2018 IEEE 26th International Requirements Engineering Conference

Discovering, Analyzing, and Managing Safety Stories in Agile Projects

Jane Cleland-Huang and Michael Vierhauser
Department of Computer Science and Engineering
University of Notre Dame
Indiana, USA
JaneClelandHuang@nd.edu, mvierhau@nd.edu

***Abstract*—Traditionally, safety-critical projects have been developed using the waterfall process. However, this makes it costly and challenging to incrementally introduce new features and to certify the modified product for use. As a result, there has been increasing interest in adopting agile development paradigms within the safety-critical domain. This in turn introduces numerous challenges. In this paper we address the specific problems of discovering, analyzing, specifying, and managing safety requirements within the agile Scrum process. We propose *SafetyScrum*, a methodology that augments the Scrum lifecycle with incrementally applied safety-related activities and introduces the notion of "safety debt" for incrementally tracking the current safety status of a project. We demonstrate the viability of *SafetyScrum* for managing safety stories in an agile development environment by applying it to a project in which our existing Unmanned Aerial Vehicle system is enhanced to support a River-Rescue scenario.**

I. Introduction

Systems operating in safety-critical domains, where failures can cause harm or injury, must not only deliver prescribed functionality, but must do so in a way that ensures that the system is safe and secure for its intended use [28]. To this end, safety-critical systems must meet stringent guidelines in order to receive approval or certification [53, 19, 7, 14]. The strict requirements of the certification process as well as constraints introduced by the rigid timelines imposed by hardware components have led many organizations to follow a traditional waterfall approach – often resulting in the phenomenon referred to as the *big freeze* [42]. The significant cost and effort of changing and then recertifying a product makes it difficult to introduce change, thus hampering the ability to provide new features or to respond to customer needs.

Agile techniques have traditionally been deemed unsuited to safety-critical development [9]; however, recently the idea of leveraging agility has gained considerable traction. For example, the European Open-DO initiative [42] is exploring techniques for integrating agility into the safety-critical development process, and there are numerous accounts reporting its experimental and effective deployment [34, 31]. Doss and Kelly [18] reported results from a recent practitioner survey with a total of 31 participants, 87% of whom had experience in safety-critical systems development, and 77% with practical experience using a broad range of agile methods. Their survey produced several insights of particular interest to the requirements process. Respondents strongly supported the notion that eliciting safety requirements, performing hazard analysis, and developing safety assurance cases must be performed iteratively, with 50% reporting that safety problems were not always identified early in the lifecycle during the upfront hazard analysis. In other words, they acknowledged the need for a more incremental development process.

On the other hand, applying agile processes in safety-critical projects introduces multiple challenges – each of which must be carefully explored in order to develop appropriate solutions and practices. In this paper we provide concrete examples derived from our experiences in using an agile process to develop the Dronology system for controlling Unmanned Aerial Vehicles (UAVs) and we describe the agile safety process we adopted as a result of those experiences. The characteristics of our project, including its initially unknown requirements, a steep technical learning curve associated with entering a new domain, and team members' prior experience with agile methods, indicated a good fit for applying an agile, Scrum-based approach [60]. However, in early phases of our project, we realized that the project's safety concerns were non-trivial and could not be adequately addressed without carefully augmenting the Scrum process.

Initially, in early phases of our project, we addressed safety issues through conducting a series of brainstorming sessions in which we identified hazards and their contributing faults, and then proposed safety requirements and design constraints that would prevent or mitigate the occurrence of the hazard. However, we found that identifying *all* hazards and their contributing failures at the start of the project was particularly challenging given the emergent nature of the UAV domain, including its novel end-user applications and rapidly changing technologies. Many new hazards and faults were discovered incrementally as we conducted field tests with the UAV hardware, met with stakeholders to explore their emerging requirements, and brainstormed design solutions. Our early observations aligned closely with those made by participants in Doss' survey [18] and highlighted the potentially competing goals of agile processes versus safety-critical development.

Agile development practices are founded upon four core principles laid out in the agile manifesto [6]. Two of these principles of "responding to change over following a plan" and delivering "working software over comprehensive documentation" are particularly challenging to achieve in safety-critical

2332-6441/18/$31.00 ©2018 IEEE
DOI 10.1109/RE.2018.00034

RE 2018, Banff, Canada
Research Paper

projects in which the delivered software must be demonstrably and arguably safe for use [42, 54, 21]. To address these challenges we identified three philosophical values that were fundamental to the process decisions we made. These can be summarized as (1) imbuing safety thinking into the entire development process, (2) embracing the incremental discovery and analysis of safety hazards, (3) maintaining continual awareness of the safety status,so that the enhancements and modifications needed to attain eventual safety, are well understood and fully achievable.

In the remainder of the paper we describe the process we developed for discovering, analyzing, and specifying safety requirements within a Scrum project environment. We start by discussing related work in Section II. Sections III and IV then introduce the Dronology project and provide a detailed description of the process with motivating examples. In Section V we describe the application of the approach across three sprints in which Dronology was extended to support a physical river rescue application. Finally in Sections VI and VII we discuss threats to validity and draw conclusions.

II. Related Work

Several authors have discussed ways in which to integrate agile methods with safety-critical practices [46, 40, 55, 40, 63, 65], however, most prior work represents broad surveys of practitioners, or is theoretical in nature without providing concrete end-to-end process descriptions. One exception, is the work by Fitzgerald *et al.* [21] who conducted a study in a large automotive company to investigate the use of agile development processes in regulated environments. They proposed R-Scrum, which emphasizes traceability of artifacts and continuous compliance throughout the development process. They augmented Scrum with activities such as "sprint hardening" to address safety-critical concerns, "continuous compliance" (i.e., audits and checkpoints at the end of each sprint), and "living traceability" (i.e., automated, and tool supported traceability between artifacts). We implemented aspects of these elements in our approach, but focused attention on tasks related to discovering, analyzing, and managing safety stories.

Stalhane *et al.* [55] presented "SafeScrum" complementing the traditional Scrum process. While our approach and SafeScrum both explicitly differentiate between critical requirements (Safety Stories) and other requirements (System Stories), SafeScrum separates concerns, by isolating everything from the actual Scrum process that is not part of the software development process itself. In contrast, *SafetyScrum* argues for a tighter integration of traditional and safety-related activities.

Wang *et al.* [65] proposed the S-Scrum development process and evaluated it on student projects. S-Scrum includes safety analysis techniques, more precisely "System Theoretic Process Analysis" (STPA), to support and guide the design of a safety-critical system. In contrast, to S-Scrum in which STPA is integrated in the Scrum process and used as part of a sprint, our approach focuses on the Scrum process itself by tailoring and adapting it to support safety analysis.

In the domain of aerospace systems, VanderLeest and Buter [63] analyzed compatibility of agile practices to the DO-178B standard. While this study provides interesting insights into the different practices and their suitability in a safety-critical domain, they did not provide a process incorporating the different practices. Stephenson *et al.* [56] proposed an "agile health model", for use in developing a safety-critical system. The model contains safety-related information and assumptions, provides documentation, and guides incremental development. Similar approaches exist for the domain of medical devices [52, 46, 36] and high-integrity software [43]. In contrast to these domain-specific approaches, we aimed to create a more generic approach that augments the traditional Scrum process with safety-related activities.

Other authors have conducted literature surveys on agile methods for safety-critical systems [41], or augmented specific practices. For example, Stalhane and Myklebust [54] presented a process for performing hazard analysis based on user stories, while Gorski and Lukasiewicz [25] proposed the use of assurance argument patterns that combined agile and established practices to reduce risks in safety-critical projects.

III. The Dronology Project

We developed our process as a result of lessons we learned whilst applying an agile approach to the development of the Dronology project for using small UAVs in emergency response applications. Dronology includes a flight manager responsible for scheduling flight routes for multiple UAVs, basic collision avoidance, an internal simulator, a flight activity logger, several UIs for planning routes, monitoring UAVs in real time, and registering UAVs, and finally a Ground Control Station (GCS) middleware as well as a concrete implementation for communicating and controlling real, physical UAVs. The Dronology GCS, interfaces with ArduPilot-based UAVs [2] and has been flight-tested with five different physical UAV models. It also interfaces with the ArduPilot Software-in-the-loop (SITL) simulator to enable high-fidelity simulations.

The project is considered medium-sized [1] based on a current budget of (>$200K), team size (5-12 people), duration (4 year plan), and impact (moderate). The development team has so far included five software engineers including the two authors – both of whom have industrial development experience (5 years and 4 years respectively), one professional developer with expertise in UAV flight control, and one expert in Human-Computer Interaction. In addition, an Electrical Engineer has been responsible for mechanical and electrical components and communication networks, and a total of 12 graduate and undergraduates were employed as summer interns in 2017 and 2018. Three team members are certified remote pilots. External stakeholders, including members of the local city administration and its firefighters participated in the requirements discovery process and actively engaged in the public river-rescue demo, conducted in April 2018, that involved UAVs operated using Dronology, firefighters, and a rescue boat.

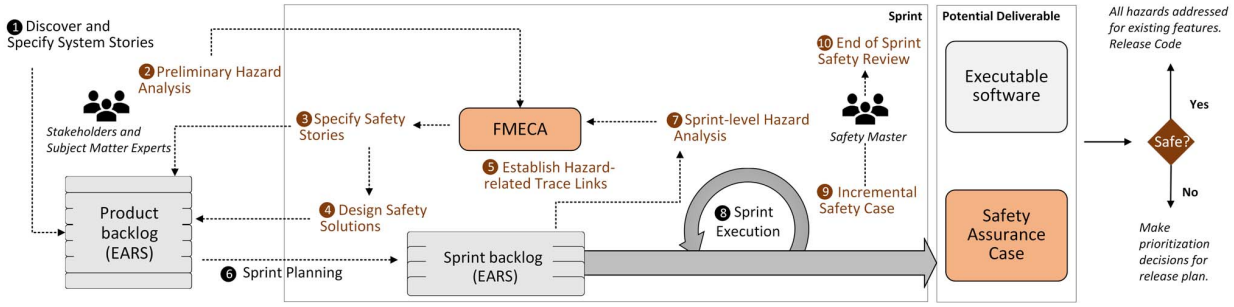

Fig. 1: *SafetyScrum* augments the traditional SCRUM process by emphasizing hazard analysis and safety assurance. Safety-related activities and artifacts are marked in red.

IV. Process

Our methodology, which we refer to as *SafetyScrum*, is summarized in Fig. 1 and includes ten specific activities. We briefly describe each of these activities, provide illustrative examples from the Dronology project, and discuss implications upon the agile process. In many cases, the activity represents an extension to the typical Scrum process and therefore adds overhead to the project. However, each additional task contributes to the achievement of a safe product and is designed to fit seamlessly into the agile development process.

Following a traditional Scrum approach, three different roles are involved in the development process: the *Scrum Master* guides and supports the team during the sprints, the *Product Owner* represents a key stakeholder, and the *developers* form the remainder of the Scrum team. In the context of a safety-critical system, where thoroughly analyzing, planning, and testing safety-related requirements is of the utmost importance, we introduce the additional role of *Safety Master*. The Safety Master bears responsibility for all safety-related tasks and activities, and ultimately determines whether hazards have been sufficiently mitigated in order to deem the current product safe for release. We create this special role to ensure that safety concerns are continually evaluated and addressed and are not overshadowed by the inevitable pressures of delivery deadlines. We discuss the role and responsibilities of the safety master throughout the remainder of this section when describing the activities of our process. Each activity is then summarized and highlighted (in a gray box.)

ACTIVITY 1: Discover and Specify System Stories

The first step of the process involves specifying functional and non-functional requirements to create the context in which safety analysis will occur. In agile projects "requirements" are typically specified as user stories [5]; however, we adopted the Easy Requirements Syntax (EARS) approach [39, 11], which is more expressive than user stories for specifying CPS requirements. It provides a simple and structured template for specifying requirements for ubiquitous, event driven, state driven, optional, and unwanted behavior. An example for each type of requirement drawn from Dronology, is shown below:

Ubiquitous: The <*component name*> shall <*response*>.
System Story (SYS-1): A UAV shall maintain a minimum separation distance from other UAVs at all times.

Event Driven: When <trigger> the <system name> shall <*system response*>.
System Story (SYS-2): When a flightplan is initiated a log event shall be created that includes the name of the route, the UAV it is assigned to, and the initiation time stamp.

State Driven: While <*in a specific state*> the <*system name*> shall <*system response*>.
System Story (SYS-3): While in landing mode, the UAV shall descend vertically until it reaches the ground.

State Option: Where <*feature is included*> the <*system name*> shall <*system response*>.
System Story (SYS-4): Where onboard obstacle detection is available,the UAV shall fly around obstacles and then resume its directed route.

Unwanted Behavior: If <*optional preconditions*> <*trigger*>, then the <*system name*> shall <system response>.
System Story (SYS-5): If wind gusts exceed maximum wind flying conditions, then the UAV shall not be granted permission to take-off.

While, our proposed safety process is agnostic to the way requirements are structured, all examples throughout the remainder of this paper are illustrated using the EARS format. Further, we refer to them as *system stories*, as opposed to *user stories*; however, in every respect, apart from their formatting, these system stories are treated in the same way as more traditional user stories. They are discovered following standard elicitation practices [50, 15], specified, analyzed for trade-offs, and ultimately placed into the product backlog. While the process of system story elicitation is shown at the start of our process in Fig. 1, in practice it continues throughout the lifetime of the software system driven by stakeholder needs as well as through emerging safety and performance concerns.

Specification of system stories lays the foundation for our overall process and is therefore depicted as activity (A1).

A1: Elicit and Specify System Stories
Engage with project stakeholders to elicit and specify an initial set of system stories, representing their functional and non-functional requirements.

ACTIVITY 2: Preliminary Hazard Analysis

Once an initial set of system stories has been identified, the project team conducts an upfront hazard analysis [35], using techniques such as Software Fault Tree (FT) Analysis [57, 59] or Failure mode, effects and criticality analysis (FMECA) [48, 37] to identify an initial set of hazards, failure modes, and their associated criticality levels.

Some researchers have observed that upfront safety analysis is contradictory to the agile philosophy which favors a just-in-time approach to all analysis and design activities [64]. However, performing an initial hazard analysis provides the foundation for the remainder of the project, leads to the discovery of new safety-related stories, influences the architectural design, and imbues safety-thinking into the Scrum process from the inception of the project [8].

We provide examples of two hazards that emerged from a series of meetings with the core project team and our stakeholders.

Data Hazard (H-1): Inaccurate GPS (Global Positioning System) coordinates for UAV. *Failure Mode:* GPS provides inaccurate readings. *Effect:* Violation of minimum separation distance between two UAVs goes undetected, and UAVs collide in midair and then crash onto bystanders. *Level:* Critical

Algorithmic Hazard (H-2): Flight path routing algorithm generates incorrect waypoints. *Failure Mode:* Routing algorithm fails to take into account relative altitude of surrounding terrain. *Effect:* UAS flies into the flight path of a commercial airplane. *Level:* Catastrophic

It is worth noting that Hazard H-1 was impacted by a design decision to initially implement collision avoidance in a centralized manner, while the goal of incorporating onboard obstacle detection and avoidance was deferred to later iterations. This is typical in agile projects, and embraces the reality that hazards and their mitigations may evolve as new features are introduced.

The time and effort invested in the preliminary hazard analysis needs to be determined on a project-by-project basis according to project characteristics such as requirements volatility, criticality, difficulty of introducing changes later, and the upfront domain knowledge of the development team. In our project we invested approximately 80 person hours in conducting a preliminary hazard analysis that included 6 of the previously mentioned team members.

The preliminary hazard analysis activity represents the second activity (A2) of our *SafetyScrum* process.

A2: Perform Preliminary Hazard Analysis
Conduct a hazard analysis in early phases of the project to identify potential hazards and failure modes in order to drive activities such as architecture design, story specification, safety analysis, and rigorous testing throughout the remainder of the project.

ACTIVITY 3: Specify Safety Stories

Each identified safety hazard, that is deemed non-trivial is transformed into a safety story and placed into the product backlog. The safety story is equivalent to a safety requirement, described by Firesmith [20] as a "requirement that specifies a minimum, mandatory amount of safety ... in terms of a system-specific quality criterion and a minimum level of an associated metric." Identifying safety stories capable of mitigating or significantly reducing the risk of a hazard is aided by industry-specific standards and policies, reuse from existing systems, subject-matter expertise, and brainstorming activities. Typically, a single hazard will be addressed by multiple safety stories describing asset protection, incident detection, incident reaction, and potential system adaptation. These requirements may be both positive (i.e., what the system must do) or negative (i.e., what the system must not do) [66]. An example of a ubiquitous safety story associated with Hazard H-1 is shown below.

Safety Story (SAF-1): The GPS coordinates of each UAV must be accurate within one meter at all times.

The activity of discovering and specifying safety stories to address identified hazards is summarized in activity A3.

A3: Specify Safety Stories
Identify, analyze, and specify safety stories that, if satisfied, will prevent the hazard from occurring or reduce the impact of its occurrence. Place safety stories into the product backlog.

ACTIVITY 4: Design Safety Solutions

In his seminal book on eXtreme Programming [5], Kent Beck referred to a story as a "placeholder for a future conversation" and deliberately avoided specifying design-level "requirements" until the start of each sprint when the selected stories are broken down into implementable tasks. However, in a safety-critical project it is necessary to be more deliberate about how a safety story is implemented and to clearly specify mitigating design solutions. These design definitions allow developers, project stakeholders, and external regulators to reason about how hazards have been mitigated and to evaluate whether system safety has been achieved.

In safety-critical domains, parts of the system which impact safety must be designed explicitly and documented as design definitions. The design activity can be conducted incrementally, following the agile mantra of no "big design upfront" (BDUF) or explored early in the project [61, 16] so that developers and other project stakeholders can weigh the benefits of designing a solution that focuses on meeting current needs or on looking-ahead to accommodate future planned features [58, 67, 23]. Either way, designing a suitable solution requires a decision-making process in which candidate designs are proposed and evaluated, leading to the selection of a solution that best satisfies the safety stories within the context of other competing concerns such as functionality, cost, usability, and

performance [66]. The solution is likely to include a diverse combination of software constraints, hardware elements, and even operating procedures.

Here we provide examples of three design definitions that represent selected solutions for addressing the previously presented safety story (SAF-1) regarding GPS accuracy.

Design Definition (DD-1): When the Dronology system is deployed in an urban environment at least two independent means of UAV localization must be used.

Design Definition (DD-2): If UAV localization mechanisms provide conflicting information, the lowest reported distance between two UAVs shall serve as their current separation distance.

Design Definition (DD-3): Real-Time Kinematic (RTK) shall be deployed. (Note: This increases the guaranteed accuracy of GPS to 2 centimeters.)

The design activity is captured as our fourth activity (A4):

> **A4: Design for Safety**
> Design a solution to address each of the safety stories. Specify the solution as a set of design definitions prior to scheduling its associated safety-stories into a sprint.

ACTIVITY 5: Establish Hazard-related Trace Links

Traceability is particularly important in an agile project in which change is embraced [10, 29, 44]. Traceability provides support for following a safety-related requirement back to its initiating hazard(s) and forward to the design decisions, code, and test-cases through which the hazard is potentially realized and mitigated [26]. This level of traceability is not only essential for assessing safety, but is also prescribed by many certification standards [22, 53]. Enabling dynamic, maintainable, and cost-effective traceability allows us to analyze safety even as new features are incrementally introduced to the project [10, 13, 49].

While traceability has traditionally been perceived as a burdensome overhead that is anathemic to an agile project, current practices and technologies make it a viable agile practice [24, 10]. At the requirements level, agile tools such as Jira, enable trace links to be created in lockstep with the specification of stories and design definitions. The process can be supported by a simple recommender system which uses trace link creation and evolution algorithms to proactively suggests missing trace links [45]. At the code level, a typical version control system such as GitHub, can be used to establish links between source code and design definitions and/or requirements, simply be tagging each commit with the ID of the design definition or requirement. It is out of scope of this paper to discuss traceability algorithms and tools that aid developers in creating and maintaining trace links; however, recent research advances create viable options for automated tracing support within an agile project environment [12].

Fig. 2 provides a simple example of traceability from design definitions to a safety hazard. The left hand side depicts links

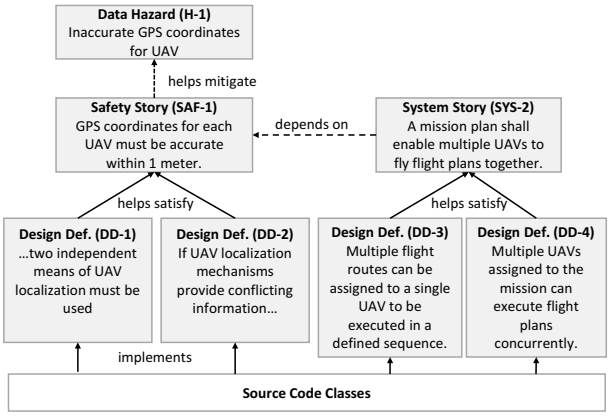

Fig. 2: A simple trace slice showing links from design definitions to safety requirements, and from safety requirements to the hazards they are designed to mitigate.

for the previously presented artifacts (H-1, SAF-1, DD-1, DD-2), while the right hand side shows a system story (SYS-2), two of its associated design definitions (DD-3, DD-4), and a dependency link from SYS-2 to SAF-1. Additional trace links are captured from source code (via tags in commit messages) [47], and (not shown in the figure) from test cases, and test case results etc; however, here we focus on the requirements-level trace links.

There are several benefits to establishing this level of traceability. First, compliance is achieved to many certification guidelines and standards; second, support is provided for evaluating the extent to which each hazard has been addressed in the design; and finally, parts of the system that are susceptible to the hazard can be identified – in this example, code implementing design definitions DD-3 and DD-4. The traceability activity is summarized as *SafetyScrum* activity five.

> **A5: Establish trace links**
> Leverage capabilities of tools commonly adopted in agile projects (e.g., Jira and Github) to incrementally construct trace links from safety stories to hazards, design definitions to safety stories, dependent system stories to safety stories, source code to design, and from acceptance tests to safety stories.

ACTIVITY 6: Sprint Planning

Sprint planning typically involves two phases – an initial phase in which the backlog is groomed to ensure that stories are prioritized, well-written, and estimated in terms of their effort, and a selection phase at the start of each sprint in which specific stories are selected for implementation according to the customer's priorities. These activities are described in numerous books on agile development [5, 38, 17]; therefore we focus this discussion on ways in which *SafetyScrum* modifies the planning process to accommodate safety concerns.

The dependencies that safety stories exhibit upon their associated design definitions should be taken into consideration during sprint planning. First, as part of the backlog grooming activity, we track the status of each safety story, to document whether its design is sufficiently complete i.e., whether sufficient and appropriate design definitions have been specified that, if implemented correctly, would satisfactorily address the safety story. Second, we schedule design definitions for implementation across specific sprints. In some cases, design definitions may need to be decomposed into sprint-sized definitions. Scheduling of a design definition could happen after the safety story is designated as "fully designed" or in a more incremental way once its core design decisions have been made. When a safety story is designated as *fully designed* and all of its associated design definitions have been implemented and tested across one or more sprints; then the safety story can be scheduled into the sprint for rigorous acceptance testing. This could happen in the same sprint that its final design definition is implemented, or could be in a subsequent sprint.

Bearing in mind the dependencies that exist between system stories, safety stories, and their associated design definitions, it makes sense to prioritize safety stories associated with currently implemented or scheduled system stories.

This is illustrated with the example in Fig. 2. Consider the case in which system story SYS-2 and its associated design definitions (DD-3, DD-4) are implemented without safety story (SAF-1), then the system, as delivered at the close of the sprint, cannot be considered safe for use as UAVs with overlapping paths could collide in midair. The system could still be released for testing the simultaneous flight of multiple UAVs, but only under additional constraints such as planning non-overlapping flight paths. This observation leads to *SafetyScrum* activity six.

A6: Safety-aware Sprint Scheduling
During the sprint planning process, track the design status of each safety requirement and mark its design status as "open" or "completed". Be aware of dependencies between system stories and safety stories and favor schedules in which safety stories and their associated design decisions are implemented as close as possible in the timeline to their dependent system stories.

ACTIVITY 7: Sprint-level Hazard Analysis

At the start of each sprint, the project team analyzes hazards associated with the scheduled stories. This is necessary for three key reasons; first, the preliminary hazard analysis may have been incomplete; second, new system stories may have been introduced after the initial preliminary hazard was performed; and finally, the scheduled stories may interact with existing features in ways that were not previously considered. An interesting example arose in our river search and rescue project when stakeholders expressed a desire to launch UAVs from a small boat used as a command center during an emergency response. As a result, we specified the following system story:

System story (SYS-5): UAVs shall be launched from a boat during river rescue.

However, on closer inspection, we realized that this introduced new hazards associated with the following pre-existing system story:

System story (SYS-6): When the return to launch (RTL) command is issued, the UAV shall return to its original launch coordinates.

First, the likelihood that the boat would change position since the UAV launch, meant that a UAV executing an RTL command would likely land in the water. Second, even if the boat were to remain at exactly the same location, the current inaccuracies in GPS localization could also cause the UAV to miss the boat or land on a person in the boat – causing potential bodily harm. Without the sprint-level hazard analysis, these new hazards may have been missed.

We further observed that launching a UAV from a boat is relatively straightforward while returning the UAV to the boat after its flight is far more complex. Additional safety stories and associated design solutions need to be devised before the the story can be realized. In this case, several combinations of options could be considered including (1) reducing the scope of the requirement to only allow launches from the boat but not landings, (2) using technology such as Real-Time Kinematic (RTK) satellite navigation to increase accuracy to 1 cm horizontally and 2 cm vertically, or (3) using additional commands e.g., voice, visual, or mobile app to provide homing mechanisms during the landing phase. We decided to support UAV launches from the boat, but to provide a safe "home" location on the river bank as the target of the RTL command. This decision defers the more difficult task of boat landings until later in the project. As a result, system story SYS-6 was modified as follows:

System Story (SYS-6'): When the return to launch (RTL) command is issued, the UAV shall return to its *home* coordinates.

Additional requirements were written to clearly specify how *home* coordinates were derived under various operating contexts. The activity of performing incremental hazard analysis prior to each sprint is defined as *SafetyScrum* activity seven.

A7: Sprint-level Hazard Analysis
Perform an indepth hazard analysis at the start of each sprint to identify new hazards, failure cases, and mitigations associated with new features and their interactions with other features.

ACTIVITY 8: Sprint Execution

During each sprint, design definitions are broken down into tasks, implemented, and tested [5]. Testing needs to be rigorous and cover multiple levels including unit tests, software integration, and systems integration [64] tests. Dedicated

"hardening" sprints, as proposed by Fitzgerald *et al.* [21] should also be scheduled.

Regardless of the rigor of upfront and pre-sprint hazard analysis, it is highly likely that additional hazards will be discovered during the testing phase. For example, in our project, we had specified the following safety story:

Safety Story (SAF-2): A human operator shall use the hand-held ground control station associated with a unique UAV to gain control from Dronology upon request.

To satisfy this requirement, we designed a solution in which Dronology cedes control if the human operator switches mode from "GUIDED" to "STABILIZE". The feature had been implemented, tested, and used for numerous flights; however, when a new operator assumed the role of human operator during a test, she positioned the throttle in the fully downward direction (which the UAV manufacturer states should be done only when the UAV is inches from the ground during landing). As a result, as soon as control was ceded, the UAV received the command to descend rapidly, resulting in it plummeting 30 meters to the ground and breaking upon impact as shown in Fig. 3. The incident was documented as a fault, and a subsequent investigation attributed the cause to incorrect throttle position due to operator error caused by insufficient training. A new hazard was documented with an associated safety story as shown below:

Data Hazard (H-3): Incorrect throttle position on hand-held GCS *Failure Mode:* Throttle switch is placed in the downward position when Dronology concedes control to the hand-held GCS. *Effect:* UAV plummets to the ground during hand-over. *Level:* Critical.

Safety Story (SAF-3): When Dronology cedes control to the handheld GCS, the UAV shall assume a stable 'hover-in-place' state.

Determining how to achieve this safety story was, and still is, a non-trivial challenge, constrained by an existing safety story that a human UAV operator must have immediate ability to control the UAV. Potential solutions include enhanced operator training, use of preflight checklists to ensure that the switches are positioned correctly prior to takeoff, ongoing checks during flight with audible warnings if switches are positioned incorrectly, and virtual interlocks to prevent erratic flight behavior immediately following handover. None of these are perfect solutions and design analysis is therefore ongoing. The need to handle emergent hazards and failures leads to *SafetyScrum* activity A8:

A8: Handle emergent Faults
When new hazards, failure modes, and safety stories emerge as a result of observed faults during testing, the faults are documented, new safety stories specified and design solutions explored. The new stories and design definitions are added to the product backlog, and dependencies are documented as trace links.

Fig. 3: A previously unknown hazard associated with the positing of a switch on the hand-held device, cause the UAV to crash during a manual take-over test.

ACTIVITY 9: Incremental Safety Case

Throughout the sprint, the team must incrementally refine the safety case (SC) – defined by Kelley [33] as a "clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context". A SC is composed from safety claims, evidence, and arguments [3, 27, 32, 62]. As reported by participants in Doss' survey [18], hazards, safety stories, and their associated mitigations tend to emerge incrementally, thereby introducing the need to inspect, and potentially update the safety case, during each sprint.

Fig. 4 depicts a small section of a SC for the Dronology-to-human handover modeled using Goal Structured Notation. The white nodes provide a hierarchical argument that a human operator can take over control from Dronology. Drawing on the previous example in which an incorrectly positioned switch caused the UAV to plummet to the ground, we update the SC with Goal $G1.3$ (shown in yellow), to reflect the decision to create a short-term mitigation based on "operator training" until a more robust solution is provided in future sprints.

The task of incrementally refining the SC involves identifying the impact of new features on the existing SC, evaluating whether any claims, arguments, or evidence are invalidated or missing, and updating the SC accordingly.

A9: Incrementally build Safety Case
Evaluate the impact of new features, emergent hazards, and new safety evidence. Update the safety case accordingly by adding appropriate safety goals, strategies, claims, and evidence.

ACTIVITY 10: End of Sprint Safety Gateway

At the end of each sprint, the entire team, led by the Safety Master must evaluate the safety of the system [64]. A release in which all of the safety stories associated with currently implemented system stories have been fully implemented and rigorously evaluated, is potentially safe for use. On the other hand, a release in which a system story has been implemented without all of its associated safety stories cannot be safe. Similarly, a system for which all safety stories have been

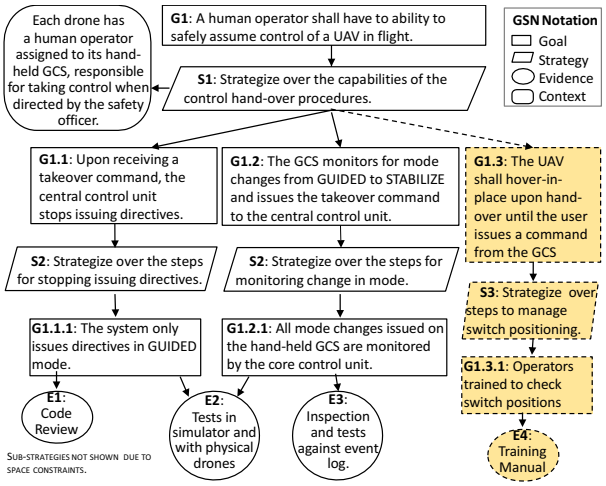

Fig. 4: The Safety Case is incrementally modified to reflect an emergent hazard and a temporary mitigation. In future sprints, as mitigating strategies are devised

implemented but not yet sufficiently inspected and evaluated is also not considered safe for use. Consider the following system story (SYS-5) and its dependency upon safety story (SAF-4).

System Story (SYS-5): Dronology shall directly control the flights of up to 50 concurrent UAVs.

Safety Story (SAF-4): UAVs must always maintain a MINIMUM_SEPARATION_DISTANCE of 3 meters.

Early sprints of Dronology delivered features to support multiple simultaneous flights; however, achieving trusted collision avoidance was more challenging and is requiring several ongoing rounds of prototyping and architectural refactoring. To pass the safety gateway that would allow field tests to continue, we introduced the following temporary constraint:

Safety Story (SAF-5): UAV flight routes shall not overlap.

This introduces the notion of *safety debt* as a form of technical debt [4, 30]. We define it as "unfulfilled safety obligations caused by prioritization decisions that expeditiously enable a working release without satisfying its safety requirements". Safety debt must be tracked and managed across the project so that unfulfilled safety stories and their associated design definitions can be clearly identified following each sprint. As in our previous example, temporary mitigations may be introduced for testing purposes; however, full safety can only be achieved when all specified safety stories have been implemented. The safety gateway is captured through *SafetyScrum* activity A10.

The nature and extent of activities associated with the end of sprint safety gateway are impacted by the domain of the system and whether the end-of-sprint aligns with a planned public or private release. In a certified domain, several sprints may need to be fully dedicated to hardening the system or preparing evidence needed for certification.

A10: End of Sprint Safety Gateway
Analyze the safety of the system at the end of each sprint. If necessary conduct "hardening" sprints to address safety concerns and/or to prepare systems in regulated domains for certification.

Earlier in this paper we introduced the role of the safety master for maintaining a safety focus across many of the identified activities. In particular, the safety master is responsible for leading and/or coordinating hazard analysis activities and ensuring that appropriate safety stories are specified (Activities 2, 3 and 7), holding the developers and/or architects of the system responsible for designing an arguably safe solution that satisfies those stories (Activity 4), influencing prioritization decisions to ensure that safety stories are scheduled in a timely way (Activity 6), ensuring that trace links are created for all hazards (Activity 5) in order to provide sufficient evidence to support Safety Assurance (Activity 9) and holding ultimate responsibility for safety-related release decisions at the end of each sprint (Activity 10).

V. An Experience Report of Applying Safety-Scrum

In this paper we have proposed a new process for eliciting, analyzing, specifying, and managing safety stories within the Scrum process. A full evaluation would require application of our approach in an entirely different project environment which is out of scope for this paper. However, we report our experiences in applying *SafetyScrum* across three new sprints dedicated to preparing Dronology for use in a public demo of river search and rescue. We qualitatively address the question of *how effectively does SafetyScrum guide the safety analysis, sprint planning and execution in new sprints, while maintaining flexibility associated with agile development processes?* The question of generalizability to other safety-critical projects is discussed in Section V-C.

A. Applying the Process

The search and rescue application of Dronology has been under discussion between our Dronology team and city emergency responders for several months. However, the project was officially launched in early 2018 through a planning and requirements elicitation meeting conducted with 15 key stakeholders including city administrators, search and rescue team members, and the fire chief. The meeting produced a set of use-case scenarios describing UAV support for search and rescue activities and a plan that the public demo would include multiple UAVs taking off from the river bank, flying a series of search patterns, transmitting thermal and visual images, inspecting key search areas in which victims were more likely to be found, and eventually returning the UAVs to their launch positions. The subsequent sprints focused on delivering crucial functionality and safety features needed to sufficiently increase the likelihood that the river rescue demo would be completed without UAV failures, mid-air collisions, hard landings, or close encounters with human bystanders. We describe the outcome of each *SafetyScrum* activity (A1-10)

ID	Safety Story
SAF-10	When in MISSION_PLANNING mode, the takeoff and positioning of all UAVs at the start of their first assigned routes shall be choreographed to avoid collisions.
SAF-11	UAVs returning to home shall follow choreographed flight paths to avoid collisions.
SAF-12	When in MISSION_PLANNING mode the system shall warn the user when routes overlap.
SAF-13	If the signal is lost between a UAV and both Dronology and the handheld, then the UAV shall return to its launch position safely.
SAF-14	Dronology shall prevent UAVs from flying outside the predefined area.
SAF-15	An internal Dronology GEOFENCE shall constrain the region in which UAVs shall fly
SAF-16	When the mission planner is initialized by the user, all current UAV flights shall be managed by the planner.

ID	Selected Design Definitions for SAF-10
DD-01	When in COORDINATED_TAKEOFF mode and commanded to takeoff, each UAV in the group will ascend to a unique takeoff altitude separated by at least 4 meters from other UAV altitudes.
DD-02	When in COORDINATED_TAKEOFF mode and a UAV reaches its target altitude it shall hover in place until it receives a command to proceed to the first waypoint of its prescribed flight route.
DD-03	When in COORDINATED_TAKEOFF mode and all UAVs have reached their target altitudes, each UAV shall fly in a direct path to the longitude and latitude of its first waypoint while maintaining its current altitude.

TABLE I: Safety Stories and sample Design Definitions for river search and rescue

and discuss the end-to-end application of the process across the first sprint, and the planning of the two subsequent sprints.

Discover System Stories & Init. Hazard Analysis: New system stories focused around the integration of a thermal camera and its imagery, the need to focus the search on predefined parts of the river (e.g., tree-lined outer bends, and "strainers" where victims might get trapped), and the need for UAVs to takeoff and land from a compact area. This resulted in the identification of three new system stories which served as the input for the preliminary hazard analysis (A1). Our team of 6 researchers and developers, including both hardware and software experts, spent approximately 2.5 hours brainstorming potential hazards, and identifying 66 potential hazards categorized as failures associated with preflight setup (21), handheld GCS (6), takeoff (11), landing (5), flight guidance (7), flight navigation (4), flight control (6), and miscellaneous incidents (6) (A2). The majority of miscellaneous hazards (e.g., rogue drones, or goose hits), and most hardware associated hazards (e.g., broken propeller, damaged batteries) were delegated to preflight inspections, checklists, and/or external entities such as the police. Of the remaining hazards, we identified 12 to be addressed during the three sprints.

Specify Safety Stories & Design Safety Solutions: Based on these hazards we specified seven safety stories (A3). For each of these we designed mitigations leading to 34 design definitions (A4) and then estimated the effort to implement them in terms of "story points" [5]. Safety stories and a selection of design definitions associated with the safety story for collision free takeoff are depicted in Table I. In addition to stories and design definitions we created the respective trace links (A5). This included trace links between hazards, safety stories, and design definitions (captured using Jira), and links to the concrete implementation (automatically provided through tagging github commit messages).

Sprint Planning: The overall goal for the three sprints was to implement the stories and their associated design definitions, and to *rigorously test* all functionality prior to the live river-rescue demo. Given a total of 42 estimated story points, and a viable project velocity of 30 story points per sprint, we created a sprint plan (A6) that included: Sprint 1: (18 story points, 12 "test" points), Sprint 2: (10 story points, 30 test points), and Sprint 3: (14 story points, 16 test points). These decisions were based on the need to deliver the most essential functionality in Sprints 1 and 2 with additional time dedicated to hardening the system and to rigorous testing. We assigned "optional" stories to Sprint 3 in case more important emergent hazards needed to be addressed. The sprint plan is summarized in Table II.

Sprint level hazard analysis & Sprint Execution: At the start of the first sprint we inspected the scheduled stories and design definitions for any new hazards (A7). No additional hazards were identified, most likely because we had recently completed the preliminary hazard analysis. However, during the sprint we visited the demo site at the river and observed that the designated takeoff space was quite narrow. As a result, we identified the new hazard that even small inaccuracies in GPS localization could cause a UAV to land close to the edge and topple into the river (A8). This hazard resulted in modifications to the UAV landing protocols that included visually checking UAV alignment over the ledge before issuing the final landing command. The flexibility of the sprint planning allowed us to prioritize implementation of the modified mitigation early in Sprint 2. No additional hazards or faults emerged during the development and testing of stories assigned to this sprint.

Safety Case & End of Sprint Safety Gateway: As new features were added and each sprint was completed, the SC was updated (A9) incrementally and the overall safety of the product assessed (A10). The safety gateways at the end of each sprint enabled us to systematically evaluate safety debt and provided a systematic approach, commensurate with practices for developing safety-critical software, for tracking progress towards our end goal and for making a go/no-go decision.

B. Qualitative Discussion

By applying our *SafetyScrum* approach to the river rescue scenario we are able to address both aspects of the posed research question. While we developed an initial plan for all three sprints, the flexibility of the agile process allowed us

Activity	Artifact Type	Total	S1	S2	S3
A1	System Stories	3	1	0	2
A2+A8	Hazards	12	7	3	2
A3	Safety Stories	7	3	3	2
A4	Design Definitions	34	16	10	8

Activity Type	Total	S1	S2	S3
Development	42	18	10	14
Testing	48	12	20	16

TABLE II: Summary of activities and stories assigned to each of the three sprints

to replace some of the stories that were initially assigned to Sprint 3 with new stories that addressed the emergent hazard related to restricted launch space at the river. At the same time, the safety-aspects of *SafetyScrum* provided the structure and rigor needed to guide us through the systematic hazard analysis and assurance process. In particular, the discipline of performing incremental hazard analysis, creating a safety case, and explicitly reviewing the product's safety in each Sprint Safety Gateway, raised awareness of safety issues, injected safety thinking into our agile development process, and resulted in a product with more clearly demonstrable safety. *SafetyScrum* therefore enabled us to achieve agility whilst simultaneously maintaining a focus on achieving critical safety goals.

C. Generalizability to Other Projects

This paper describes a safety process for use within the Scrum management environment. The process was developed as a result of our own experiences within only one project. While we cannot claim generalizability to other project environments, we contribute to the body of knowledge by documenting and publishing the process we utilized in our project. There is significant evidence (discussed in the related work in Section II) that practitioners are applying agile techniques to diverse regulated and non-regulated safety-critical domains ranging across autonomous cars, medical devices, and transportation systems – often without adequate process guidance. The question is therefore not "can it be done?" but rather "how can it be done?"

With respect to generalizability, we also note that the Dronology system meets the traditional definition of a safety critical system (i.e., a system whose failure or malfunction may result in death, serious injury to people, extensive loss of equipment, or property damage) [35], as UAVs can potentially fly into the paths of commercial airlines, crash into busy highways causing accidents, or collide with bystanders. However, the likelihood of high severity events may be less than those of other domains, such as avionics, autonomous cars, or medical devices, and this clearly influenced the process we developed. On the other hand, as a result of discussions with members of the University of Waterloo's Self-Driving Car project [51] it emerged that their safety-critical project had followed a similar, albeit undocumented, agile approach. While this suggests that *SafetyScrum* is generalizable to more diverse projects, we leave the customization of agile processes across diverse safety-critical domains as an open research question to be addressed in future work.

VI. Threats to Validity

Our study carries several threats to validity [68] which influence how our results should be interpreted. In terms of *internal validity* our method was developed as a result of lessons we learned in our own project environment. The project had external stakeholders, an experienced project team that included professional developers and UAV pilots, and was conducted over approximately 16 months. However, our approach has not been tested on external projects, and this paper should therefore be perceived as a case study that documents our experiences in a shareable form. In terms of *external validity*, different projects have unique characteristics which clearly impact the way in which an agile approach might be applied. Potential differentiators include the criticality level of the system, the role played by the software – for example, whether the software is embedded in a single system or whether it controls multiple external devices, the potential for breaking the system into constituent parts to be developed incrementally, and the extent to which the system readily supports isolation of safety- and non-safety critical components. Our UAV system can be described as medium criticality, non-embedded, with high potential for incremental delivery of diverse features. Furthermore, features such as the UI are easily separated from the core controllers. Our approach is likely to generalize across other systems exhibiting similar characteristics; however, we leave the broader question of applicability to other types of systems as an open question.

VII. Conclusion

Industries operating in safety-critical domains are expressing significant interest in adopting more agile approaches in order to be more responsive to market needs. However, traditional agile development processes are not designed to rigorously address hazards, faults, mitigation strategies, and compliance in a safety-critical, potentially regulated, project environment. To address this lack of guidance, we documented ten *SafetyScrum* activities which enable the flexibility of an agile process while promoting systematic safety analysis practices that are standard in safety-critical domains.

Hazard definitions, system and safety requirements, design definitions, features, and code for the first release of Dronology are publicly available at http://dronology.info.

Acknowledgment

Funding for this work has been provided by the US National Science Foundation grant US National Science Foundation Grants CCF:17417881 and CCF:1647342 and by the Austrian Science Fund (FWF): J3998-N31.

References

[1] DoIT Project Management Advisor. https://pma.doit.wisc.edu/size_factors.html, [Last accessed: 1/1/2018].

[2] Ardupilot: Open source autopilot software, http://ardupilot.org, [Last accessed: 2/28/2018].

[3] Adelard. Claims, Arguments and Evidence (CAE). https://www.adelard.com/asce/choosing-asce/cae.html, [Last accessed: 28/2/2018].

[4] E. Allman. Managing technical debt. *Commun. ACM*, 55(5):50–55, 2012.

[5] K. Beck. *Extreme programming explained - embrace change*. Addison-Wesley, 1990.

[6] K. Beck and et al. The Agile Manifesto, http://agilemanifesto.org, [Last accessed: 2/28/2018].

[7] BEL-V, BfS, CSN, ISTec, ONR, SSM, STUK. IEC 60880:2013: Licensing of safety critical software for nuclear reactors (common position of seven european nuclear regulators and authorised technical support organisations), 2013.

[8] S. Bellomo, I. Gorton, and R. Kazman. Toward agile architecture: Insights from 15 years of ATAM data. *IEEE Software*, 32(5):38–45, 2015.

[9] B. W. Boehm and R. Turner. Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods. In *Proc. of the 26th Int'l Conf. on Software Engineering*, pages 718–719, 2004.

[10] J. Cleland-Huang. Traceability in agile projects. In *Software and Systems Traceability*, pages 265–275. Springer, 2012.

[11] J. Cleland-Huang. Safety stories in agile development. *IEEE Software*, 34(4):16–19, 2017.

[12] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman. Software traceability: trends and future directions. In *Proc. of the on Future of Software Engineering*, pages 55–69, 2014.

[13] J. Cleland-Huang, S. Rayadurgam, P. Mäder, and W. Schäfer. Software and systems traceability for safety-critical projects (dagstuhl seminar 15162). *Dagstuhl Reports*, 5(4):76–97, 2015.

[14] C. Comar, F. Gasperoni, and J. Ruiz. Open-do: An open-source initiative for the development of safety-critical software. In *Proc. of the 4th IET Int'l Conf. on Systems Safety*, pages 1–5. IET, 2009.

[15] A. M. Davis, Ó. D. Tubío, A. M. Hickey, N. J. Juzgado, and A. M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *Proc. of the 14th IEEE Int'l Conf. on Requirements Engineering*, pages 176–185, 2006.

[16] M. Denne and J. Cleland-Huang. *Software by Numbers - Low-Risk, High-Return Development*. Prentice Hall, 2004.

[17] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe. A decade of agile methodologies: Towards explaining agile software development, 2012.

[18] O. Doss and T. P. Kelly. Challenges and opportunities in agile development in safety critical systems: A survey. *SIGSOFT Softw. Eng. Notes*, 41(2):30–31, May 2016.

[19] ECSS. ECSS-E-40C: principles and requirements applicable to space software engineering, 2009.

[20] D. Firesmith. Engineering safety requirements, safety constraints, and safety-critical requirements. *Journal of Object Technology*, 3(3):27–42, 2004.

[21] B. Fitzgerald, K.-J. Stol, R. O'Sullivan, and D. O'Brien. Scaling agile methods to regulated environments: An industry case study. In *Proc. of the 35th Int'l Conf. on Software Engineering*, pages 863–872. IEEE, 2013.

[22] Food and Drug Administration. *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices*, 2005.

[23] M. Galster and S. Angelov. Understanding the use of reference architectures in agile software development projects. In *Proc. of the 9th European Conf. on Software Architecture*, pages 268–276, 2015.

[24] A. Ghazarian. Traceability patterns: an approach to requirement-component traceability in agile software development. In *Proc. of the 8th Conf. on Applied Computer Science*, pages 236–241. WSEAS, 2008.

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

[26] O. Gotel and A. Finkelstein. Extended requirements traceability: results of an industrial case study. In *Proc. of the 3rd Int'l Symp. on Requirements Engineering*, 1997.

[27] P. J. Graydon and C. M. Holloway. An investigation of proposed techniques for quantifying confidence in assurance arguments. *Saf. Sci.*, 92:53–65, feb 2017.

[28] W. S. Greenwell, E. A. Strunk, and J. C. Knight. Failure analysis and the safety-case lifecycle. In *Human Error, Safety and Systems Development*, pages 163–176, 2004.

[29] J. Hill and S. Tilley. Creating safety requirements traceability for assuring and recertifying legacy safety-critical systems. In *Proc. of the 18th IEEE Int'l Requirements Engineering Conf.*, pages 297–302, 2010.

[30] C. Izurieta, I. Ozkaya, C. B. Seaman, and W. Snipes. Technical debt: A research roadmap report on the 8th ws on managing technical debt). *ACM SIGSOFT Software Engineering Notes*, 42(1):28–31, 2017.

[31] H. Jonsson, S. Larsson, and S. Punnekkat. Agile practices in regulated railway software development. In *Proc. of the IEEE 23rd Int'l Symp. on Software Reliability Engineering Workshops*, pages 355–360. IEEE, 2012.

[32] T. P. Kelly and J. A. McDermid. Safety Case Construction and Reuse Using Patterns. In *Safe Comp 97*, pages 55–69. Springer London, 1997.

[33] T. P. Kelly and J. A. McDermid. A systematic approach to safety case maintenance. *Reliability Engineering & System Safety*, 71(3):271–284, 2001.

[34] X. Larrucea, A. Combelles, and J. Favaro. Safety-critical software [guest editors' introduction]. *IEEE Software*, 30(3):25–27, 2013.

[35] N. G. Leveson. *Safeware: System Safety and Computers*. ACM, New York, NY, USA, 1995.

[36] W. Lin and X. Fan. Software development practice for fda-compliant medical devices. In *Proc. of the 2009 Int'l Conf. on Computational Sciences and Optimization*, volume 2, pages 388–390. IEEE, 2009.

[37] R. R. Lutz and R. M. Woodhouse. Requirements analysis using forward and backward search. *Ann. Software Eng.*, 3:459–475, 1997.

[38] R. C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[39] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. Easy approach to requirements syntax (EARS). In *Proc. of the 17th IEEE Int'l Requirements Engineering Conf.*, pages 317–322, 2009.

[40] F. McCaffery, M. Pikkarainen, and I. Richardson. Ahaa–agile, hybrid assessment method for automotive, safety critical smes. In *Proc. of the 30th Int'l Conf. on Software Engineering*, pages 551–560. ACM, 2008.

[41] M. W. Mwadulo. Suitability of agile methods for safety-critical systems development: A survey of literature.

[42] The Open-DO Initiative, www.open-do.org, 2013.

[43] R. F. Paige, R. Charalambous, X. Ge, and P. J. Brooke. Towards agile engineering of high-integrity systems. In *Proc. of the Int'l Conf. on Computer Safety, Reliability, and Security*, pages 30–43. Springer, 2008.

[44] M.-A. Peraldi-Frati and A. Albinet. Requirement traceability in safety critical systems. In *EDCC-CARS*, pages 11–14, 2010.

[45] M. Rahimi, W. Goss, and J. Cleland-Huang. Evolving requirements-to-code trace links across versions of a software system. In *Int'l Conf. on Software Maintenance and Evolution*, pages 99–109, 2016.

[46] R. Rasmussen, T. Hughes, J. Jenks, and J. Skach. Adopting agile in an fda regulated environment. In *Proc. of the 2009 Agile Conf.*, pages 151–155. IEEE, 2009.

[47] M. Rath, J. Rendal, J. L.C.Guo, J. Cleland-Huang, and P. Mäder. Traceability in the wild: Automatically augmenting incomplete trace links. In *Proc. of the 40th Int'l Conf. on Software Engineering*, 2018.

[48] D. J. Reifer. Software failure modes and effects analysis. *IEEE Trans. Reliability*, R-28,3:247–249, 1979.

[49] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang. Mind the gap: assessing the conformance of software traceability to relevant guidelines. In *Int'l Conf. on Software Engineering*, pages 943–954, 2014.

[50] S. Robertson. Requirements trawling: techniques for discovering requirements. *Int. J. Hum.-Comput. Stud.*, 55(4):405–421, 2001.

[51] J. A. Ross, A. Murashkin, J. H. Liang, M. Antkiewicz, and K. Czarnecki. Synthesis and exploration of multi-level, multi-perspective architectures of automotive embedded systems (sosym abstract). In *Proc. of the 20th ACM/IEEE Int'l Conf. on Model Driven Engineering Languages and Systems*, page 178, 2017.

[52] P. A. Rottier and V. Rodrigues. Agile development in a medical device company. In *Proc. of the Agile 2008 Conf.*, pages 218–223. IEEE, 2008.

[53] RTCA/EUROCAE. DO-178C/ED-12C: Software considerations in airborne systems and equipment certification, 2011.

[54] T. Stålhane and T. Myklebust. Agile safety analysis. *ACM SIGSOFT Software Eng. Notes*, 41(2):27–29, 2016.

[55] T. Stålhane, T. Myklebust, and G. Hanssen. The application of safe scrum to iec 61508 certifiable software. In *Proc. of the 11th Int'l Probabilistic Safety Assessment and Management Conf. and the Annual European Safety and Reliability Conf.*, pages 6052–6061, 2012.

[56] Z. Stephenson, J. McDermid, and A. Ward. Health modelling for agility in safety-critical systems development. 2006.

[57] N. R. Storey. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[58] D. J. Sturtevant. Modular architectures make you agile in the long run. *IEEE Software*, 35(1):104–108, 2018.

[59] K. J. Sullivan, J. B. Dugan, and D. Coppit. The Galileo fault tree analysis tool. In *Digest of Papers: FTCS-29, The 29th Int'l Symp. on Fault-Tolerant Computing*, pages 232–235. IEEE Computer Society, 1999.

[60] J. Sutherland. Future of scrum: Parallel pipelining of sprints in complex projects. In *Proc. of the AGILE 2005 Conf.*, pages 90–102, 2005.

[61] A. Uhl and S. W. Ambler. Point/counterpoint: Model driven architecture is ready for prime time / agile model driven development is good enough. *IEEE Software*, 20(5):70–73, 2003.

[62] U.K. Ministry of Defence. Defence Standard 00-56, Issue 7: Safety Management Requirements for Defence Systems. Part 1: Requirements, 2017.

[63] S. H. VanderLeest and A. Buter. Escape the waterfall: Agile for aerospace. In *Proc. of the 28th IEEE/AIAA Digital Avionics Systems Conf.*, pages 6–D. IEEE, 2009.

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

[65] Y. Wang, J. Ramadani, and S. Wagner. An exploratory study on applying a scrum development process for safety-critical systems. In *Proc. of the 18th Int'l Conf. on Product-Focused Software Process Improvement*, pages 324–340, 2017.

[66] W. Wu and T. Kelly. Managing architectural design decisions for safety-critical software systems. In *Proc. of the 2nd Int'l Conf. on Quality of Software Architectures*, pages 59–77, 2006.

[67] C. Yang, P. Liang, and P. Avgeriou. A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, 111:157–184, 2016.

[68] R. K. Yin. *Case study research and applications: Design and methods*. Sage publications, 2017.