Software Quality Journal (2022) 30:1067–1088
https://doi.org/10.1007/s11219-022-09593-2

Scrum for safety: an agile methodology for safety-critical software systems

Mario Barbareschi[2] · **Salvatore Barone**[1] · **Riccardo Carbone**[1] · **Valentina Casola**[1]

Accepted: 23 May 2022 / Published online: 23 July 2022

Abstract
In the last years, agile methodologies are gaining substantial momentum, becoming increasingly popular in a broad plethora of industrial contexts. Unfortunately, many obstacles have been met while pursuing adoption in secure and safe systems, where different standards and operational constraints apply. In this paper, we propose a novel agile methodology for the development and innovation of safety-critical systems. In particular, we developed an extension of the well-known Scrum methodology and discussed the complete workflow. We finally validated the applicability of the proposed methodology over a real case study from the railway domain.

Keywords Agile software development · Agile processes · Software development · Safety-critical software systems

1 Introduction

Nowadays, computer systems are responsible for every aspect of our lives. Ranging from the life-sciences to the transportation field, they currently cover a central role to assure people and environment integrity. Nevertheless, the deployment of digital devices in

Mario Barbareschi, Salvatore Barone, Riccardo Carbone, and Valentina Casola are authors contributed equally to this work.

✉ Riccardo Carbone
riccardo.carbone@unina.it

Mario Barbareschi
m.barbareschi@rfi.it

Salvatore Barone
salvatore.barone@unina.it

Valentina Casola
valentina.casola@unina.it

[1] Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Via Claudio 21, Naples 80125, Italy

[2] Research and Development, Rete Ferroviaria Italiana SpA, Piazza della Croce Rossa 1, Rome 00161, Italy

safety-critical domains remains a very complex task, since this requires the adoption of rigorous and expensive quality assurance processes from international standards, such as those from the International Electrotechnical Commission (IEC) and European Committee for Electrotechnical Standardization (CENELEC) (International Electrotechnical Commission, 2010; International Organization for Standardization, 2011; Comité européen de normalisation en électronique et en électrotechnique, 1999).

For what pertains to software, engineers of each field have developed, during the years, many safe versions of the waterfall process model (Sommerville, 2015). That is exemplified by the V-Model, proposed by CENELEC (European Committee for Electrotechnical Standardization) EN50128 standard (Comité européen de normalisation en électronique et en électrotechnique, 2011).

In order to correctly apply a waterfall-based process, a company needs to: (i) work with stable user requirements; (ii) have in-depth knowledge of the delivered product, and (iii) establish clear interfaces between the various software and hardware subsystems (Sommerville, 2015; Boehm & Turner, 2003; Cockburn, 2006). These hypotheses are not difficult to guarantee when working with known products, but they will become hard to follow within the conditions of an innovation process. Indeed, the innovation of a safe system is very complicated, since uncertain safety risks could arise from new technologies. Moreover, critical systems could be made of many parts, requiring the coordination and work of many experts.

In that situation, engineers have to work together, in order to explore the side effects of every new solution before starting to use it. Here an agile methodology could help companies to improve team communication and efficiency during the entire research project. However, agility still encounters obstacles when referring to secure and safe systems, where the adoption of automatic tools to design, develop, test and continuously integrate components is conflicting with the need of copying with strict standards, that mainly refer to traditional waterfall models.

Indeed, some pioneering works concerning the adoption of agile methods in the safety-critical domain concluded, hastily, that the former and the latter are incompatible (Cawley et al., 2010; Hajou et al., 2014). Nevertheless, more recent results questioned this conclusion, identifying four main challenges inherently arising while adopting agile methods in the safety-critical context, i.e.: (i) documentation, since it is not essential in agile software development; (ii) requirements, since traditional safety-critical development processes discourage requirement changes (Notander et al., 2013); (iii) project life-cycle, since safety-critical projects are developed neither iteratively nor incrementally (Ge et al., 2010), and (iv) testing, which, in the safety-critical context, is done only at final stages of the development (McCaffery et al., 2016).

As for the adoption of agile methodologies in the development of secure systems, authors in (Casola et al., 2020) introduced a novel methodology to extend the DevOps approach towards secure systems. They mainly implemented an automated security-by-design approach that can be easily mapped with the well-known Scrum framework (Schwaber & Sutherland, 2020).

In this paper, we propose an extension of the Scrum agile methodology, namely Scrum for Safety (S4S), to guide and help research & development (R&D) groups involved in the design of safe solutions for the railway domain. In this domain, software development is not linear as expected, yet the output of a regular mediation between multiple stakeholders, heterogeneous complex technologies and mandatory regulations to be satisfied. Therefore, an agile-based process instead of a classical waterfall has been developed.

The reminder of this paper is structured as follows. In Sect. 2 we review the current state of the art. In Sect. 3 we present our novel proposal for the adoption of Scrum in the development of safe software. In Sect. 4 we illustrate the applicability of our methodology over a real case-study from the railway domain. Finally, in Sect. 5 we will discuss some conclusions and future work.

2 Related works

In order to identify possible related work, we started analyzing the application of agile methodologies in different domains, critical for security and safety. We located many scientific works with a focus on different open issues, and we also located some (few) approaches towards the adoption of the well-known Scrum methodology in safety critical domains.

2.1 Agile adoption in critical domains

Among the open issues identified in the scientific literature, documentation is considered one of the major barrier hindering the adoption of agile methods in the safety-critical context (McHugh et al., 2012; Misra et al., 2010; Stålhane et al., 2012).

Indeed, regulatory agencies responsible for inspecting of software do not agree to less documentation of software requirements and design (Vogel, 2006), since this makes hard to determine the quality of systems (Wolff, 2012). The scientific literature, however, empirically proved that the documentation is not a problem, since agile processes strive to deliver what is requested by the customer, which includes evidence to prove the safety of critical software (Gary et al., 2011; McHugh et al., 2012). In addition, in order to keep that evidence at a minimum, the purpose of the documentation itself must be considered, determining which knowledge needs to be expressed (Grenning, 2001; Misra et al., 2010).

Testing is yet another aspect that seriously limits the adoption of agile methods in the safety critical context, since incorporating verification techniques is challenging, and these activities are work intensive (Paige et al., 2008). Indeed, while test-driven development is widely used in the agile community (Nerur et al., 2005), in safety-critical software development, instead, testing is done only in the final phases (McCaffery et al., 2016). Moreover, some standards, such as the CENELEC EN50128 (Comité européen de normalisation en électronique et en électrotechnique, 2011), mandate that the testers must be responsible for specifying the test and that developers and testers must be separate persons (Jonsson et al., 2012). This is in contrast with test-driven development, which requires developers to write the tests themselves. The literature reports examples of safety-critical software development in which test-first processes have been implemented successfully (Drobka et al., 2004; Górski & Łukasiewicz, 2013). In (Vander-Leest & Buter, 2009), for instance, authors proposed a test-aware development process: test developers are involved in the development of requirements, in order to ensure that the latter are testable at the needed level. This allows mitigating the risk of requirement changes due to untestable requirements.

2.2 Scrum-based methodologies

Recently, the Scrum framework has been profitably used in a variety of contexts, including military (Messina et al., 2016; Benedicenti et al., 2016), railway (Myklebust et al., 2015) and aerospace (Smith et al., 2019). Furthermore, some recent works employed the framework to formalize better-articulated methodologies, such as R-Scrum (Fitzgerald et al., 2013) and Safe-Scrum (Hanssen et al., 2018).

The first is a comprehensive description of how the Ireland company QUMAS Inc adopted the Scrum framework to develop software for the pharmaceutical domain. The work represents a revelatory case study for companies interested in adopting an agile quality management process in regulated fields. However, the paper did not address how to fit the proposed techniques in the context of large projects, which are composed of different subsystems, and in the presence of other parallel processes deputed to analyze and control safety (Comité européen de normalisation en électronique et en électrotechnique, 1999). In addition, there is no discussion of how the process documents and artifacts assist the final certification process.

The second work represents the result of a theoretical work in which Scrum has been brought into compliance with various standards in the critical systems world, including the IEC 61508 (International Electrotechnical Commission, 2010) and the CENELEC EN50128 (Comité européen de normalisation en électronique et en électrotechnique, 2011). Although the entire work presented a strong relationship with some industrial standards, it must be corroborated with practical demonstrations about its main benefits. Moreover, given the difficulties of testing independence and documentation management, the authors proposed the adoption of test-driven development and the presence of a dedicated team for software documentation, which may be not very efficient and applicable for vital systems.

Towards the related works, our purpose is to define a novel agile process in order to:

1. discuss how agility can help the innovation of safety-critical products, in order to improve the efficiency and safety of research projects;
2. merge the core concepts of R-Scrum and SafeScrum, including a more consistent proposal for documentation and agile quality management;
3. expand the current empirical evidence on the possibility and advantages of using agility in safety-critical domains, with a real-world case study extracted from the railway domain.

3 Scrum for safety

Scrum for Safety (S4S) aims to guide and help research & development groups involved in the exploration of effective, efficient and safe solutions in the railway domain. Nevertheless, it can be adopted in every domain in which safety must be considered. Indeed, in the R&D context, software development is not a linear and graceful activity, rather the output of a regular mediation between multiple actors, various complex technologies and strict regulations to satisfy. Therefore, an agile-based process allows engineers to rapidly explore and validate every single possibility before taking any crucial decision.

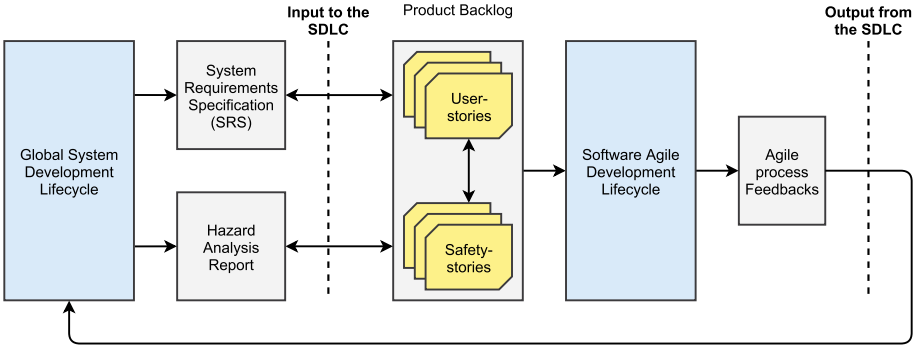

Fig. 1 S4S integration with the global system life-cycle

In this Section, we provide the reader with full details concerning our proposed methodology, including its context, principles, roles, workflow, and metrics.

3.1 Context

The S4S context is not limited to a single software project. Actually, in the safety-critical world, a system is made of various hardware and software components. In the railway domain, for instance, an entire signalling infrastructure is typically broken into small parts. Each of these is then developed following the CENELEC V-Model (Comité européen de normalisation en électronique et en électrotechnique, 1999, 2011, 2003). Therefore, S4S was constructed in order to work inside the context of a global system project, as described in Fig. 1.

As shown, the software life-cycle is embedded into the global system process, with a set of well established relationships between the two. This separation has the first advantage of helping the Scrum team to adapt and change his approach, with no side effects on other system parts. The second advantage, as defined in SafeScrum (Myklebust et al., 2015), is that it creates the fundamental isolation of non-agile activities, such as the RAMS[1] life-cycle (Comité européen de normalisation en électronique et en électrotechnique, 1999), at the system level.

Regarding the inputs needed by the S4S agile process, the two most important artifacts are the *System Requirements Specification (SRS)* and the *Hazard Analysis Report*. The first contains the definition of the user requirements that have been allocated to the software. The second, instead, includes the specification of the safety countermeasures that developers have to consider during the design and implementation of their functions. These two documents constitute the core set of requirements used to define the *Scrum Product Backlog*.

The Product Backlog is the expression of what the developers have to implement in order to fulfil an essential set of system functions. Moreover, for its specification, team members can employ dedicated safety stories in order to maintain a clear trace of safety requirements life-cycle.

[1] Reliability, Availability, Maintainability and Safety.

For what pertains to the outputs of the S4S agile process, there is the need for developers to return regular feedbacks to system engineers if some errors were discovered inside the defined requirements. Feedbacks are the most valuable output of the agile process, since they represent an actual validation of user requirements.

3.2 Principles

S4S embraces all the agile core principles (Beck et al., 2001) and Scrum values (Schwaber & Sutherland, 2020), yet extends them with some new objectives, which are derived from critical software requirements. Results of this extension are the following eight principles, which constitutes the main base of the development process:

1. *Cover all the alternatives before making some decision*: all architectural or detailed design decisions must be preceded by an in-depth evaluation of all possible options. Valuable solutions could be cut-off if one restricts and focus its attention on a single alternative;
2. *Experiment and fail frequently*: the best way to evaluate the effect of a single design choice is to try it, and potentially fail. System modeling and simulation are good tools, but developers have also to implement and test their solutions on target architectures to prove their real effectiveness;
3. *Deliver software continuously to the users*: as soon as the research starts to produce partially implemented software architectures, principal financiers have to begin a review of the achieved results. This is one of the core values of agile where the stakeholders have a central role in the development process (Beck et al., 2001);
4. *Integrate software continuously with other actors*: large and complex projects are frequently broken into small and much more controllable ones. In that case, coordination and periodic integration activities between the various development teams could anticipate a great number of subsequent incompatibilities among the developed subsystems;
5. *Continuously Verify & Validate*: in order to release software in critical environments, where a single failure could potentially cause loss of human lives, environmental pollution or huge economic losses, each developed subsystem must be meticulously verified and validated. Verification & Validation (V&V) are two core activities for Software Quality Assurance (SQA): they must be done in order to increase our trust that the developed product satisfies its specification, and it is adequate to resolve the original research purpose. Furthermore, V&V activities have to be applied continuously, possibly when the research work reaches a new development step, to rapidly identify and manage deviations that could affect critical properties, such as security and safety;
6. *Make your work traceable*: a trace of all the done work for the currently developed software must be always present and available to developers. In that way, all the principal design decision and software architecture are visible, and they determine the basis for the subsequent work. Moreover, traceability constitutes the only way to prove to an independent *Assessor* how the risks related to software functionalities were identified and mitigated. Otherwise, it is impossible to observe and appreciate the fundamental design choices for the final product;
7. *Let your approach be risk-based*: finding, covering and monitoring risks related to software functionalities constitutes a vital activity for critical software. A risk-based approach is much more effective than a “no one”, since it explicitly identifies and

Table 1 CENELEC EN50128:2011 defined roles (from https://link.springer.com/chapter/10.1007%2F978-3-030-85347-1_10)

Role	Responsibilities
Project Manager	Creating the development team; Defining the project scheduling and milestones; Overseeing the respect of the defined schedules; Monitoring the quality of the produced software.
Requirements Manager	Managing the requirements engineering process; Specify the software requirements.
Designer	Select the design principles, techniques and tools; Specify the software architecture.
Implementer	Transform the software design in code; Applying a coding standard; Maintain the codebase under version control; Integrate the software with the target hardware.
Configuration Manager	Managing all the defined software configurations.
Tester	Managing the software testing process; Select the testing techniques and tools; Communicate any observed deviation of software from its specification to the "Change Management Body".
Integrator	Managing the software integration process; Specify a set of test suites for each integration activity; Communicate any observed deviation of software from its specification to the "Change Management Body".
Verifier	Overseeing the software verification process; Check the adequacy of the testing and integration processes to the verification objectives; Guarantee the independence of the verification process; Communicate any observed deviation of software from its specification to the "Change Management Body".
Validator	Validate the software requirements specification; Check the compliance of the development process with the CENELEC standards; Check the correctness and consistency of the verification process; Check the existence and correctness of a unique trace between the requirements and the software implementation; Check that all the hazard are classified, identified and correctly mitigated; Communicate any identified problem to the "Change Management Body".
Assessor	Check the conformity of the developed software and the adopted development process with the CENELEC standards; Verifying the adequacy of the V&V; Verifying the organization personnel competence; Verifying the correctness of the software quality assurance process adopted; Verifying the correctness of the configuration management process adopted; Check that all the hazard are classified, identified and correctly mitigated.

addresses all the software failures which could cause tangible damages to people or to the environment;

8. *Don't break or lose the already achieved quality*: as the work proceeds, and the SQA techniques improve its internal and external quality, it becomes essential to preserve it. In particular, new requested changes to the software must not conflict or undermine the already done risk management activities and software implemented functions.

3.3 Roles

Table 1 reports the professional figures described in CENELEC standards. It is noticeable that the Scrum basic roles of *Product Owner*, *Scrum Master* and *Development Team* (Rubin, 2012) fit perfectly with figures related to software development, i.e., *Requirements Managers*, *Designers* and *Implementers*, while others not frequently mentioned roles—like *Managers*—correspond for responsibilities to *Project Managers*.

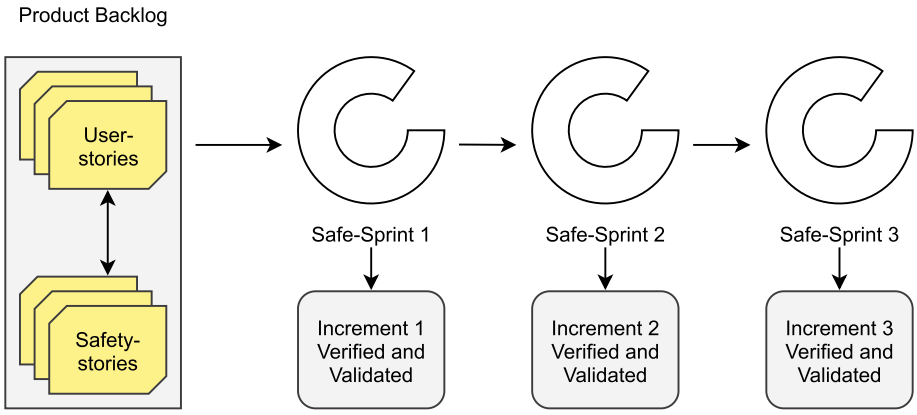

Fig. 2 S4S Workflow

Despite this evident matching, the Scrum framework does not provide any professional independent figure related to the Software Quality Assurance (SQA) process (Rubin, 2012). Therefore, since the importance that SQA has for critical software, *Verifiers*, *Validators* and *Assessors* roles have to be introduced, in order to be able to check for software technical quality, and its adequacy to the original problem. In particular, S4S extends the set of Scrum roles with these figures, which are strictly related to critical software development, while adapting their activity in an agile perspective. Thus, as we will observe after, V&V and Assessment activities are potentially performed at the end of each iteration, providing rapid identification of possible compliance and safety issues.

Another fundamental point regards the independence of *Integrators* and *Testers* from software implementation. Although in the CENELEC standards domain, tests written by programmers could be accepted by the *Verifier* whether they are adequate and completely specified (Comité européen de normalisation en électronique et en électrotechnique, 2011), verification independence is considered crucial for safety-critical products. Therefore, S4S prescribes that during each Scrum sprint, the same developer cannot cover both implementation and verification activities. In that way, design considerations will not influence *Testers* or *Integrators* judgment.

3.4 Workflow

3.4.1 Safe-sprints

The fundamental concept of the S4S workflow is the Safe-Sprint. As described in Fig. 2, a Safe-Sprint is a time-boxed iteration that produces a new software increment verified and validated against the applicable standards. It effectively defines the way used by the Scrum team to check and monitor software quality. Nevertheless, the concept is not new and was first introduced in R-Scrum (Fitzgerald et al., 2013).

In the R-Scrum agile process, what makes a sprint *safe*, i.e., adequate for safety-critical software, are three key factors. The first one is called *Sprint hardening*, which expresses that the output of every iteration has to contain the needed user documentation and proof-of-conformance required for software assessment. Hardening means, therefore, releasing a new validated software version at the end of each sprint.

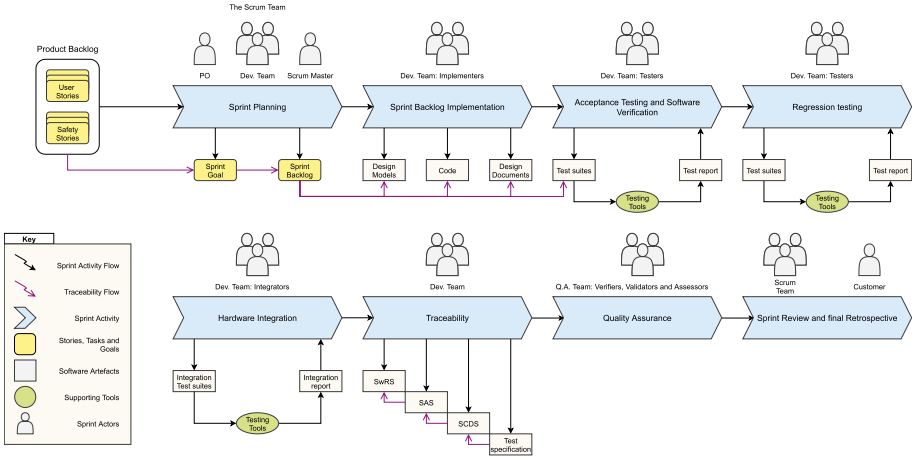

Fig. 3 S4S Safe-Sprint Structure

The second factor is *Continuous compliance*, which summarizes that the product has to be continuously subjected to V&V activities. Continuous compliance is essential for the Scrum team to monitor and assure the technical quality of the software during the incremental flow of the agile process. In particular, the regular application of quality assurance techniques helps to discover and fix critical bugs rapidly.

The third factor is *Living traceability*, which defines the possibility at any time to generate a clear trace about the user requirements implementation. Living traceability has the purpose of making the entire development process more accessible to external people. Therefore, it covers an important role in certification.

Thereafter, the concept of Safe-Sprint evolved inside the context of Safe-Scrum project (Hanssen et al., 2018). Here it was extended to work with complex safety-critical systems made of many parts and comply with various sector standards. Such of these are the IEC 61508 (International Electrotechnical Commission, 2010) and the EN 50128 (Comité européen de normalisation en électronique et en électrotechnique, 2011).

With S4S, our objective is to inherit and further extend the Safe-Sprint concept by (i) enforcing the independence between developers and testers during the iteration, and (ii) defining a practical approach for documentation management. We consider, in fact, these aspects two fundamental pillars of critical software, for which more discussion is needed.

In the next subsection, we provide the reader with a depth view of the structure of our Safe-Sprint.

3.4.2 Structure of a safe-sprint

The entire workflow of our Safe-Sprint is described in Fig. 3. As shown, the first step of every iteration is the *Sprint Planning*. Here, the scope remains the same as in Scrum, i.e., to select a group of manageable and most important stories (the Sprint Backlog), starting from a clear and feasible objective (the Sprint Goal) (Rubin, 2012). However, team members must distribute the work with the same philosophy of the CENELEC EN50128 V-Model (Comité européen de normalisation en électronique et en électrotechnique, 2011). No one can cover both design and testing activities; otherwise, verification could fail its

critical objective. Therefore, in S4S, those who are responsible for testing and integration cannot participate in the software design process. Practically, this avoids tests written by programmers, considering it crucial to not restrict verification only to the software design expected scenarios.

Sprint Planning is then followed by the *Sprint Implementation*, which represents the research practical part beginning. During that phase, design developers have to experiment with their ideas in order to understand all possible consequences. The effects of each design choice have to be clear in order to make effective decisions in the future.

Another fundamental point of Sprint Implementation regards traceability. Although the team works following an adaptive and flexible process, it must be capable to produce a complete trace of the software development process. Indeed, this constitutes an important step to describe to an independent assessor, how the software was constructed. Thus, through the implementation, every produced artifact must be linked to its relative Product Backlog story.

Next to the Sprint Implementation, there is the *Acceptance Testing and Software Verification*: test developers check whether all the selected user stories are correctly implemented and software behaviors as expected. Here, other techniques, such as static analysis and formal methods, may be employed by testers if they retain them useful. Even more, for Safety Integrity Level (SIL) 3 & 4 software systems, combining testing with one of these techniques is strictly required.

At the end of Software Verification, the development of the Sprint Backlog will be completed. Nevertheless, for safety-critical software research projects, there are other fundamental needed steps.

Firstly, although stories added to the Sprint Goal have been tracked and verified, there is no confirmation that they did not adversely impact the already available features and covered hazards. Thus, a subsequent *Regression Testing* step becomes essential in order to preserve the already achieved technical quality. That activity, differently from the expectations, is straightforward to realize in the context of an iterative development process. Indeed, already planned tests may be reused without the need of producing new code.

Secondly, integration developers have to check for software behavior on the target hardware. Typically, such as for the railway domain, the product has to be distributed on custom boards and operative systems. It is not rare that the final target is an embedded industrial architecture with a limited set of computational resources. In these conditions, integrators have the crucial purpose of increasing the team trust about technical problems, that could arise when the software is released in its real environment. Therefore, there is the need for a *Hardware Integration* phase.

Thirdly, is important to provide a *Traceability* phase during which all the development team has to contribute to update the software specification documents with the newly discovered observations. Indeed, the research purpose is not to produce a vendible product, but guidelines to construct it. In that way, documents could be employed multiple times to create an engineered version of the final product and prove its functional safety. By default, S4S adopts the set of documents described in the CENELEC standards for railway signaling. Anyway, needed documents can be adapted depending on the developed product.

Finally, there is the last essential phase of *Quality Assurance*, which implements the Continuous Verification & Validation concept described in the literature. Letting an independent group of *Verifiers*, *Validator*, and *Assessors*, to check the produced increment against the software requirements specification, and the applicable standards allows identifying and rapidly correct any critical violation. Ideally, the output of each Sprint could be potentially released to the final user. However, the reality is quite different. In most cases,

research groups are small, and SQA experts, if available, can review the work on a timeline of months, not weeks. Therefore, in S4S this phase could be also planned after a group of Sprints, and possibly exploiting automatic SQA tools and external experts, if any.

After the Quality Assurance phase ends, the Safe-Sprint terminates with the Scrum known activities of *Review* and *Retrospective*.

3.4.3 Documentation management

In this subsection, we want to explain more deeply how documentation management works in S4S.

Documentation, referring to this term to formal evidence that a standard requires for software assessment, is a description of both the development life-cycle and the implemented product. An assessor uses this evidence to inspect if the software released by a company could be considered safe for its intended application (Comité européen de normalisation en électronique et en électrotechnique, 2011).

Documentation management, instead, refers to the process an organization employs to manage that evidence.

One possible way of doing that is to treat documentation as the main driver of the software process. Waterfall-based development processes adopt this approach, linking subsequent phases with several requirements or design specifications. The development process has, in that case, a linear organization, with all its main steps organized in a strict sequence (Sommerville, 2015). Here, documents become information drivers between adjacent process steps.

However, in the context of an innovation process, we will not have a linear evolution since requirements have to be progressively refined. As a consequence, changes in one process phase require reworking all related documents (Sommerville, 2015; Boehm & Turner, 2003).

In S4S, instead, the Scrum team sees documentation only as an output. Proof-of-conformance evidence requested by an assessor is, in that case, the result of a complex research activity, which has conducted the team to the requirements and architecture that software must have for its intended application. Product documentation and software process activity descriptions are refined and updated during each Safe-Sprint, as described in the previous subsection.

Managing documentation as output has three main advantages. The first one is that it can reduce reworking since specifications are updated when engineers have validated their design solutions. No effort has to be spent on documentation of untestable and unfeasible requirements.

The second advantage regards a reduced complexity in writing documents. In particular, during a single iteration, the development team has not to specify and trace the entire product but only its new or modified parts.

Finally, the third advantage is that some documents can be automatically extracted from software models or internal code documentation. For instance, with tools such as *Doxygen*[2], one can produce a Software Component Design Specification (SCDS) as requested by the CENELEC EN50128.

Nevertheless, there are also some other things to consider.

[2] https://www.doxygen.nl/index.html

At first, in order to not reduce the team speed during the process, it is essential to clarify the purpose and the content of each specification. Defining templates for each type of document can help the team to focus only on required information.

Secondly, for each documentation management tool being used, a set of guidelines and policies for developers have to be defined. Indeed, each tool must be effectively configured and used to produce high-quality results.

Therefore, a clear set of templates, guidelines and policies have to be identified in order to produce good documentation.

3.5 Metrics

In this subsection, we want to analyze and propose some metrics that readers can use to evaluate and improve an application of S4S.

In particular, we identified three categories of metrics that will assist practitioners to evaluate process efficiency, safety and traceability.

Indeed, the primary purpose of S4S is to help researchers make safe, transparent, and sustainable decisions which is a fundamental requirement for safety-critical research projects.

In the next paragraph, we describe in-depth categories and the importance of related metrics.

3.5.1 Efficiency metrics

The efficiency of the software development process, i.e. the sustainable use of available resources (including humans and time), could be determined by different factors, which depend on the process structure and organization. In the case of S4S, the main concerns for efficiency are the cost of V&V activities requested by regulatory agencies and the impact of requirements or design changes that could arise during a research project.

V&V cost includes time, human resources, and tools needed to verify and validate the output of each iteration. We considered it since its related tasks constitute the most prominent part of the Safe-Sprint. Thus, if the V&V cost is too high and not sustainable, developers cannot achieve an economical and efficient implementation of S4S.

Requirements or design changes cost is, instead, given by the reworking, i.e. the cost to revise all the documents and the code already developed. The importance of this parameter comes from the conception of S4S. In fact, we introduced the methodology to help researchers with complex problems, where the solution is not known a priori. Therefore, changing something has to be light and efficient.

Given the considerations above, we defined the two following metrics:

- the *percentage of time spent for software V&V*, since it determines the final cost of human resources and adopted tools;
- the *quantity of resources to revise in case of changes* (to requirements or design), which is proportional to the reworking cost.

3.5.2 Safety metrics

We consider "safe" a software development process that provides instruments for developers to measure and control the impact of their decisions. The importance of process safety

comes from the fact that fixing software bugs too late could cause different problems, such as budget overrun, user dissatisfaction, and in the worst case, project failure.

About S4S, we structured the Safe-Sprint in order to give developers enough space to validate their solutions. In particular, we identified continuous testing, regression testing and the periodic application of V&V analysis as the primary means to control software technical quality.

From these considerations, we selected the subsequent metrics:

- the *number of discovered issues* for each software version, since the knowledge of bugs, is essential to determine the product quality;
- the *testing code coverage* data, since the team must have a complete vision of software issues.

3.5.3 Transparency metrics

A development process is transparent if developers can trace and show how requirements were implemented and verified. In critical fields, transparency is a fundamental property since, for certification purposes, products have to be assessed by an independent agency.

From a practical point of view, we can implement transparency by documenting the process with pieces of information and documents that explain how we refined software requirements during each step. Moreover, nowadays, documents can be stored and managed using requirements management tools such as Rational IBM DOORS[3].

Considering S4S, from one side, we worked to make the production of documentation required by CENELEC certification bodies sustainable. On the other side, we tried to make the overall process transparent and accessible to external people.

From the above considerations, we identified the following set of metrics to evaluate transparency:

- *process coverage*, since an external certification body has to understand all the mechanisms behind our development process;
- *the adoption of requirements management tools or platforms* to view and update development process documentation efficiently;
- *the adherence between the produced documents and the applicable standards*, since standards constitute the main base for certification bodies.

4 A case study

In order to validate the core principles and ideas behind the presented methodology, we had the chance to adopt it within an industrial research project founded by Rete Ferroviaria Italiana SpA[4]. The project regards the implementation of a message-oriented Middleware to support safe and reliable communications among the nodes of a railway signalling system.

[3] https://www.ibm.com/docs/en/ermd/9.5?topic=doors-overview-rational

[4] "Rete Ferroviaria Italiana SpA" is part of "Ferrovie dello Stato Italiane" group. The company is responsible for managing the Italian railway infrastructure.

In terms of complexity, the software exhibits different challenges, which could be summarized in:

- supporting all the pre-existent RFI hardware platforms and communication protocols;
- assuring that a communication fault could not lead to a catastrophic failure;
- fulfilling the real-time communication constraints and performance requirements of the RFI signalling infrastructure;
- satisfying the current European and National standards for railway signalling systems;

Given the complexity of the project, it represented an ideal case study to validate the proposed S4S agile process.

In the next sections, we illustrate the details about the design of the experiment, including the development team and the initialization of S4S. Then, we discuss the analysis conducted in retrospective meetings based on metrics described in Subsec. 3.5.

4.1 Design of experiment for S4S application

4.1.1 Scrum team composition

As described in Subsec. 3.3, S4S includes professional figures related to software development, verification and validation activities. In particular, the scrum team, which is responsible to build high-quality software, must include people with knowledge in requirements management, software design, software programming, and testing. Whereas the verifier, validator and assessor figures, who are external personnel from the scrum team, have to supervise the correctness of testing activities, requirements validity, and project standard-compliance.

In the context of this case study, the scrum team was composed by young researchers from the University, while RFI SpA provided the necessary support for V&V activities, with a team with strong experience in software safety validation and assessment tasks.

In particular, regarding the other scrum roles, the product owner was an embedded system engineer who has participated in and directed other research projects related to critical environments in the past. Tester and integrator roles, instead, were covered by a single researcher with experience in functional and white-box testing approaches. The scrum master role was assigned to a software engineer, who had the necessary knowledge on agile software development and the Scrum framework. Then, two junior developers were responsible for software design and development. In particular, since the group was made by experienced developers and novices, we have also decided to employ the latter only for software development tasks. In that way, senior figures were able to prioritize and check the features produced by junior programmers.

4.1.2 S4S initialization

The application of the S4S development process to the research project required some preliminary steps, which had the purpose of identifying:

- the starting set of software features in the form of Product Backlog user and safety stories;
- the time estimate for each backlog item needed for implementation and testing;

- the verification and validation activities required for checking the output of each Safe-Sprint;
- the set of technologies and tools to support the implementation of the Safe-Sprint workflow described in Subsec. 3.4.2.

For the creation of the Product Backlog, we studied the documentation provided by RFI and organized preliminary workshops for requirements analysis. The output consisted of 126 backlog items, which reflected some technical project difficulties due to the presence of: (i) SIL 4 safety functions; (ii) constrained embedded targets, and (iii) many different target platforms.

Regarding the identification of V&V activities to check the quality of the produced code, as strongly recommended by the railway standard Comité européen de normalisation en électronique et en électrotechnique (2011), we selected :

- unit and integration testing to verify the functional behaviour of new features;
- regression testing to assure the behaviour of pre-existent software components;
- static analysis to enforce software compliance with RFI coding standard.

Considering the selection of tools to support the Safe-Sprint workflow, we employed a set of fundamental technologies to build a transparent and efficient development process.

The first type of technology we want to mention is collaboration platforms. Collaboration platforms, such as GitLab[5], Atlassian[6] and Microsoft Azure DevOps[7], represent a comprehensive environment where agile teams can plan, inspect, and adapt their work, basing their decision on visible results. Regarding critical software, collaboration platforms can help agile teams to build a traceable and open development process, where an Assessor can improve his understanding of how the team works.

We decided to manage the entire development process using the GitLab open-source platform. Indeed, this enabled us to:

1. represent and share the items of our Product Backlog using *Issue Lists*[8];
2. study the weight and priority of each issue to optimize planning;
3. plan Safe-Sprints using *Milestones*[9];
4. track the status of Safe-Sprints with *Issue boards*[10];
5. direct link our codebase and relative changes to backlog issues.

Thus, the adoption of Gitlab allowed us to track and share all the main steps of the development process.

The second type of technology was testing and debugging tools, where there is the need to automate test runs during a Safe-Sprint. Indeed, although the continuous execution of test cases could improve the confidence of developers in software quality, testing remains a very time-consuming activity (Paige et al., 2008).

[5] https://about.gitlab.com/
[6] https://www.atlassian.com
[7] https://azure.microsoft.com/en-us/services/devops
[8] https://docs.gitlab.com/ee/user/project/issues/
[9] https://docs.gitlab.com/ee/user/project/milestones/
[10] https://docs.gitlab.com/ee/user/project/issue_board.html

For this task, sector standards require the adoption of non-invasive testing and debugging tools, since the tester cannot introduce changes in the software source code (Comité européen de normalisation en électronique et en électrotechnique, 2011). Therefore, we selected the *Lauterbach Trace32*[11] commercial product, which enables developers to:

- debug and trace program execution without any modification of source code;
- inspecting the current program state at different levels, including internal hardware and operative system (if present) structures;
- profiling the task response time for Worst-Case Execution Time (WCET) analysis;
- analyze software on different industrial hardware platforms and operative systems;
- demonstrate the capability of the testing tool to the Assessor.

The use of a certified instrument supported our group to improve verification efficiency while maintaining compliance with railway signalling standards.

Finally, the last type of technology, which we consider essential to mention, is static code-checkers. Code-checkers can be used to enforce software compliance against a coding standard. The enforcement of a coding standard is very useful, since it can reduce the number of software bugs by promoting the use of programming best practices.

For our project, RFI provided our group with the possibility to use the MISRA-C code checker of *MATLAB Polyspace*[12], which completed our set of essential tools.

4.2 Case study results

4.2.1 Analysis of process efficiency

Considering the percentage of time spent for software V&V, we found that Safe-Sprints of 4 weeks were essential to managing the required verification and validation activities. In particular, we organized a single Safe-Sprint as the following:

1. in the first week, we defined and implemented the Safe-Sprint backlog;
2. in the second week we worked on unit, integration, and regression testing, and checked coding-standard compliance of code;
3. in the third week we updated software requirements and architecture with all the captured observations;
4. during the last week, we released the new version to RFI experts to revise testing plans and analyze product safety;
5. finally, at the end of the Safe-Sprint, we reviewed all the done work with RFI project managers, concluding the iteration with a Sprint Retrospective.

This organization presents the 75% of time spent for V&V, which means that its related cost was at least three times the development one. We consider this result as a good starting point given that:

- the project is a safety-critical software with a high level of risk;
- the time to develop and certify a safety-critical product in RFI is in the order of years.

[11] https://www.lauterbach.com/frames.html?home.html
[12] https://it.mathworks.com/products/polyspace.html

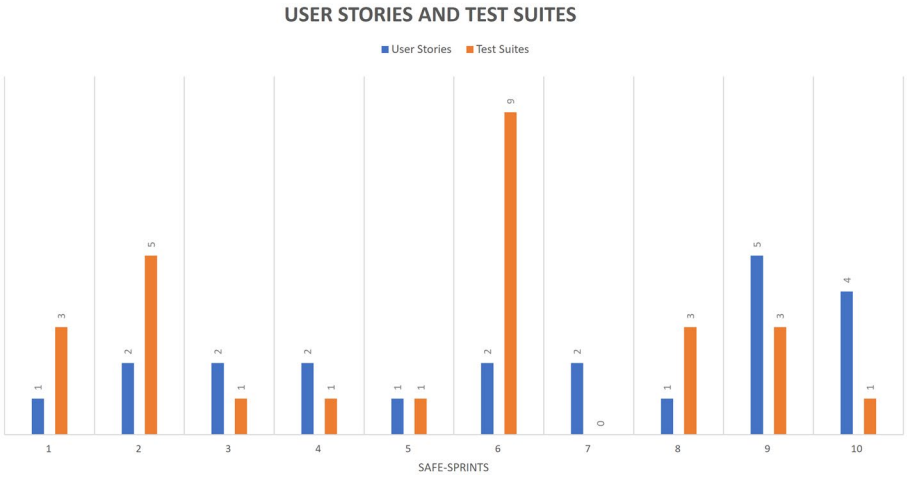

Fig. 4 User Stories and Test Suites for each Safe-Sprint

Furthermore, since we have not yet automated tasks such as test suites generation and integration testing, there is also space for optimization.

About the reworking cost, i.e. the quantity of items that the team has to revise in case of requirements or design changes, we experienced with the case study different situations. Summarizing them, it was possible that:

- the review of a Safe-Sprint did not meet user expectations;
- integration with other subsystems shown incompatibilities or usability issues;

In the first case, the team had to revise the implementation and the documents produced in the last Safe-Sprint. Thus, the reworking cost was related to the time allocated for single sprints as team productivity. For our project, four weeks represented a good thread-off between the need to produce a significant increment while limiting reworking.

In the second case, integration activities exhibited an unpredictable reworking cost. As we said in the case study introduction, the selected project has to work with the pre-existent hardware and software subsystems, as with other currently developed research projects. During those integration activities, we experienced different tricky and unpredictable compatibility issues. Indeed, since many teams worked at different levels and contexts, they took contrasting decisions.

In terms of reworking, the resolution of integration issues required the revision of multiple Safe-Sprints. Therefore, it was essential, during the application of S4S, to plan frequent integration with other subsystems. Otherwise, the reworking cost would have been hard to limit and predict.

4.2.2 Analysis of process safety

Concerning process safety, the increased awareness of developers is observable from the data of test suites and discovered issues reported in Figs. 4 and 5. As described, the

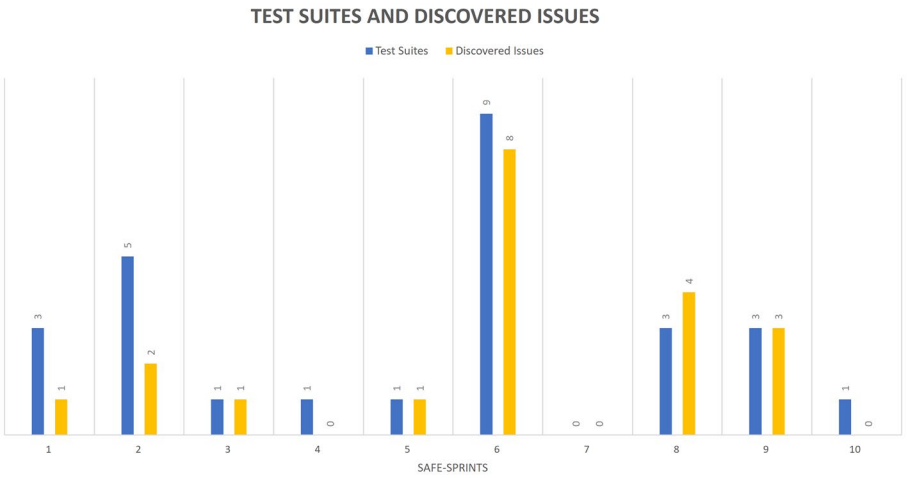

Fig. 5 Test Suites and Discovered Issues for each Safe-Sprint

team iteratively tested the code to check its technical quality (see Fig. 4), enabling the discovery of software bugs since the beginning of the process (see Fig. 5).

In terms of testing coverage, the above results were supported by an approximately complete analysis of the software code, as reported from the data of our test suites in Table 2. Therefore, developers had at least some evidence of the behaviour of each software part.

Concerning regression testing, its application helped the group, during the project, also to consolidate the work of the previous Safe-Sprints. For each iteration, the team executed test suites of software modules impacted by new modifications, gaining a much clearer vision of the effect of single changes. In our project, we experienced this phenomenon in the Safe-Sprint 8, where a software modification made for a single target platform generated an incompatibility with other architectures.

4.2.3 Analysis of process transparency

In terms of process transparency, we analyzed adherence of process documentation against railway standards, process coverage, and traceability cost-effectiveness.

About the adherence between the standards and development process documentation, we opted for full adoption of the formal set presented in the CENELEC EN50128. Indeed,

Table 2 Code coverage data extracted with *Lauterbach Trace32*

Software Components	Statement	Decision	Condition
COMP.A	100.0%	100.0%	100.0%
COMP.B	92.105%	92.105%	94.736%
COMP.C	90.0%	90.0%	100.0%
COMP.D	85.294%	85.294%	90.625%
COMP.E	92.875%	92.875%	100.0%
COMP.F	100.0%	100.0%	100.0%

although the proposed specifications refer to a traditional waterfall process, they can also be employed to describe the requirements and design of every software product.

The main advantage of this is that the certification body is not required to change its workflow to inspect the quality of produced software. Thus, as expected, the product is accompanied by requirements, architecture, and design specifications, as V&V analysis reports. However, since we used an iterative software lifecycle, we had to integrate the proposed documents with evidence that could describe, as well, the quality of the S4S process.

In order to do that, we thought that a complete audit of the executed Safe-Sprints could be essential to show the Assessor how each function was verified and validated. In particular, we considered the Safe-Sprints audit composed by:

1. The initial version of the product backlog;
2. The composition and competencies of the Scrum Team;
3. The list of all the adopted technologies with related manuals and licenses;
4. The following information for each executed Safe-Sprint:

    (a) The Safe-Sprint backlog;
    (b) The changes to software source code;
    (c) The acceptance tests used to check sprint backlog items;
    (d) The integration tests used to check software behavior on specified targets;
    (e) The software modules involved in regression testing;
    (f) The report of each testing activity;
    (g) The achieved compliance to coding standard;
    (h) The results of the V&V analysis of the Quality Assurance Team;
    (i) The Sprint Review and Retrospective results;
    (j) The modification made to the product backlog;

As the reader can notice, the purpose is to describe in depth what the team did to guarantee, measure and control the technical quality of the product.

Therefore, with S4S we achieved a strict relationship with applicable standards, but we had to introduce additional evidence to show its iteratively quality assurance process.

Another remarkable aspect, strictly related to the documents and the Safe-Sprints audit, is that they constitute an open and accessible trace of all the done activities. Thus, due to the possibility to show in-depth the refinement and validation of software requirements, we achieved, as well, a high-level process coverage.

Considering the adoption of tools to manage the above documents and audits, we needed instruments to improve process cost-effectiveness.

In particular, regarding the traceability of the S4S agile process, all the required data were captured by the GitLab collaborative platform. As we described in Fig. 3 and in Subsec. 4.1.2 the team had a comprehensive platform to trace the implementation of each backlog story. However, since the Safe-Sprints audit is a custom output, we are currently working on custom tools to extract and compose it automatically.

Then, about the software requirements, architecture, and design specifications, as other V&V reports requested by the standard, we are evaluating the possibility to use Rational IBM DOORS. Indeed, we found it very difficult to work outside a requirements management tool, given the large number and complexity of software requirements. Therefore, we are currently collaborating with RFI to address the problem of identifying a proper Integrated Development Environment (IDE) and migrating documentation of pre-existent products.

5 Conclusion

In this paper, we proposed an extension of the Scrum agile methodology, namely S4S, suitable to guide and help the design and development of software components in safety-critical domains, in particular, in the railway domain. We discussed S4S in full details, including its context, principles, roles and workflow. Furthermore, in order to evaluate the methodology, we report a case-study on a real, highly complex safety-critical research product with changeable requirements, which represents a typical situation for research groups.

The reported case study highlighted that S4S (i) enables iterative and evolutive development of safety-critical software, even if architecture and/or requirements need to be refined, (ii) allows documentation to be produced – and kept updated – as an output of the entire process, and (iii) makes the entire process much more safe and reactive w.r.t human errors.

Therefore, from these conclusions, we could state that the agile mindset remains effective in a critical context if it embraces all its values in terms of quality. Nevertheless, this paper only constitutes a starting point: we intend to apply S4S in other different critical research projects, even those involving third-party and/or legacy software components, in order to add new tools and techniques that would increase its current efficiency and safety.

Funding Open access funding provided by Università degli Studi di Napoli Federico II within the CRUI-CARE Agreement.

Declarations

Conflicts of interest This work was partially founded by Rete Ferroviaria Italiana SpA and by the University of Naples Federico II (Finanziamento delle Ricerca di Ateneo 2020), within the "REYNA" project.

References

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for agile software development.

Benedicenti, L., Cotugno, F., Ciancarini, P., Messina, A., Pedrycz, W., Sillitti, A., & Succi, G. (2016). Applying scrum to the army: a case study. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 725–727). IEEE.

Boehm, B., & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional.

Casola, V., De Benedictis, A., Rak, M., & Villano, U. (2020). A novel security-by-design methodology: Modeling and assessing security by slas with a quantitative approach. *Journal of Systems and Software, 163,* 110537.

Cawley, O., Wang, X., & Richardson, I. (2010). Lean/agile software development methodologies in regulated environments-state of the art. *International Conference on Lean Enterprise Software and Systems* (pp. 31–36). Heidelberg: Springer.

Cockburn, A. (2006). *Agile software development: the cooperative game*. Pearson Education.

Comité européen de normalisation en électronique et en électrotechnique, C. (1999). *Railway Applications The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*. (Standard) Comité européen de normalisation en électronique et en électrotechnique (CENELEC).

Comité européen de normalisation en électronique et en électrotechnique, C. (2003). *Railway application – Communications, signaling and processing systems – Safety related electronic systems for signaling*. (Standard) Comité européen de normalisation en électronique et en électrotechnique (CENELEC).

Comité européen de normalisation en électronique et en électrotechnique, C. (2011). *Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems*. (Standard) Comité européen de normalisation en électronique et en électrotechnique (CENELEC).

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

Fitzgerald, B., Stol, K., O'Sullivan, R., & O'Brien, D. (2013). Scaling agile methods to regulated environments: An industry case study. In *2013 35th International Conference on Software Engineering (ICSE)* (pp. 863–872). IEEE.

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*, 945–962.

Ge, X., Paige, R. F., & McDermid, J. A. (2010). An iterative approach for development of safety-critical software and safety arguments. In *2010 Agile Conference* (pp. 35–43). IEEE.

Górski, J., & Łukasiewicz, K. (2013). Towards agile development of critical software. *International Workshop on Software Engineering for Resilient Systems* (pp. 48–55). Heidelberg: Springer.

Grenning, J. (2001). Launching extreme programming at a process-intensive company. *IEEE Software, 18,* 27–33.

Hajou, A., Batenburg, R., & Jansen, S. (2014). How the pharmaceutical industry and agile software development methods conflict: A systematic literature review. In *2014 14th International Conference on Computational Science and Its Applications* (pp. 40–48). IEEE.

Hanssen, G., Stålhane, T., & Myklebust, T. (2018). *SafeScrum® - Agile Development of Safety-Critical Software*. New York: Springer.

International Organization for Standardization, I. (2011). *Road vehicles - Functional safety*. (Standard) International Organization for Standardization (ISO).

International Electrotechnical Commission, I. (2010). *Functional safety of electrical/electronic/programmable electronic safety-related systems*. (Standard) International Electrotechnical Commission (IEC).

Jonsson, H., Larsson, S., & Punnekkat, S. (2012). Agile practices in regulated railway software development. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops* (pp. 355–360). IEEE.

McCaffery, F., Trektere, K., & Ozcan-Top, O. (2016). Agile–is it suitable for medical device software development? In *International Conference on Software Process Improvement and Capability Determination* (pp. 417–422). Springer.

McHugh, M., McCaffery, F., & Casey, V. (2012). Barriers to adopting agile practices when developing medical device software. *International Conference on Software Process Improvement and Capability Determination* (pp. 141–147). Heidelberg: Springer.

Messina, A., Fiore, F., Ruggiero, M., Ciancarini, P., & Russo, D. (2016). A new agile paradigm for mission-critical software development. *CrossTalk, 29,* 25–30.

Misra, S. C., 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,* 451–474.

Myklebust, T., Stålhane, T., & Lyngby, N. (2015). Application of an agile development process for en50128/railway conformant software. Paper presented at the Esrel Conference of Safety and Reliability of Complex Engineered Systems, Zurich, 7–9 September 2015.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM, 48,* 72–78.

Notander, J. P., 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* (pp. 1137–1144). New York, NY, USA: Association for Computing Machinery.

Paige, R. F., Charalambous, R., Ge, X., & Brooke, P. J. (2008). Towards agile engineering of high-integrity systems. *International Conference on Computer Safety, Reliability, and Security* (pp. 30–43). Heidelberg: Springer.

Rubin, K. S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Boston: Addison-Wesley Professional.

Schwaber, K., & Sutherland, J. (2020). The scrum guide?™.

Smith, J., Bradbury, J., Hayes, W., & Deadrick, W. (2019). Agile approach to assuring the safety-critical embedded software for nasa's orion spacecraft. In *2019 IEEE Aerospace Conference* (pp. 1–10). IEEE.

Stålhane, T., Myklebust, T., & Geir, H. (2012). The application of safe scrum to iec 61508 certifiable software. In *11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference 2012, 25-29 June 2012, Helsinki, Finland* (pp. 6052 – 6061). Curran, Associates, Inc. volume 8.

Sommerville, I. (2015). *Software processes*. Pearson.

VanderLeest, S. H., & Buter, A. (2009). Escape the waterfall: Agile for aerospace. In *2009 IEEE/AIAA 28th Digital Avionics Systems Conference* (pp. 6–D). IEEE.

Vogel, D. (2006). Agile methods: Most are not ready for prime time in medical device software design and development. *DesignFax Online*, *2006*.

Wolff, S. (2012). Scrum goes formal: Agile methods for safety-critical systems. In *2012 first international workshop on formal methods in software engineering: Rigorous and agile approaches (formsera)* (pp. 23–29). IEEE.