how they have operated with use cases collected in a use case document, which was supplemented by the software requirement specification detailing all the non-functional requirements. To separate the functional and safety-critical requirements, it is also suggested to introduce two additional types of user stories: abuser stories (threat scenarios) and security-related user stories (security functionalities) (S5).

3.3.3. Problem area: iterative and incremental lifecycle

The agile and the traditional methods also differ substantially in their recommendations for the project lifecycle. Agile methods advocate the use of iterations that include all phases of the development process to create flexibility and possibilities of adapting to changes [20]. Each iteration should result in a running, tested version of the system (an increment) that is in direct use by the customers [18]. For example, XP introduces the concepts of weekly cycles to plan work a week at a time and uses quarterly cycles to reflect on the work every quarter [4], whereas Scrum implements sprints in which the developers work without interruption [75].

Safety-critical projects are often developed according to the V-model (S19) and are thus not carried out iteratively and incrementally (S14). The V-model is a variation of the waterfall model with a particular focus on quality management, as it matches the diverse types of testing to each stage, planning the testing in parallel with a corresponding development phase (S24). The model is therefore suited for developing safety-critical software, as it produces the necessary deliverables required when seeking regulatory approval (S30). The regulatory process requirements do not prescribe a certain development lifecycle (S23; S25; S29; S40).

As the highly iterative and incremental approach is a central feature of agile processes, this must be maintained when using agile software development in safety-critical contexts. The plan-driven lifecycle models must therefore be adapted to become iterative (S7). Some researchers have found this to be a challenge and therefore a barrier for increasing the use of agile software development (S4; S25; S38). The key challenge is to develop an incremental safety assurance process (S15).

Some studies show that iterative safety-critical development can be performed (S1; S34) and even that it can be advantageous to do so, as developers are forced to break down tasks and obtain a deeper understanding before attempting to solve issues (S20). Using incremental development in safety-critical projects is similar to incremental development in non-critical development, though more difficult (S3). Rasmussen et al. (S39) reported on a team using iterations of 4 to 10 weeks after which they delivered an increment of executable software that was put into use. Thus, as shown by this study, longer iterations than recommended in non-critical software development may be needed. This is supported by Heeager (S20) and Heeager and Nielsen (S21), who reported on developers who were frustrated when having to implement a full increment for 4 weeks. Another study reported how a software development team implemented iterative software development embedded in an overall documentation-driven systems engineering project and that the iterations in software development were influenced by the plan-driven project milestones and sequential process, forcing a specific focus (for example on documentation) onto the iterations (S20).

3.3.4. Problem area: test-first process

Test-driven development is widely used in the agile community [64]. In XP, for example, testing is a core practice in which the test cases are written before the software. All parts of an increment are tested, and completion is determined by the increment passing all tests [4]. Due to the heavy reliance on testing, agile methods focus on automating the tests [4], as it is acknowledged that automated tests are

necessary to complete short iterations [45].

In safety-critical software development, extensive testing is vital (S15). Following the V-model, the testing is done in the final phases of the development (S24). Reconciling these two different testing processes has proven to be difficult, as the developers are not used to focusing on the tests in the initial stages (S20). Other empirical studies report examples of safety-critical software development in which test-first processes have been implemented successfully (S11; S17; S43; S47). A comparative analysis has shown that the test-driven development is compatible with various regulatory standards (S38).

Other research has pointed out that an agile test-first process can be used but must be further developed to fit safety process requirements (S16). In test-driven development, developers write the tests themselves, and this proves problematic, as some standards (such as EN 50,128) require that the tester must be responsible for specifying the test and that the developer and tester must be separate persons (S22). Furthermore, to comply with the strict regulatory process requirements, additional testing must be performed (S15). Incorporating verification techniques is challenging, and these activities are work intensive and reduce the use of agile software development (S38).

3.4. Challenging relationships

A more detailed analysis revealed how challenges arise not only due to the four problem areas but also in the five relationships between these problematic practice areas. The challenges of the relationships are as follows:

- To ensure traceability of the requirements when the documentation is light (17 papers).
- To secure safety when requirements are flexible (22 papers).
- To adopt an iterative and incremental test-first process (14 papers).
- To keep light documentation when testing is iterative and incremental (8 papers).
- To verify and validate the safety of software increments when developed in iterations (23 papers).

None of the reviewed literature provides a detailed and explicit analysis of how these relationships are challenging for agile processes when developing safety-critical software. From the review of the literature, it is possible to combine the findings and present an overview of the existing research on the relationships.

3.4.1. Relationship: traceability of requirements with light documentation

Research has shown how difficult it is to adopt the practices of writing and updating the documentation of requirements in an iterative and incremental manner. Standards such as the US FDA process standards require that software requirements are explicitly documented prior to implementation and testing (S29) and that documents are only changed through controlled procedures (S21). Documents will have to be (partly) rewritten when one or more requirements change (S46). A related aspect is assuring traceability between requirements and all stages of development (S25; S29) as mandated by the safety standards (S34 and S40). The IEEE defines traceability as:

> "The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match." ([44], p. 82)

In several case studies (S8) and surveys of industry practices, traceability issues have been identified as a barrier for using agile software development in safety-critical contexts (S25). Using agile practices, requirements are not fixed before development begins. During development, changes to requirements are welcomed (S25), which makes the process of traceability difficult (S28). The agile principle of prioritising working software over documentation inhibits traceability, as documentation is the primary evidence of traceability (S12).

The reviewed literature points to the idea that traceability can be assured using agile processes by fully documenting all software requirements and changes (S25). This can be simplified by only working with a few features at a time and by updating the documents concurrently (S22). Moreover, Stålhane et al. (S45) suggest introducing the maintenance of documentation and tracing of information as a separate activity in each iteration and working with backlogs for functional requirements and non-functional safety requirements. Tools for creating traceability of agile requirements can also support this practice (S25).

3.4.2. Relationship: securing safety with flexible requirements

The ability to adapt to changing requirements using iterations is considered an advantage of the agile processes (S23). Using agile processes, the requirement management strategy is to deliver the system functionalities with the highest business value and to create value faster (S6). Agile processes are also popular, as they help capture requirement-related problems earlier, for example, unfulfilling, wrong, unrealistic, missing, or unwanted requirements (S36). The linear models, such as the waterfall model and the V-model, are considered risky because rework is costly when requirements change during the development phase (S8).

Organisations developing safety-critical software experience difficulties handling changes in requirements (S27; S30). Requirement management and multiple releases are seen by practitioners as a main difficulty of safety-critical software development (S25). However, many practitioners also see potential benefits of agile, iterative development (S9) because it can be used to avoid major and expensive rewriting of the requirement specifications in the late stages of development (S20).

The reviewed literature suggests that treating requirements in an iterative manner makes it difficult to build and present evidence of safety. Thus, using the agile method of iteratively adding and modifying functionality, the systems will be composed of smaller parts that are added, removed, changed, and integrated with each other over time. This poses a conflict with the verification and validation of system level properties (e.g., safety) (S35). Górski and Łukasiewicz (S16) stated that an iterative approach to requirement management can narrow the scope and undermine the rigour and discipline needed from a safety viewpoint.

Achieving balance between upfront design and just-in-time design in the development of safety-critical software is especially difficult (S24). Several process standards require that requirements are specified prior to their design and implementation (S21), as the upfront design needs to be sufficiently detailed to serve as input to the hazard analysis (S14; S21). For example, the US FDA requires that medical device manufacturers submit high-level requirements prior to beginning development. Therefore, this can only be done once (S30). This challenges the view that safety requirements and hazard analysis can primarily be performed outside of the sprints and iterative cycles (S10). A safety impact analysis conducted before the design and coding phases must verify that the requirements are complete and comprehensive. It is also a requirement of some standards to establish the reliability of the system. However, for a safety impact analysis to be conducted, all requirements must be known up front (S42). Upfront planning conflicts with the agile processes welcoming requirement changes. User stories collected prior to the project have, in one case, served as a form of upfront planning and give the necessary stability to allow a project to start (S25). To solve this, it is suggested to distinguish between critical requirements that are formally specified and less critical requirements that can change (S50). Other studies have found that safety requirements also change and must be revisited and revised during software development (S10).

3.4.3. Relationship: iterative and incremental testing

When developing safety-critical software, each increment must be

fully working, fully tested, and validated before release (S26; S30). Thus, implementing an iterative, incremental lifecycle entails adopting an iterative and early testing process. The most significant verification-related practice in agile software development is test-driven development (S7). In contrast, in a typical waterfall project, the verification is an end-loaded process (S39).

Some studies in the literature review report on successful adoption and integration of an iterative approach using test-driven development (S47). In addition, McHugh et al. (S25) found that the necessary testing can be done once several iterations have been completed. However, adopting iterative and incremental testing in safety-critical software development poses some difficulties:

- The validation of the software: The reviewed literature suggests that, due to the high level of required validation and the quality of documentation, fully testing increments and longer iterations in safety-critical software development are necessary (S40). Moreover, Rasmussen et al. (S39) provides an example in which, given the inherent documentation requirements of medical device development, iterations shorter than 6 weeks did not generate sufficient velocity. Iterations much longer than 8 weeks provide opportunities for loss of organisational focus. Weekly goals were established for each week of a given iteration.
- Hardware-software integration: A high volume of safety-critical software is embedded in a device that cannot be built incrementally like the software. This poses challenges related to the implementation of iterations and incremental development. Because of the dependencies on dedicated hardware, it is very challenging to conduct incremental testing on embedded safety-critical software (S37; S49).
- Changes in work routines of the developers: Changing the work routines of the developers has also proven difficult, as developers that are used to following the V-model are not used to focusing on testing during the development, for instance. In a project developing software for a medical device, the software team was struggling to fully test each increment within the time-boxed iterations. This was, in part, because of interruptions from the other project groups and, in part, because the developers were used to postponing the testing (S21).

3.4.4. Relationship: light documentation of testing

Only eight of the reviewed papers concern the issue of how to do agile documentation of tests, but these papers indicated that producing light documentation of testing is difficult due to the strict requirements from the safety standards. The tests define when a safety solution is of sufficiently high functional quality and is sufficiently safe (S3). The extensive testing needs to document this safety, which results in a large amount of test documentation (S18; S39). In general, detailed documents must be drawn up, and traceability must be maintained, ensuring quality (S23). The US FDA, for example, requires that unit tests, integration tests, system tests, and user site tests are performed, and that test traceability is ensured (i.e., unit tests must be mapped to detailed design, integration tests must be mapped to high-level design, and system tests must be mapped to software requirements). The whole testing process must be supported by documentation of test plans, test procedures, test cases, test reports, and test logs. These process requirements go far beyond what is advocated in agile processes. Being a test-driven approach with several practices and roles dedicated to testing, XP only partly supports the US FDA requirements. In addition, XP documents test plans, test procedures, test logs (kept for debugging purposes), and test cases, but no practice in XP ensures traceability by relating the different artefacts (S31; S32). The automated unit tests suggested by XP can serve as part of the documentation. Moreover, XP does not prohibit documentation; it just warns that it comes at a cost (S17).

3.4.5. Relationship: iterative and incremental verification and validation of safety

Documentation serves as proof that the defined processes have been maintained. To provide such a proof, several types of documentation need to be conducted, kept up to date, and be traceable. Due to the agile iterative and incremental lifecycle, these documents need to be generated within each iteration along with other essential work (S39). The agile methods advocate that the documents are incrementally fill with detailed requirements, test cases, and designs relevant to the current increment (S22). The process of producing such evidence can be conducted concurrently with the software development process. To produce safe software incrementally, the process of creating evidence must also be conducted incrementally (S1) to always have an acceptably safe software system with each release (S14). To achieve this, the evidence of safety must be built incrementally, and the evidence for safety of previous releases can be reused in producing the evidence for the current release. This is difficult because the evidence in general is monolithic and is constructed for complete software systems (S14). It is difficult to foresee the full effect of a change (S44). At this point, several standards require that the evidence is addressed up front with all the requirements and risks (S15).

Case studies have shown how handling documentation and validation of an increment in an iterative manner is challenging, as the documentation continuously needs to be created and updated when needed (S22). A major concern with iterative development processes is change management. When developing systems incrementally, changes are introduced in most iterations, which may invalidate previous work on assuring safety (S22). The regulatory requirements for safety-critical software development are not in direct conflict with the practice of frequently delivering and demonstrating functionality. However, the extensive requirements on documentation, verification, validation, and assessments make it costly to release systems to users (S22). Changing the documents may also become costly, as it is required that documents are changed through controlled procedures (S21). Preparing the software for verification creates overhead, which impedes small incremental changes (S35).

Several studies seek solutions to the issue of incremental validation of the safety. Ge et al. (S14) proposed an approach for developing both the software and evidence iteratively. They suggested that not only must the software be constructed iteratively and incrementally but the argument that the software is acceptably safe must also be constructed in this way to have an acceptably safe software system with each release. This should be done using languages and tools for creating safety arguments. The method is insufficiently described. Thus far, there is little knowledge of possible implementations (S15). Some studies suggest treating documents like source code and applying continuous integration to ensure that they are kept up to date (S49). Tool support is suggested to handle a larger amount of iterative documentation (S13). Another solution found is that not every iteration releases a new increment of the software (as this is not required by agile methods). This allows a development team to implement short iterations of development and longer iterations and increments of validation (S25), thus addressing the issue of incremental validation and verification by introducing minor and major iterations (S38).

4. Discussion

The analysis of the research literature on agile development of safety-critical software has primarily resulted in the conceptual model in Fig. 6 This conceptual model features the structure of the research literature on agile development of safety-critical software. The conceptual model summarises all the literature, and it is reasonable to state that there are four problem areas for agile processes, agile practices, and challenges met when seeking to adopt and adapt these processes and practices to the special application domain of safety-critical software systems. In addition to the four problem areas, five relationships

are challenged.

The conceptual model is a contribution at an overall level to (i) provide an overview of the literature that we did not previously have as part of the existing body of knowledge. (ii) It is important to point out that these are the four most important problem areas and the five most important relationships. In addition, (iii) other issues exist (e.g., culture clashes and human aspects), but the literature points to these as less important. In understanding the ramifications of the conceptual model, there are also contributions at a more detailed level, for example, the suggestions of solutions provided in the literature. We will select the most significant of the problem areas and relationships for a detailed discussion about challenges and solutions.

First, the following observations of the existing research literature concern two of the four problem areas:

- Light documentation is a challenge that several practitioners seem to be struggling with, and it seems very necessary to deal with (37 of the reviewed papers explicitly mention documentation as a main barrier). We suggest that it has too much focus because heavy documentation is explicitly mentioned in the agile manifesto as something to leave behind. The research focusing particularly on safety-critical software development acknowledges the need for documentation and advocates a more balanced view, considering the special needs of quality assurance (cf. Section 3.3.1).
- Flexible requirements are challenging in the development of safety-critical software to a much larger degree than in less critical agile development (cf. Section 3.3.2). Some requirements (perhaps mostly functional requirements) are easy to change and should remain easy to change and then develop. Other requirements (most safety requirements) need to be changed in a more controlled way. It is not always as easy without overhead or without a defined process. Where the efficient process in non-critical agile development consists of (i) just change the requirement, (ii) just implement, and (iii) refactoring the code, if needed [4,21], the process for safety-critical software needs to include more elaborate parts answering questions such as the following: (i) Is this easy to change or difficult to change? (ii) What is the value to safety? (iii) How will it affect other parts? (iv) Is it an incremental addition or an iterative rework? (S1; S14; S45).

Second, while the problem areas are important, the relationships are perhaps even more important. The relationships are crucially interesting and should be understood in the following way. The problem areas are connected, and they are connected in complex ways in which (i) they are mutually dependent and (ii) are also specific to the topic. The problem areas are mutually dependent in the sense that one cannot expect to change one problem area without influencing another problem area through the relationships. If, for example, the approach to requirements is changed, it will then influence both the 'light documentation' and 'incremental and iterative lifecycle'. It can also have a reverse effect; that is, trying to change one problem area and ignoring the problem areas that depend on it will slow the change down and possibly even obstruct the change. For example, trying to change the way safety-critical requirements are handled may quickly meet resistance in how light documentation is developed. In this way, the four problem areas are tied closely with through the five relationships. The relationships are also specific, and we can go to the research literature to see the particulars of the relationships. For example, it matters considerably to be informed that the relationship between 'requirements' and 'light documentation' is a question of traceability. With that in mind, there is a way forward if we want to plan and conduct change. Regarding how we deal with safety requirements knowing that light documentation must track the requirements incrementally, we must maintain traceability between requirements and other parts of the documentation no matter how light that documentation is.

The following observations on the relationships are the most interesting:

- Traceability of requirements with light documentation is interesting because no matter how much the requirements become changeable (both iteratively reworking and incrementally adding requirements), there must be traceability (cf. Section 3.4.1). That has ramifications for how the documentation can be built and conditions a significant part of the contents of the light documentation. In effect, it conditions how light the documentation can be (S8; S25).
- The iterative and incremental validation and verification of safety is interesting for another reason. This relationship is concerned with keeping light documentation of the quality assurance aspects of the development process (cf. Section 3.4.5). The implication is significant; if the documentation of the process is too light, does not have the required contents, or is not produced in a sequence, or is produced with dependencies that are not in accordance with regulations, then the iteration with its rework is the only way out. It is suggested in the literature that process documentation should be produced incrementally (S1; S14).
- Securing safety with iterative requirements is key because it links changeable requirements with a process that must include an understanding of how the requirements may change both in terms of adding more requirements incrementally and changing existing requirements through a managed process (cf. Section 3.4.2). This requires striking balances between upfront design and flexible requirements and between validation and verification before or after implementation of the software and between integrating all requirements and separating safety requirements from other requirements.

We also notice that the research literature on safety-critical software development is rich in challenges for agile processes and that there are few proven solutions to these challenges. This may be caused by the recent inception of agile development for safety-critical software where the interest has started, but it has not reached a final level yet. Much of the early research that we have found is in the form of experience reports that are less mature in their research and in addressing relevant matters and solutions. The more substantial findings are still sporadic, despite the impression that problems and challenges are widespread in the industry of safety-critical software. It could potentially influence the state of the art if software companies are afraid of regulatory agencies and simply do not want to risk a discussion where the regulatory agencies are in complete control. To the extent that we have found solutions to the observed problems and challenges, we have yet to see a vast body of systematic case studies, a trial of novel solutions, and robust empirical evidence.

It is clear from the literature review that we need more research on this topic. We may ask: What is the advice to software companies if they are developing safety-critical software? How should they document for regulatory agencies that the quality assurance of safety is under control? We may also ask whether regulatory agencies are setting up the right and the best requirements for assuring safety and quality in a broader sense. It is the whole industry that depends on these regulatory agencies, and it matters what they are requesting and that their requirements are leading to safe software products.

We suggest that future research should have a primary focus on the relationships. We suggest, at a more detailed level, that the following propositions be investigated in future research:

1) **The relationships tie the problem areas to each other, and one problem area cannot be changed without influencing the other.** As we have suggested above, this is a generalisation built directly on the existing literature, and in that sense, it includes most of the literature. It also points to research to be done because, with the conceptual model, we can now ask more detailed questions. For example, how strong are the ties and how can we change and

improve the problem areas through pushing the ties in a direction? This is by far the most significant and far-reaching proposition, as it contains the dynamics of the whole field of agile development of safety-critical software. The following depicts relationships that are interesting and worth researching further:

2) **Incremental development seems better suited than iterative development for safety-critical software.** The existing research literature is not always conceptually clear on the difference between incremental and iterative development. Much agile literature does not even make the distinction but uses the terms in a conglomerate concept as 'incremental and iterative development' (e.g. [50]) as conceptualised in Section 3.1. We suggest, based on the above analysis, that the distinction is important for safety-critical software development because incremental development seems to be better suited than iterative development. There is an indication of this when we look at the incremental safety assurance process (S1; S14) (cf. Section 3.4.5) and incremental documentation of safety concerns (e.g., S22) (cf. Section 3.4.1). Iteration on the other hand is to change previous requirements and engage in rework (S8; [18]). There is an indication that what is difficult is iteration (S15; S35). Hence, it will be interesting to investigate how that may lead to a better development practice based on a more precise distinction between incremental and iterative development, which is more open to safety assurance work.
3) **Quality assurance can create value, and it can be sufficient and minimal, but not without extra effort.** Agile processes are strong on producing value and on not spending time on anything that is not contributing value to the software product. The agile manifesto, XP, and Scrum have no concern for documentation *per se* for that reason. The claim is that documentation adds no value to the software. From the viewpoint of safety-critical software development, there must be documentation to a level, with content that can be defended (S21), yet the agile movement and its manifesto should perhaps be considered a strong reaction to too much documentation and to unreasonable documentation (S13; [2,18]) (cf. Section 3.3.1). Hence, it is necessary to develop a better understanding of which documentation will prove invaluable in a development project. We suggest that we need to measure the value of the quality assurance and what is sufficient and determine how to measure sufficiency, thus producing minimal documentation to be used in the quality assurance and documenting the quality assurance to third parties. In continuation of Proposition 2, we further suggest that we need to measure value to develop documentation incrementally rather than iteratively. There is no reason to believe that this burden of documentation will occur as a side effect of the agile development. It cannot be produced without extra effort, and in the understanding of minimal documentation, it will be relevant to measure the amount of documentation work to compare with development work and to compare the velocity of both documentation and development.

With these propositions, we offer a basis for further research into this topic of agile software development safety-critical contexts.

5. Conclusion

The objective of this study was to understand how agile processes can be used in safety-critical software development. Research has been published on this topic since 2001, and our analysis showed how this research field is starting to mature. We found that a literature review accumulating the knowledge on the topic was needed to provide guidance for future research.

The analysis of the research literature on agile development of safety-critical software has primarily resulted in the conceptual model (Fig. 6), depicting the structure of the research literature. It shows that the literature focuses on four problematic practice areas. Light documentation is a great challenge that receives much attention in the literature. This is problematic, but we suggest that this topic has gained too much attention. A second observation on the problematic areas is that flexible requirements are challenging in the development of safety-critical software to a much larger degree than in less critical agile development, and this is worth studying. In addition to the four problem areas, there are five relationships that are challenged, which our analysis suggests are even more important than the problematic areas. In relation to the relationships, we made three interesting observations:

- Traceability of changeable requirements is challenging and has ramifications for documentation,
- The iterative and incremental validation and verification of safety is perhaps the most challenging relation.
- Securing safety with iterative requirements requires balances between upfront design and flexible requirements, between validation and verification before or after implementation of the software, and between integrating all requirements and separating safety requirements from other requirements.

As previous research that has focused on the problematic areas, relationships, and dynamics of the whole conceptual model seems to be of the most importance, we suggest that future research should have a primary focus on the relationships. Research should focus on the dynamics of the whole field of agile development of safety-critical software, as our study showed that the relationships tie the problem areas to each other and that one problem area cannot be changed without influencing the other. The analysis also indicates that incremental development is better suited than iterative development for safety-critical software. We suggest that future research based on a more precise distinction between incremental and iterative development should investigate how that may lead to better development, which is more open to safety assurance work. The third contribution is that quality assurance can create value and can be sufficient and minimal, but not without extra effort. To advance knowledge on how to keep the effort minimal but sufficient, we suggest that the value of the quality assurance needs to be measured. Hence, we need to investigate what is sufficient and how to measure sufficiency.

List of included studies

S1. Abdelaziz AA, El-Tahir Y, Osman R (2015) Adaptive Software Development for developing safety critical software. In: Proceedings of the 1st International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), Khartoum, Sudan, IEEE, pp 41–46.

S2. Bedoll R (2003) A Tail of Two Projects: How 'Agile' Methods Succeeded after 'Traditional' Methods Had Failed in a Critical System-Development Project In: Proceedings of the 3rd Conference on Extreme Programming and Agile Methods, New Orleans, LA, USA, Springer, pp 25–34.

S3. Beznosov K (2003) Extreme Security Engineering: On Employing XP Practices to Achieve 'Good Enough Security' without Defining It X. In: Proceedings of the 1st ACM Workshop on Business Driven Security Engineering (BizSec), Fairfax, USA, Citeseer, pp 1–7.

S4. Beznosov K, Kruchten P (2004) Towards agile security assurance. In: Proceedings of the 17th Workshop on New Security Paradigms, Victoria, Canada, ACM. pp 47–54.

S5. Boström G, Wäyrynen J, Bodén M, Beznosov K, Kruchten P (2006) Extending XP practices to support security requirements engineering. In: Proceedings of the 28th International Workshop on Software Engineering for Secure Systems (SESS), Shanghai, China, ACM, pp 11–17.

S6. Cordeiro L, Barreto R, Barcelos R, Oliveira M, Lucena V, Maciel P (2007) TXM: an agile HW/SW development methodology for building medical devices. ACM SIGSOFT Software Engineering Notes 32 (6):4.

S7. Del Bianco V, Stosic D, Kiniry JR (2010) Agile Formality: A Mole of Software Engineering Practices. In: Proceedings of the 2nd International Workshop on Formal Methods and Agile Methods, Pisa, Italy, pp 29–48.

S8. Demissie S, Keenan F, McCaffery F (2016) Investigating the Suitability of Using Agile for Medical Embedded Software Development. In: Proceedings of the 16th International Conference on Software Process Improvement and Capability Determination (SPICE), Springer, pp 409–416.

S9. Doss O, Kelly T (2016) The 4 + 1 Principles of Software Safety Assurance and Their Implications for Scrum. In: Proceedings of the 14th International Conference on Agile Software Development, Edinburgh, United Kingdom, Springer, pp 286–290.

S10. Doss O, Kelly T (2016) Challenges and Opportunities in Agile Development in Safety Critical Systems: A Survey. ACM SIGSOFT Software Engineering Notes 41 (2):30–31.

S11. Drobka J, Noftz D, Raghu R (2004) Piloting XP on four mission-critical projects. IEEE Software 21 (6):70–75.

S12. Fitzgerald B, Stol K-J, O'Sullivan R, O'Brien D (2013) Scaling agile methods to regulated environments: An industry case study. In: Proceedings of the 35th International Conference on Software Engineering (ICSE), San Francisco, USA, IEEE Press, pp 863–872.

S13. Gary K, Enquobahrie A, Ibanez L, Cheng P, Yaniv Z, Cleary K, Kokoori S, Muffih B, Heidenreich J (2011) Agile methods for open source safety-critical software. Software: Practice and Experience 41 (9):945–962.

S14. Ge X, Paige RF, McDermid JA (2010) An iterative approach for development of safety-critical software and safety arguments. In: Proceedings fo the Agile Conference (AGILE), Orloando, Florida, USA, IEEE, pp 35–43.

S15. Górski J, Łukasiewicz K (2012) Assessment of risks introduced to safety critical software by agile practices-a software engineer's perspective. Computer Science 13 (4):165–182.

S16. Górski J, Łukasiewicz K (2013) Towards Agile Development of Critical Software. In: Proceedings of the 3rd International Workshop on Software Engineering for Resilient Systems, Kiev, Ukraine, Springer, pp 48–55.

S17. Grenning J (2001) Launching extreme programming at a process-intensive company. IEEE Software 18 (6):27.

S18. Hajou A, Batenburg R, Jansen S (2015) An Insight into the Difficulties of Software Development Projects in the Pharmaceutical Industry. Lecture Notes on Software Engineering 3 (4):267.

S19. Hajou A, Batenburg R, Jansen S (2015) Method æ, the Agile Software Development Method Tailored for the Pharmaceutical Industry. Lecture Notes on Software Engineering 3 (4):251.

S20. Heeager L (2012) Introducing Agile Practices in a Documentation-Driven Software Development Practice: A Case Study. Journal of Information Technology Case and Application Research 14 (1):3–24.

S21. Heeager L, Nielsen P (2009) Agile Software Development and its Compatibility with a Document-Driven Approach? A Case Study. In: Proceedings of the 20th Australasian Conference on Information Systems, Melbourne, Australien, pp 205–214.

S22. Jonsson H, Larsson S, Punnekkat S (2012) Agile practices in regulated railway software development. In: Proceedings of the 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW), Dallas, Texas, IEEE, pp 355–360.

S23. Lin W, Fan X (2009) Software development practice for FDA-compliant medical devices. In: Proceedings of the International Joint Conference on Computational Sciences and Optimization (CSO 2009), Hainan, Sanya, China, IEEE, pp 388–390.

S24. McCaffery F, Trektere K, Ozcan-Top O (2016) Agile – Is it Suitable for Medical Device Software Development? In: Proceedings of the International Conference on Software Process Improvement and Capability Determination (SPICE), Dublin, Ireland, Springer, pp 417–422.

S25. McHugh M, McCaffery F, Casey V (2012) Barriers to adopting agile practices when developing medical device software. In: Proceedings of the 12th International Conference on Software Process Improvement and Capability Determination (SPICE), Palma de Mallorca, Spain, Springer, pp 141–147.

S26. McHugh M, McCaffery F, Casey V (2012) Barriers to using agile software development practices within the medical device industry. In: Proceedings of the 19th European Systems and Software Process Improvement and Innovation Conference, Vienna, Austria, Springer.

S27. McHugh M, Cawley O, McCaffery F, Richardson I, Wang X (2013) An agile v-model for medical device software development to overcome the challenges with plan-driven software development lifecycles. The 5th International Workshop on Software Engineering in Health Care (SEHC), IEEE, pp 12–19.

S28. McHugh M, McCaffery F, Fitzgerald B, Stol KJ, Casey V, Coady G (2013) Balancing agility and discipline in a medical device software organisation. In: Proceedings of the International Conference on Software Process Improvement and Capability Determination, Springer, pp 199–210.

S29. McHugh M, McCaffery F, Casey V (2014) Adopting agile practices when developing software for use in the medical domain. Journal of Software: Evolution and Process 26 (5):504–512.

S30. McHugh M, McCaffery F, Coady G (2014) An Agile Implementation within a Medical Device Software Organisation. In: Proceedings of the International Conference on Software Process Improvement and Capability Determination, Springer, pp 190–201.

S31. Mehrfard H, Hamou-Lhadj A (2013) The impact of regulatory compliance on Agile software processes with a focus on the FDA guidelines for medical device software. International Journal of Information System Modeling and Design, 2(2): pp 67–81.

S32. Mehrfard H, Pirzadeh H, Hamou-Lhadj A (2010) Investigating the capability of agile processes to support life-science regulations: the case of XP and FDA regulations with a focus on human factor requirements. In: Proceedings of the 8th Conference of Software Engineering Research, Management and Applications (SERA), Montreal, Canada, Springer, pp 241–255.

S33. Misra S, Kumar V, Kumar U (2010) Identifying some critical changes required in adopting agile practices in traditional software development projects. International Journal of Quality & Reliability Management 27 (4):451–474.

S34. Notander JP, Höst M, Runeson P (2013) Challenges in flexible safety-critical software development–an industrial qualitative survey. In: Proceedings of the 14th International Conference on Product Focused Software Process Improvement, Paphos, Cyprus, Springer, pp 283–297.

S35. Notander JP, Runeson P, Höst M (2013) A model-based framework for flexible safety-critical software development: a design study. In: Proceedings of the 28th Annual ACM Symposium on Applied

Computing, Coimbra, Portugal, ACM, pp 1137–1144.

S36. Notander JP, Runeson P, Höst M (2013) SimPal: a design study on a framework for flexible safety-critical software development. ACM SIGAPP Applied Computing Review 13 (4):17–29.

S37. Paige RF, Chivers H, McDermid JA, Stephenson ZR (2005) High-integrity extreme programming. In: Proceedings of the 30th ACM symposium on Applied computing, Santa Fe, New Mexico, ACM, pp 1518–1523.

S38. Paige RF, Charalambous R, Ge X, Brooke PJ (2008) Towards agile engineering of high-integrity systems. In: Proceedings of the 27th International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Newcastle upon Tyne, UK, Springer, pp 30–43.

S39. Rasmussen R, Hughes T, Jenks J, Skach J (2009) Adopting agile in an FDA regulated environment. In: Proceedings of the 2009 Agile Conference (AGILE), Chicago. USA, IEEE, pp 151–155.

S40. Rottier PA, Rodrigues V (2008) Agile development in a medical device company. In: Proccedings of the 2008 Agile Conference (AGILE), Toronto, Canada, IEEE, pp 218–223.

S41. Shafiq S, Minhas NM (2014) Integrating Formal Methods in XP—A Conceptual Solution. Journal of Software Engineering and Applications, 7(4): pp 299–310.

S42. Sidky A, Arthur J (2007) Determining the applicability of agile practices to mission and life-critical systems. In: Proccedings of the 31st Software Engineering Workshop (SEW), Columbia, USA, IEEE, pp 3–12.

S43. Spence JW (2005) There has to be a better way! In: Proccedings of the Agile Development Conference (ADC' 05), Denver, USA, IEEE, pp 272–278.

S44. Stephenson Z, McDermid J, Ward A (2006) Health modelling for agility in safety-critical systems development. In: Proceedings of the 1st International Conference on System Safety, London, UK, Institution of Engineering and Technology (IET), pp 260–265.

S45. Stålhane T, Myklebust T, Hanssen G (2012) The Application of Safe Scrum to IEC 61,508 Certifiable Software, In: Proceedings of the 12th European Safety and Reliability Conference (ESREL), Helsinki, Taylor and Francis, pp 6052–6061.

S46. Stålhane T, Katta V, Myklebust T (2013) Scrum and IEC 60,880. The 37th Enlarged Halden Reactor Project meeting, Storefjell, Norway.

S47. VanderLeest SH, Buter A (2009) Escape the waterfall: Agile for aerospace. In: Proceedings of the 28th Conference on Digital Avionics Systems, Orlanda, USA, IEEE, pp 6–16.

S48. Vogel D (2006) Agile Methods: Most are not ready for prime time in medical device software design and development. DesignFax Online, July, pp 1–6.

S49. Wils A, Van Baelen S, Holvoet T, De Vlaminck K (2006) Agility in the avionics software world. In: Proceedings of the 7th International Conference on Extreme Programming and Agile Processes in Software Engineering, Oulu, Finland, Springer, pp 123–13.

S50. Wolff S (2012) Scrum goes formal: Agile methods for safety-critical systems. In: Proceedings of the 1st International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, Zurich, Schwitzerland, IEEE Press, pp 23–29.

S51. Wäyrynen J, Bodén M, Boström G (2004) Security Engineering and eXtreme Programming: An Impossible Marriage? In: Proceedings of the 4th Conference on Extreme Programming and Agile Methods, Calgary, Canada, Springer, pp 117–128.

Appendix A: Overview of papers

Study	Year	Research method	Publication type	Agile method	Specific domain	Embedded software	Outcome
S1	2015	Normative writing	Conference	–	Avionics	No	A model combining agile and safety-critical methods
S2	2003	Experience report	Conference	–	Avionics	No	Practical advice on becoming agile
S3	2003	Normative writing	Workshop	XP	–	No	Tailored XP to fit security
S4	2004	Theoretical	Workshop	–	–	No	Mapping of security methods / techniques and agile methods
S5	2006	Laboratory experiment	Workshop	XP	–	No	Extension of XP to fit security
S7	2007	Design study	Notes	–	Medical	Yes	An agile methodology for medical devices (TXM)
S8	2010	Design study	Workshop	–	–	Yes	A blended method (the Mole)
S9	2016	Normative writing	Conference	–	Medical	Yes	Suitability of agile methods for developing medical devices
S10	2016	Normative writing	Conference	Scrum	–	No	Security principles for Scrum
S11	2016	Survey	Notes	–	–	No	Practitioner opinions of challenges
S12	2004	Experience report	Magazine	XP	Radio communication	Yes	Experiences with XP in mission-critical development
S13	2013	Design study	Conference	Scrum	–	No	Develop and test a combined Scrum method (R-Scrum)
S14	2011	Case study	Journal	–	Medical	No	Practical experience on agile for safety-critical
S15	2010	Design study	Conference	–	Avionics	No	Propose and test an iterative approach
S16	2012	Case study	Journal	XP & Scrum	Medical	No	Risks and solutions for overcoming these
S17	2013	Case study	Workshop	XP & Scrum	Medical	No	Investigating risks
S18	2001	Experience report	Magazine	XP	–	Yes	Experiences with XP for development of safety-critical
S20	2015	Case study	Journal	–	Medical	No	Expert evaluation of agile in pharmaceutical projects

S21	2015	Design study	Journal	XP & Scrum	Medical	No	A tailored method for medical projects (method æ)
S22	2012	Case study	Journal	Scrum	Medical	Yes	Understanding possibilities and challenges of agile in medical
S23	2009	Case study	Conference	Scrum	Medical	Yes	Evaluation of agility of a safety-critical development practice
S24	2012	Theoretical	Workshop	–	Railway	No	Mapping agile practices with the EN 50,128 standard
S25	2009	Field experiment	Conference	–	Medical	No	A hybrid methodology used in practice
S26	2016	Normative writing	Conference	–	Medical	No	Challenges of using agile for developing medical devices
S27	2012	Survey	Conference	–	Medical	No	Barriers to agile adoption for medical
S28	2012	Survey	Conference	–	Medical	No	Identification of barriers in literature & practice, compare
S30	2013	Design study	Workshop	–	Medical	No	Developing an SDLC founded in plan-driven but with agile
S31	2013	Case study	Conference	–	Medical	No	Evaluation and improvement of agility in practice
S32	2014	Survey	Journal	–	Medical	No	Identification of barriers (internal and external)
S33	2014	Design study	Conference	–	Medical	No	Development and validation of the AV-model
S34	2013	Theoretical	Journal	XP	Medical	No	Comparison of XP and FDA
S35	2010	Theoretical	Conference	XP	Medical	No	Extension of XP to comply with FDA
S36	2010	Survey	Conference	–	–	No	Identification of issues for regulatory compliance
S37	2013	Case study	Conference	–	Robotics Transportation Automation Aerospace	No	Identification of industrial needs and challenges
S38	2013	Design study	Conference	–	Robotics	No	Evaluation of a model-based, agile framework (SimPal)
S39	2013	Design study	Journal	–	Robotics	No	Evaluation and improvement of a framework (SimPal)
S40	2005	Normative writing	Symposium	XP	–	No	Assessment of XP
S41	2008	Case study	Conference	–	Avionics	No	Identification of challenges of applying agile
S42	2009	Experience report	Conference	–	Medical	Yes	Description of experiences adopting agile for medical
S43	2008	Field experiment	Conference	Scrum	Medical	No	Experiences adopting Scrum for medical
S44	2014	Laboratory experiment	Journal	XP	Police reporting	No	Development and test of a formal XP method
S45	2007	Normative writing	Workshop	–	–	No	Process for identifying agile practices for critical projects
S46	2005	Field experiment	Conference	–	Medical	No	How agile was implemented
S47	2006	Normative writing	Conference	–	–	No	Propose a process model (the agile health model)
S48	2012	Theoretical	Conference	Scrum	Real time operation	No	Develop and theoretically evaluate the SafeScrum model
S49	2013	Theoretical	Workshop	Scrum	Nuclear	No	Theoretical evaluation of the Safe Scrum model
S50	2009	Case study	Conference	–	Avionics	Yes	Show how agile practices can be used in Airspace
S51	2006	Normative writing	Magazine	–	Medical	No	Discussing the compliance of agile for medical
S52	2006	Theoretical	Conference	XP	Avionics	Yes	How to increase speed and handle changing requirements
S53	2012	Experience report	Workshop	Scrum	–	No	Propose how formality can be embedded in Scrum
S54	2004	Theoretical	Conference	XP	–	No	Evaluation of XP for safety development

Appendix B: Conceptual matrix

Table B1
Conceptual matrix.
C1 = documentation, C2 = requirements, C3 = lifecycle, C4 = testing; I1 = relation between documentation & requirements, I2 = relation between requirements & lifecycle, I3 = relation between lifecycle & testing, I4 = relation between documentation & testing, I5 = relation between documentation & lifecycle.

#	Paper	C1	C2	C3	C4	I1	I2	I3	I4	I5
1.	Abdelaziz et al. [1]		X	X						x
2.	Bedoll [6]		X	X	X					
3.	Beznosov [7]		X	X					X	
4.	Beznosov and Kruchten [8]	X		X	X					
5.	Boström et al. [14]	X	X	X	X	X	X			
6.	Cordeiro et al. [22]		X	X	X		X			
7.	Del Bianco et al. [25]				X			X		X
8.	Demissie et al. [26]	X	X	X		X	X			
9.	Doss and Kelly [27]			X			X			
10.	Doss and Kelly [28]	X	X	X			X			X
11.	Drobka et al. [29]	X	X	X	X					X
12.	Fitzgerald et al. [30]	X		X	X	X				
13.	Gary et al. [31]	X	X	X						
14.	Ge et al. [32]		X	X	X		X			X
15.	Górski and Łukasiewicz [33]	X			X					X
16.	Górski and Łukasiewicz [34]		X		X		X			
17.	Grenning [35]	X		X	X	X			X	
18.	Hajou et al. [37]	X				X			X	
19.	Hajou et al. [38]	X	X	X	X	X		X		
20.	Heeager [39]	X	X	X	X		X			
21.	Heeager and Nielsen [40]	X	X	X	X			X		
22.	Jonsson et al. [46]	X	X	X	X	X				X
23.	Lin and Fan [52]	X					X		X	
24.	McCaffery et al. [53]	X	X		X		X			X
25.	McHugh et al. [55]	X	X	X		X	X	X		X
26.	McHugh et al. [56]	X	X			X	X	X		X
27.	McHugh et al. [54]		X	X	X		X			X
28.	McHugh et al. [60]			X						
29.	McHugh et al. [57]	X		X		X				X
30.	McHugh et al. [59]		X	X	X		X			X
31.	Mehrfard and Hamou-Lhadj [61]	X	X		X		X	X	X	
32.	Mehrfard et al. [62]	X		X	X	X			X	
33.	Misra et al. [63]	X								
34.	Notander et al. [65]	X	X			X				
35.	Notander et al. [66]	X	X	X						X
36.	Notander et al. [67]	X		X			X			X
37.	Paige et al. [69]			X	X			X		
38.	Paige et al. [68]	X		X	X		X	X		X
39.	Rasmussen et al. [72]		X	X				X	X	X
40.	Rottier and Rodrigues [73]	X	X	X	X	X	X	X		X
41.	Shafiq and Minhas [76]	X	X	X	X			X		
42.	Sidky and Arthur [77]	X	X		X		X			
43.	Spence [79]			X	X			X		
44.	Stephenson et al. [80]		X	X	X					X
45.	Stålhane et al. [82]	X	X	X		X	X			X
46.	Stålhane et al. [81]	X	X	X	X	X				X
47.	VanderLeest and Buter [85]	X		X	X			X		X
48.	Vogel [86]	X	X			X				
49.	Wils et al. [89]	X			X		X	X	X	X
50.	Wolff [90]		X	X	X	X	X			
51.	Wäyrynen et al. [93]	X	X		X					
Total		35	33	37	32	17	22	14	8	23

References

[1] A. Abdelaziz, Y. El-Tahir, R. Osman, Adaptive software development for developing safety critical software, Proceedings of the International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), Khartoum, Sudan, IEEE, 2015, pp. 41–46.

[2] S.W. Baker, Formalizing agility: an agile organization's journey toward CMMI accreditation, Proceedings of the the Agile Development Conference, Denver, USA, IEEE Computer Society, 2005, pp. 185–192.

[3] W. Bandara, E. Furtmueller, E. Gorbacheva, S. Miskon, J. Beekhuyzen, Achieving rigour in literature reviews: insights from qualitative data analysis and tool-support, *Comm. AIS. Information* 37 (1) (2015) 154–204.

[4] K. Beck, C. Andres, *Extreme Programming explained: Embrace change*, Addison-Wesley, Boston, USA, 2004.

[5] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R.C. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas, "Manifesto for agile software development, (2001). available at http://agilemanifesto.org/ (accessed 26 January 2018).

[6] R. Bedoll, A Tail of Two Projects: How 'Agile' Methods Succeeded after 'Traditional' Methods Had Failed in a Critical System-Development Project, Proceedings of the

Conference on Extreme Programming and Agile Methods, Springer, 2003, pp. 25–34.
[7] K. Beznosov, Extreme security engineering: on employing xp practices to achieve 'good enough security' without defining it X, Proceedings of the The First ACM Workshop on Business Driven Security Engineering, BizSec in Fairfax, VA, USA, 2003, pp. 1–7.
[8] K. Beznosov, P. Kruchten, Towards agile security assurance, Proceedings of the 2004 Workshop on New Security Paradigms, Nova Scotia. Canada, ACM, 2004.
[9] S. Black, P. Boca, J. Bowen, J. Gorman, M. Hinchey, Formal versus agile: survival of the fittest, Computer 42 (9) (2009) 37–45.
[10] B. Boehm, R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley Professional, Boston, 2004.
[11] B. Boehm, R. Turner, Balancing agility and discipline: a guide for the perplexed, *J. Prod. Innov. Manag.* 22 (2) (2005) 216–218.
[12] B.W. Boehm, R. Ross, Theory-W software project management principles and examples, *IEEE Trans. Softw. Eng.* 15 (7) (1989) 902–916.
[14] G. Boström, J. Wäyrynen, M. Bodén, K. Beznosov, P. Kruchten, Extending XP practices to support security requirements engineering, Proceedings of the 2006 international workshop on Software engineering for secure systems in Shanghai, China, ACM, 2006, pp. 11–17.
[15] A. Carugati, W. Fernández, L. Mola, C. Rossignoli, My choice, your problem? Mandating IT use in large organisational networks, *Inform. Syst. J.* 28 (1) (2018) 6–47.
[16] O. Cawley, X. Wang, I. Richardson, Lean/agile software development methodologies in regulated environments–state of the art, *Lean Enterpr. Softw. Syst.* 65 (1) (2010) 31–36.
[17] M. Chrissis, M. Konrad, S. Shrum, CMMI Guidelines For Process Integration and Product Improvement, Addison-Wesley Longman Publishing Co., Boston, MA, USA, 2003.
[18] A. Cockburn, Agile Software development: The Cooperative Game, Addison-Wesley Professional, Boston, USA, 2006.
[19] M. Cohn, User Stories applied: For agile Software Development, Addison-Wesley Professional, Boston, USA, 2004.
[20] M. Cohn, D. Ford, Introducing an agile process to an organization, Computer 36 (6) (2003) 74–78.
[21] K. Conboy, Agility from first principles: reconstructing the concept of agility in information systems development, *Inform. Syst. Res.* 20 (3) (2009) 329–354.
[22] L. Cordeiro, R. Barreto, R. Barcelos, M. Oliveira, V. Lucena, P. Maciel, TXM: an agile HW/SW development methodology for building medical devices, *ACM SIGSOFT Softw. Eng. Notes* 32 (6) (2007) 4.
[23] G. Davis, Strategies for information requirements determination, *IBM Syst. J.* 21 (1) (1982) 4–30.
[24] I. de Sousa Santos, R.M. de Castro Andrade, L.S. Rocha, S. Matalonga, K.M. de Oliveira, G.H. Travassos, Test case design for context-aware applications: are we there yet? *Inform. Softw. Technol.* 88 (August) (2017) 1–16.
[25] V. Del Bianco, D. Stosic, J.R. Kiniry, Agile formality: a mole of software engineering practices, Proceedings of the FM+ AM, 2010, pp. 29–48.
[26] S. Demissie, F. Keenan, F. McCaffery, Investigating the suitability of using agile for medical embedded software development, Proceedings of the International Conference on Software Process Improvement and Capability Determination (SPICE), Dublin, Ireland, Springer, 2016, pp. 409–416.
[27] O. Doss, T. Kelly, The 4+1 principles of software safety assurance and their implications for scrum, Proceedings of the International Conference on Agile Software Development, Springer, 2016, pp. 286–290.
[28] O. Doss, T. Kelly, Challenges and opportunities in agile development in safety critical systems: a survey, *ACM SIGSOFT Softw. Eng. Notes* 41 (2) (2016) 30–31.
[29] J. Drobka, D. Noftz, R. Raghu, Piloting XP on four mission-critical projects, *IEEE Softw.* 21 (6) (2004) 70–75.
[30] B. Fitzgerald, K.-J. Stol, R. O'Sullivan, D. O'Brien, Scaling agile methods to regulated environments: an industry case study, Proceedings of the 2013 International Conference on Software Engineering, San Francisco, USA, IEEE Press, 2013, pp. 863–872.
[31] K. Gary, A. Enquobahrie, L. Ibanez, P. Cheng, Z. Yaniv, K. Cleary, S. Kokoori, B. Muffih, J. Heidenreich, Agile methods for open source safety-critical software, *Softw.: Pract. Exp.* 41 (9) (2011) 945–962.
[32] X. Ge, R.F. Paige, J.A. McDermid, An iterative approach for development of safety-critical software and safety arguments, Proceedings of the Agile Conference (AGILE), Orlando, Florida, IEEE, 2010, pp. 35–43.
[33] J. Górski, K. Łukasiewicz, Assessment of risks introduced to safety critical software by agile practices-a software engineer's perspective, *Comp. Sci.* 13 (4) (2012) 165–182.
[34] J. Górski, K. Łukasiewicz, Towards agile development of critical software, Proceedings of the International Workshop on Software Engineering for Resilient Systems, Springer, 2013, pp. 48–55.
[35] J. Grenning, Launching extreme programming at a process-intensive company, *IEEE Softw.* 18 (6) (2001) 27.
[36] A. Hajou, R. Batenburg, S. Jansen, How the pharmaceutical industry and agile software development methods conflict, Proceedings of t14th International Conference on Computational Science and Its Applications, 2014, pp. 40–48.
[37] A. Hajou, R. Batenburg, S. Jansen, An insight into the difficulties of software development projects in the pharmaceutical industry, *Lecture Notes Softw. Eng.* 3 (4) (2015) 267.
[38] A. Hajou, R. Batenburg, S. Jansen, Method æ, the agile software development method tailored for the pharmaceutical industry, *Lecture Notes Softw. Eng.* 3 (4) (2015) 251.
[39] L. Heeager, Introducing agile practices in a documentation-driven software development practice: a case study, *J. Inform. Technol. Case Appl. Res.* 14 (1) (2012) 3–24.
[40] L. Heeager, P. Nielsen, Agile software development and its compatibility with a document-driven approach? A case study, Proceedings of the Australasian Conference on Information Systems, Melbourne, Australia, 2009, p. 205.
[41] L. Heinemann, G.A. Fleming, J.R. Petrie, R.W. Holl, R.M. Bergenstal, A.L. Peters, Insulin pump risks and benefits: a clinical appraisal of pump safety standards, adverse event reporting, and research needs a joint statement of the european association for the study of diabetes and the american diabetes association diabetes technology working group, Diabetes Care 38 (4) (2015) 716–722.
[42] D. Hoyle, ISO 9000 Quality Systems Handbook, Butterworth-Heinemann, Oxford, UK, 2006.
[43] W.S. Humphrey, Managing the Software Process, Addison-Wesley Longman Publishing Co, 1990.
[44] IEEE 1990. IEEE standard glossary of software engineering terminology. *610-12-1990*.
[45] C. Jakobsen, K. Johnson, Mature agile with a twist of CMMI, Proceedings of the The Agile Conference, Toronto, Canada, IEEE Computer Society, 2008, pp. 212–217.
[46] H. Jonsson, S. Larsson, S. Punnekkat, Agile practices in regulated railway software development, Proceedings of the 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, 2012, pp. 355–360.
[47] B. Kitchenham, R. Pretorius, D. Budgen, O.P. Brereton, M. Turner, M. Niazi, S. Linkman, Systematic literature reviews in software engineering–a tertiary study, *Inform. Softw. Technol.* 52 (8) (2010) 792–805.
[48] B.A. Kitchenham, D. Budgen, O.P. Brereton, The value of mapping studies - a participant-observer case study, Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, Keele, UK, 2010, pp. 25–33.
[49] J. Kjeldskov, C. Graham, A review of mobile HCI research methods, Proceedings of the International Conference on Mobile Human-Computer Interaction, Udine, Italy, Springer, 2003, pp. 317–335.
[50] C. Larman, Agile and Iterative development: a Manager's Guide, Addison-Wesley Professional, Boston, USA, 2004.
[51] G. Lee, W. Xia, Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility, MIS Quarterly 34 (1) (2010) 87–114.
[52] W. Lin, X. Fan, Software development practice for FDA-compliant medical devices, Proceedings of the International Joint Conference on Computational Sciences and Optimization (CSO), Hainan, Sanya, China, IEEE, 2009, pp. 388–390.
[53] F. McCaffery, K. Trektere, O. Ozcan-Top, Agile–is it suitable for medical device software development? Proceedings of the International Conference on Software Process Improvement and Capability Determination (SPICE), Dublin, Ireland, Springer, 2016, pp. 417–422.
[54] M. McHugh, O. Cawley, F. McCaffery, I. Richardson, X. Wang, An agile v-model for medical device software development to overcome the challenges with plan-driven software development lifecycles, Proceedings of the The 5th International Workshop on Software Engineering in Health Care (SEHC), San Francisco, USA, IEEE, 2013, pp. 12–19.
[55] M. McHugh, F. McCaffery, V. Casey, Barriers to adopting agile practices when developing medical device software, Proceedings of the International Conference on Software Process Improvement and Capability Determination, Palma de Mallorca, Spain, Springer, 2012, pp. 141–147.
[56] M. McHugh, F. McCaffery, V. Casey, Barriers to using agile software development practices within the medical device industry, In: International Conference on Software Process Improvement and Capability Determination, Springer, Vienna, Austria, 2012.
[57] M. McHugh, F. McCaffery, V. Casey, Adopting agile practices when developing software for use in the medical domain, *J. Softw. Evol. Proc.* 26 (5) (2014) 504–512.
[58] M. McHugh, F. McCaffery, V. Casey, M. Pikkarainen, Integrating agile practices with a medical device SDLC, Proceedings of European Systems and Software Process Improvement and Innovation Conference, EuroSPI, Vienna, Austria, 2012.
[59] M. McHugh, F. McCaffery, G. Coady, An agile implementation within a medical device software organisation, Proceedings of the International Conference on Software Process Improvement and Capability Determination, Vilnius, Lithuania, Springer, 2014, pp. 190–201.
[60] M. McHugh, F. McCaffery, B. Fitzgerald, K.J. Stol, V. Casey, G. Coady, Balancing agility and discipline in a medical device software organisation, Proceedings of the International Conference on Software Process Improvement and Capability Determination, Springer, 2013, pp. 199–210.
[61] H. Mehrfard, A. Hamou-Lhadj, The impact of regulatory compliance on Agile software processes with a focus on the FDA guidelines for medical device software, Int. J. Inf. Syst. Modeling and Design 2 (2) (2011) 67–81.
[62] H. Mehrfard, H. Pirzadeh, A. Hamou-Lhadj, Investigating the capability of agile processes to support life-science regulations: the case of XP and FDA regulations with a focus on human factor requirements, Proceedings of the 8th Conference on Software Engineering Research, Management and Applications, Springer, Montreal, Canada, 2010, pp. 241–255.
[63] S. Misra, V. Kumar, U. Kumar, Identifying some critical changes required in adopting agile practices in traditional software development projects, *Intern. J. Quality Reliab. Manag.* 27 (4) (2010) 451–474.
[64] S. Nerur, R. Mahapatra, G. Mangalaraj, Challenges of migrating to agile methodologies, *Commun. ACM* 48 (5) (2005) 73–78.
[65] J.P. Notander, M. Höst, P. Runeson, Challenges in flexible safety-critical software development–an industrial qualitative survey, Proceedings of the International Conference on Product Focused Software Process Improvement, Paphos, Cyprus, Springer, 2013, pp. 283–297.
[66] J.P. Notander, P. Runeson, M. Höst, A model-based framework for flexible safety-critical software development: a design study, Proceedings of the 28th Annual ACM

Symposium on Applied Computing, Coimbra, Portugal, ACM, 2013, pp. 1137–1144.
[67] J.P. Notander, P. Runeson, M. Höst, SimPal: a design study on a framework for flexible safety-critical software development, *ACM SIGAPP Appl. Comp. Rev.* 13 (4) (2013) 17–29.
[68] R.F. Paige, R. Charalambous, X. Ge, P.J. Brooke, Towards agile engineering of high-integrity systems, Proceedings of the International Conference on Computer Safety, Reliability, and Security, Springer, Newcastle upon Tyne, UK, 2008, pp. 30–43.
[69] R.F. Paige, H. Chivers, J.A. McDermid, Z.R. Stephenson, High-integrity extreme programming, Proceedings of the The ACM symposium on Applied computing, Santa Fe, New Mexico, ACM, 2005, pp. 1518–1523.
[70] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, *Inform. Softw. Technol.* 64 (1-18) (2015).
[71] R.S. Pressman, *Software Engineering—A practitioner's Approach*, McGraw-Hill Publishing Company, Maidenhead, UK, 2010.
[72] R. Rasmussen, T. Hughes, J. Jenks, J. Skach, Adopting agile in an FDA regulated environment, Proceedings of the Agile Conference (AGILE), Chigaco, USA,, IEEE, 2009, pp. 151–155.
[73] P.A. Rottier, V. Rodrigues, Agile development in a medical device company, Proceedings of the Agile Conference (AGILE), Toronto, Ontario Canada, IEEE, 2008, pp. 218–223.
[74] G. Schryen, Writing qualitative IS literature reviews—Guidelines for synthesis, interpretation and guidance of research, *Commun. AIS* 37 (Art 12) (2015) 286–325.
[75] K. Schwaber, M. Beedle, Agile Software Development With Scrum, Prentice Hall, Upper Saddle River, New Jersey, USA, 2001.
[76] S. Shafiq, N.M. Minhas, Integrating formal methods in XP—A Conceptual Solution, *J. Softw. Eng. Appl.* 7 (4) (2014) 299.
[77] A. Sidky, J. Arthur, Determining the applicability of agile practices to mission and life-critical systems, Proceedings of the Software Engineering Workshop (SEW), Columbia, USA, IEEE, 2007, pp. 3–12.
[78] I. Sommerville, Software Engineering, Addison-Wesley, Harlow, UK, 2015.
[79] J. Spence, There has to be a better way!, Proceedings of the Agile Development Conference (ADC'05), Denver, USA, IEEE, 2005, pp. 272–278.
[80] Z. Stephenson, J. McDermid, A. Ward, Health modelling for agility in safety-critical systems development, Proceedings of the First Institution of Engineering and Technology International Conference on System Safety, IET, 2006, p. 260-265.
[81] T. Stålhane, V. Katta, T. Myklebust, Scrum and IEC 60880, Proceedings of the Enlarged Halden Reactor Project meeting, Storefjell, Norway, 2013 pp.
[82] T. Stålhane, T. Myklebust, G. Hanssen, The application of safe scrum to IEC 61508 certifiable software, Proceedings of the European Safety and Reliability Conference (ESREL), Helsinki, Finland, 2012, pp. 6052–6061.
[83] M. Templier, G. Paré, A framework for guiding and evaluating literature reviews, *Commun. AIS Information* 37 (1) (2015) 6.
[84] U. S. Department of Health, H., Services, FDA U.S. Food and Drug Administration, U.S. Department of Health and Human Services, 2010.
[85] S.H. VanderLeest, A. Buter, Escape the waterfall: Agile for aerospace, Proceedings of the 28th Digital Avionics Systems Conference, Orlanda, USA, IEEE, 2009, pp. 6–16.
[86] D. Vogel, Agile methods: Most are not ready for prime time in medical device software design and development, DesignFax Online, (July, 2006, 1-6).
[87] J. vom Brocke, A. Simons, K. Riemer, B. Niehaves, R. Plattfaut, A. Cleven, Standing on the shoulders of giants: Challenges and recommendations of literature search in information systems research, *Commun. AIS Information* 37 (9) (2015) 205–224.
[88] J. Webster, R.T. Watson, Analyzing the past to prepare for the future: writing a literature review, MIS Quarterly 26 (2) (2002) 13–23.
[89] A. Wils, S. Van Baelen, T. Holvoet, K. De Vlaminck, Agility in the avionics software world, Proceedings of the International Conference on Extreme Programming and Agile Processes in Software Engineering, Oulu, Finland, Springer, 2006, pp. 123–132.
[90] S. Wolff, Scrum goes formal: agile methods for safety-critical systems, Proceedings of the Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches, IEEE Press, 2012, pp. 23–29.
[91] J.F. Wolfswinkel, E. Furtmueller, C.P. Wilderom, Using grounded theory as a method for rigorously reviewing literature, *Eur. J. Inform. Syst.* 22 (1) (2013) 45–55.
[92] J.L. Wynekoop, S.A. Conger, A review of computer aided software engineering research methods, Information Systems Research: Contemporary Approaches & Emergent Traditions, IFIP TC8 WG 8.2 Working Conference, Copenhagen, Denmark, 1992.
[93] J. Wäyrynen, M. Bodén, G. Boström, Security engineering and extreme programming: an impossible marriage? *Lecture Notes Comp. Sci.* (2004) 117–128.
[94] R.E. Zultner, TQM for technical teams, *Commun. ACM* 36 (10) (1993) 79–91.