Towards Applying a Safety Analysis and Verification Method based on STPA to Agile Software Development

Yang Wang
University of Stuttgart, Germany
yang.wang@informatik.uni-stuttgart.de

Stefan Wagner
University of Stuttgart, Germany
stefan.wagner@informatik.uni-stuttgart.de

ABSTRACT

Agile methodologies are becoming widespread in modern software development. However, due to a lack of safety assurance activities, agile methods are criticized for being inadequate for the development of safe software. Safety analysis and safety verification are complementary methods for safety assurance. Yet, both usually rely on traditional, waterfall-like processes. Therefore, it is strongly needed to integrate an appropriate safety analysis approach into agile software development processes driving architecture design and verify the safe design at the code level.

This paper presents a novel agile process model "S-Scrum" based on the existing development process "Safe Scrum" and extended by a safety analysis method and a safety verification approach based on STPA (System-Theoretic Process Analysis).

The proposed agile development process S-Scrum can be separated into three parts: (1) performing safety-guided design by STPA inside each sprint. (2) Verifying safety requirements at the code level by using model checking. (3) Replacing traditional RAMS (Reliability, Availability, Maintainability, Safety) validation on the final product by STPA safety analysis. We adopt other aspects from the original Safe Scrum.

Finally, the feasibility of S-Scrum is illustrated with the example of an airbag system.

CCS Concepts

•**Software and its engineering** → **Agile software development; Software safety;** *Requirements analysis; Formal software verification;*

Keywords

Agile methods, STPA, Safety analysis, Safety verification, Safety-critical sytems

1. INTRODUCTION

Agile has gained a good reputation for its higher customer satisfaction, lower defect rates, faster development times and as a solution to rapidly changing requirements [7]. Nevertheless, there are still limitations of agile processes for developing safety-critical software [25]. Although agile methods highlight improving quality, the quality control mechanisms have not proven to be adequate to assure that the product is safe.

For engineering safety in agile methods, most researchers prefer to combine this lightweight approach with traditional development processes relying on safety standards, like ISO 26262 [24] and IEC 61508 [9]. They attempt to balance discipline and agility as the solution for safety-critical systems [7]. However, a continuously changing architecture design makes this balance the bottleneck. Architecture acts as a vital role in the basic safety strategy for traditional safety assurance. Without a stable architecture in agile development processes, little safety analysis works well. Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA) are two classic safety analysis methods, yet both of them need an up-front architecture design. STPA, however, advocates safety-guided design. It is a novel safety analysis approach proposed by Leveson [18], for the purpose of identifying safety requirements and constraints at the system level, to drive a safe design. It has been successfully used in a lot of domains [1] [6] [11], and shown to be more effective and efficient than classic methods [3]. A novel software safety analysis and verification method based on STPA is proposed in [2]. Abdulkhaleq, Wagner, and Leverson then extend it to a comprehensive safety engineering approach [5], which motivates us to apply the method to an existing agile process for safety-critical systems−Safe Scrum. We concentrate on integrating STPA-based safety analysis and safety verification of the safety engineering approach into agile methods. The proposed agile development process in our paper is still in the concept stage, we name it "S-Scrum" for further exploration and investigation.

Problem Statement: Although there exist a lot of domain-specific safety standards for development processes, they are all based on traditional sequential development processes and do not mandate any specific safety assurance activities. Safety analysis and safety verification are mentioned as two critical safety assurance activities. However, current safety analysis technologies are inadequate for agile methodologies due to a lack of a stable architecture design. Thus, it is an uncultivated land for exploring and integrating an appropriate safety analysis and safety verification technology into

CSED'16, May 14-15, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4157-8/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2896941.2896948

agile methods.

Research Objectives: The overall objective of our article is to fill the gap on safety assurance activities in agile software development processes by integrating a novel safety analysis and verification approach based on STPA into the existing agile process model Safe Scrum. We aim to enhance the handling of safety of the original Safe Scrum, while we increase the agility at the same time. We keep Scrum adherent to safety standards by adopting Safe Scrum and improve the agility of Safe Scrum by using STPA.

Contributions: We propose a novel agile development process S-Scrum by integrating a safety analysis and verification approach in Safe Scrum.

1. We perform safety-guided design in an agile way by integrating STPA into each sprint before reaching the design level in Safe Scrum.

2. We integrate a formal verification method with model checking into Safe Scrum to formalize and verify the safety requirements of STPA at the code level.

3. We replace traditional RAMS validation in Safe Scrum by STPA on the product increment.

2. RELATED WORK

To the best of our knowledge, there exists little research published on the utilization of safety analysis technologies and also formal verification in agile methodologies. Thus, we separate the related work into two sections. First, we demonstrate the state of using agile methodologies in safety-critical domains. Second, we introduce existing work on mapping formal methods into agile technologies for safety requirements verification.

Safe Scrum, proposed by Stålhale and Myklebust [23], is motivated by the need to make it possible to use methods that are flexible with respect to planning, documentation and specification while still being acceptable to IEC 61508, as well as making Scrum a practically useful approach for developing safety-critical systems. Safe Scrum is a considerable success for its innovative combination. However, too much adherence to the safety standard IEC 61508 makes the process lacking agility. First, all the additional safety assurance activities are kept outside Scrum. Second, each sprint in Scrum should be swarming rather than sequential as mini waterfall or mini V-model [20]. We believe that safety-guided design is strongly needed in agile methods instead of the purely "add-on" safety assurance.

Although Safe Scrum has not mentioned any safety analysis method, Lauritsen and Stålhale [17] did preliminary guided research on generally integrating two FMEA activities (Functional FMEA and Detailed FMEA) into RUP and XP. In XP 2015, Stålhale and Myklebust introduced agile safety analysis [22]. However, most of their research focuses on the connection between user stories and FMEA, not the whole agile software development process.

In addition, Ge, Paige and McDermid [13] published an iterative approach to develop safety-critical software. Vuori [26] proposed a hybrid model like Safe Scrum. However, both of them suggest an up-front design, which is contrary to the nature of Scrum [8]. Gary et al. [12] mentioned design for safety in agile methods in their paper, but no safety analysis technology is explored.

In comparison to the above references, we apply an appropriate safety analysis technique, STPA, in agile methods, to abandon this heavy weight architecture up-front and still keep safety in agile. Moreover, the safety requirements should be captured into the code.

A lot of research has integrated agile practices with formal methods, but little of them connects their formal verification approaches with any safety analysis methods.

Shafiq and Minhas [21] proposed integrating formal verification from the requirement specifications using automated code driven test. However, the verification for safety requirements is mostly not automated like functional requirements [15]. Eleftherakis and Cowling [10] proposed a lightweight formal development process "XFun". Ghezzi et al. [14] suggested an agile verification environment "AGAVE" that enables developers to use model checking. Both of them mentioned the need to support safety analysis. Therefore, we select the appropriate approach STPA to drive safe design, and integrate STPA-based model checking for safety verification.

3. BACKGROUND

3.1 STPA

We suggest STPA to confront the aforementioned problems. It is a new hazard analysis technique based on systems thinking and a new model of accident causation based on systems theory rather than reliability theory. It consists of two main steps: (1) Identify the potential for inadequate control of the system that could lead to a hazardous state. (2) Determine how each potentially hazardous control action identified in step 1 could occur [18].

Now we describe the method in more detail. Before the concrete safety analysis starts, we identify the accidents and hazards at the system level. To apply STPA, we start from the system control structure. Based on the control structure, we evaluate each control action against four general types of hazardous behaviors: (1) a control action required for safety is not provided; (2) an unsafe action is provided; (3) a potentially safe control action is provided too early, too late or out of sequence, and (4) a safe control action is stopped too soon or applied too long. Then we derive the initial safety requirements from the unsafe control actions. By using STPA step 2, we focus on the causal factors for the unsafe control actions of step 1. We identify the process model and the variables that affect the safety of the control actions and include them in the software controller in the control structure diagram to document how each unsafe control action could occur. The process model contains three types of variables: Internal variables of the software controller, interaction interface variables, which receive data/command of the environmental components and environmental variables of other components in the system interacting with the software controller. After that, we identify detailed software safety requirements by using Boolean operators AND and OR, to constrain the unsafe combinations of process variables.

We recommend this novel technique for two reasons: (1) The current safety analysis techniques, such as FMEA and FTA, assume that accidents are caused by component failures, which is mostly not true for software. The primary

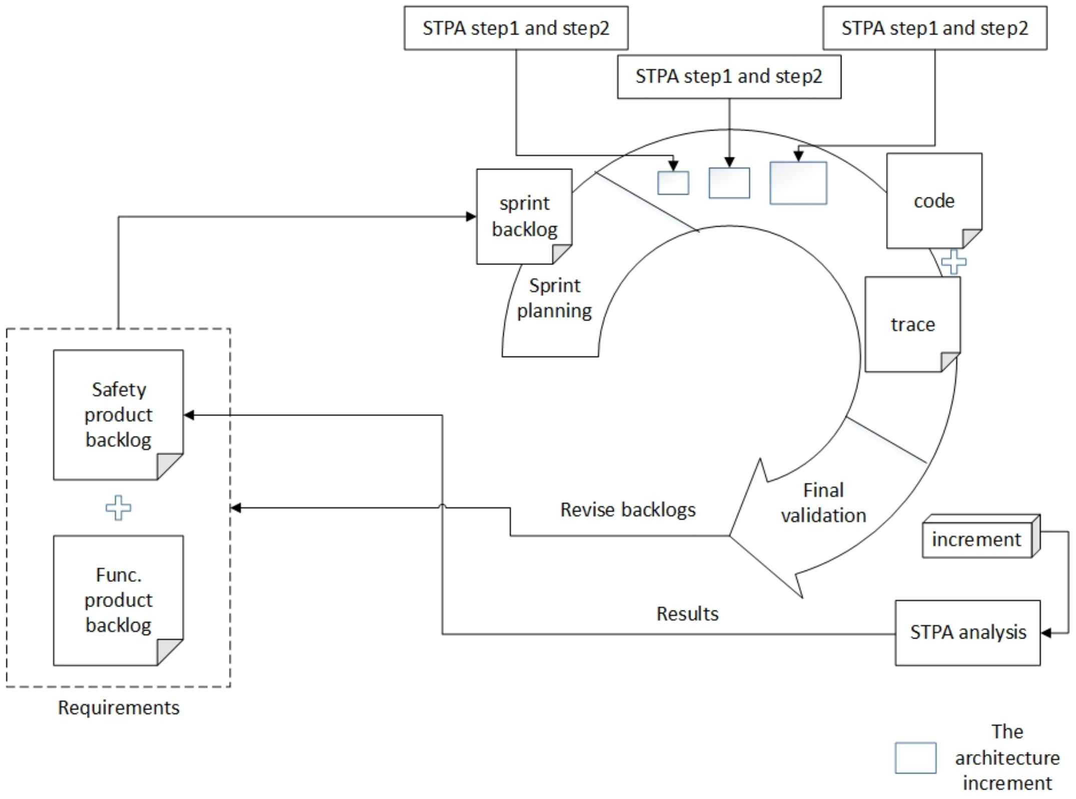

Figure 1: Mapping STPA in S-Scrum

advantage of STPA is that it emphasises causal factors from the system view, such as component interaction accidents or cognitively complex human decision-making errors. (2) Current safety analysis techniques start from a complete design, which is not consistent to agile methodologies. STPA, on the contrary, provides the necessary information to guide the design process.

3.2 Software Safety Verification

The safety requirements verification method by using model checking is the second part of the novel safety engineering approach [5]. Based on STPA, [5] provides a link between the safety analysis at the system level and safety verification at the code level. The safety verification approach consist of three steps: (1) Formulating the safety requirements using temporal logic (LTL/CTL). (2) Modeling the source code as input model (PROMELA) by using Modex [27]. (3) The safety requirements specification and the input model are verified by using the SPIN model checker [16].

Now we describe the software safety verification process in more detail. First, by using the results of STPA, we fomulate the corresponding software safety requirements, which have been identified and expressed by Boolean operators. Then we map them into a formal specification in LTL. An LTL formula can be defined over a set of atomic propositions (Boolean operators and temporal operators). Second, we extract the input model for the model checker SPIN. The input model is written in PROMELA code which is similar to C code. We extract the input model automatically by using Modex. Finally, the verification activities with the SPIN model checker can be performed. Thus, we can verify if the SPIN model of the software conforms to the software safety requirements identified in STPA step 1 and step 2.

We adopt this method for two reasons: (1) this safety verification method is extended from STPA, which we use preliminary for safety analysis. (2) It can be applied at an early stage of development on existing software. The idea is in align with safety-guided design. (3) The proposed methodology is appropriate for agile methods, since it can be iterated until a software code that fulfills the software safety requirements is reached [4]. The comprehensive safety engineering approach is based so far on a V-Model software development process [5]. Hence, we propose an agile software development process instead.

4. S-SCRUM CONCEPT

4.1 Stage 1: STPA in S-Scrum

In this section, we integrate STPA in Safe Scrum. We extend Safe Scrum in three aspects: (1) During each sprint we integrate STPA as safety-guided design. (2) At the end of each sprint, we use STPA on the product instead of a RAMS validation. (3) We replace the final RAMS validation with STPA. The other parts which are still kept consistent to Safe Scrum are: (1) The environment description and the SSRS phases 1-4 (concept, overall scope definitions, hazard and risk analysis and overall safety requirements). (2) Test Driven Development. (3) Safety product backlog. (4) A safety expert. For other risk and hazard analysis during the process, we also use STPA.

In STPA, the accidents are regarded as resulting from inadequate control. Thus, the control structure (architecture)

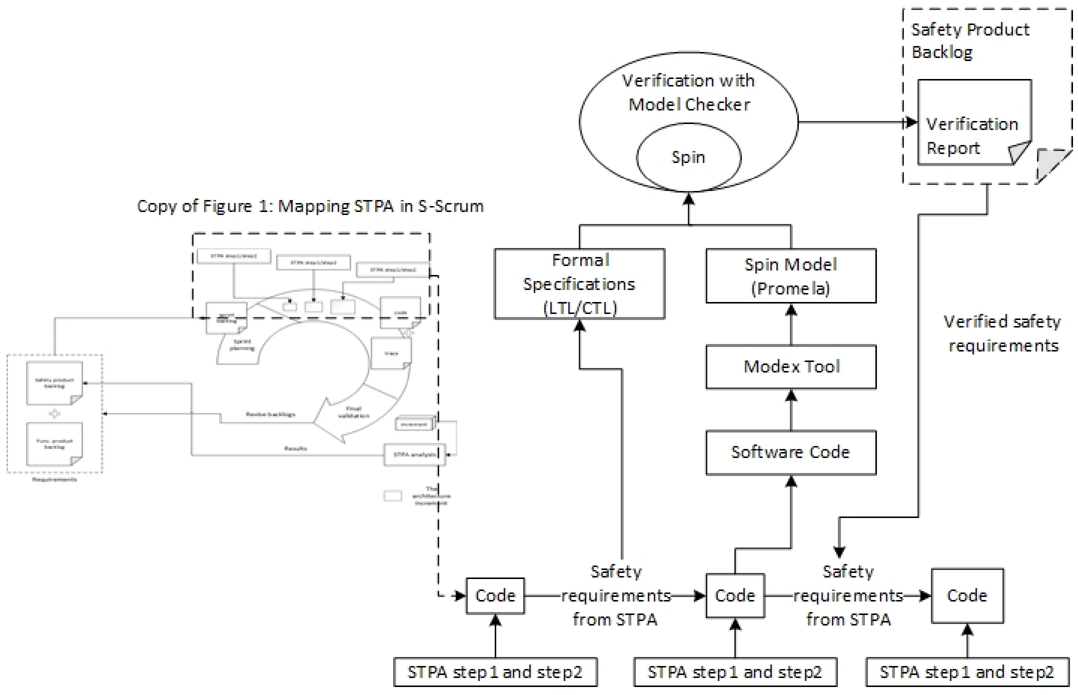
Safety Product Backlog
Verification with Model Checker
Spin
Verification Report
Copy of Figure 1: Mapping STPA in S-Scrum
Formal Specifications (LTL/CTL)
Spin Model (Promela)
Modex Tool
Software Code
Verified safety requirements
Code
Safety requirements from STPA
Code
Safety requirements from STPA
Code
STPA step1 and step2
STPA step1 and step2
STPA step1 and step2

Figure 2: Safety Verification in S-Scrum

is considered as the cross point between STPA and Safe Scrum.

As we can see from Figure 1, we start from a general description of the environment and initial systems through SSRS phases 1-4 of IEC 61508. An approximate safety target or safety-related targets are determined in up-front plannings and story boarding in agile [19].

During each sprint, we apply STPA in development when there is a sufficient amount of new architecture design. The safety analysis result is regarded as a driving force for the next architecture design.

By applying STPA step1 and step2, the unsafe control actions and the factors that could lead to violating the safety constraints are determined. More detailed safety requirements are to be elicited depending on the stepwise system design.

After each sprint, a product is created. We finally apply STPA to it for the following reasons: (1) Getting a final safety assessment. (2) Combining with safety verification on the system level. (3) Driving further sprint development.

All the safety analysis activities aforementioned are executed by a safety expert and the results are documented in the safety product backlog.

4.2 Stage 2: Safety Verification in S-Scrum

After identifying the safety requirements by using STPA in Safe Scrum, the S-Scrum development team designs and develops the software code based on the safety requirements iteratively and incrementally.

Yet, we still need to check whether the source code fulfills the derived safety requirements. Therefore, in this section we verify the consistency of safety requirements driving the design decisions and the software after the design.

We use model checking in each sprint of S-Scrum following STPA step1/step2. In parallel with the development team, the safety expert formulates the safety requirements as LTL specifications. After the code is produced by the development team, the safety expert automatically translates the software code into a PROMELA model, and then verifies the PROMELA model with the safety requirements specifications one by one using the SPIN model checker. The generation process is automatic, but it still needs manual intervention. To this end, the verification report is formulated as a part of the safety product backlog, which drives the traceability of safe development during each sprint, see Figure 2.

We apply the verification process iteratively in each sprint of "S-Scrum", when there are new safety requirements derived from STPA and the software architecture produced. STPA safety analysis, the safety requirements specification and manual parts of model checking are performed by a safety expert in the S-Scrum team. We fomulate the safety verification report as a part of the safety product backlog for minimizing the proceeding documents [20].

5. EXAMPLE: AIRBAG SYSTEMS

To illustrate our approach S-Scrum, we use air bag systems as an example. The paper is the starting point of S-Scrum, so we demonstrate the airbag system as a concept.

5.1 System Overview

Airbag systems are equipped as one of the safety-critical systems in modern cars to protect the occupants from fatal injuries. According to the ISO functional safety standard 26262, new airbag systems have to comply with ASIL D (Automotive Safety Integrity Level D) for unintended deployment of the airbag. Currently, it is only required to comply with ASIL B. An airbag systems can be divided into three major parts: sensors, crash evaluation and actuators. Once an impact happens, it would be detected by acceleration sensors and pressure sensors. Rollover accidents are typically detected by roll rate sensors. Through the sensor information, microcontrollers decide whether the sensed acceleration corresponds to a crash situation or not. The deployment of the airbags is activated if there was indeed a critical crash. Using airbags can protect the passengers from critical injury.

5.2 Applying S-Scrum in Airbag System Development

5.2.1 Environment and SSRS 1-4

Before STPA starts, we apply the original Safe Scrum system engineering process performing environment description and SSRS 1-4 to get an overall system safety goal.

System Safety Goal: During a critical crash, the airbag system should protect the passengers from being injured.
Accident (AC.1): The occupants in the target vehicle are injured when a traffic accident occurred.
Hazard (H.1): The airbag is not ignited even though a critical crash occurred.
Hazard (H.2): The airbag is deployed unintentionally, which means that it is ignited even though no crash at all or only a non-critical crash has occurred.
Hazard (H.3): The airbag is ignited after T = 45ms.

The highest-level safety requirements transform directly from the identified hazard for the system.

System Safety Requirements (SR.1): If a critical crash occurred, the airbag must be ignited.
System Safety Requirements (SR.2): If there was no crash or only a non-critical crash, the airbag must not be ignited.
System Safety Requirements (SR.3): If a critical crash occurred, the airbag must be ignited before T = 45ms:

The hazard and related safety requirements must be recorded in the safety product backlog in the first sprint planning meeting and taken as reference through out the whole development process.

The verification of safety requirements at the system level is performed by the STPA on the product as user acceptance tests after the development process is finished.

5.2.2 Safety-Guided Design in Sprint

We use STPA starting with the first forming architecture of the software code. The safety expert and development team discuss and decide the times they will perform STPA in the daily Scrum meetings. The process of performing STPA is as follows:

Safety Control Structure Diagram:
We start with building the control structure of the airbag system, see Figure 3. The control structure depicts not only the components at a high level of airbag systems, but also the main interconnections. There should be two micro-controllers for decreasing the hazard of unintended airbag deployment. Due to the same functions (one of them is for redundancy), we integrate them into one micro-controller in the control structure diagram. The deployment (actor) of the airbag is secured by two protection mechanisms, the Field Effect Transistor (FET) controls the power supply

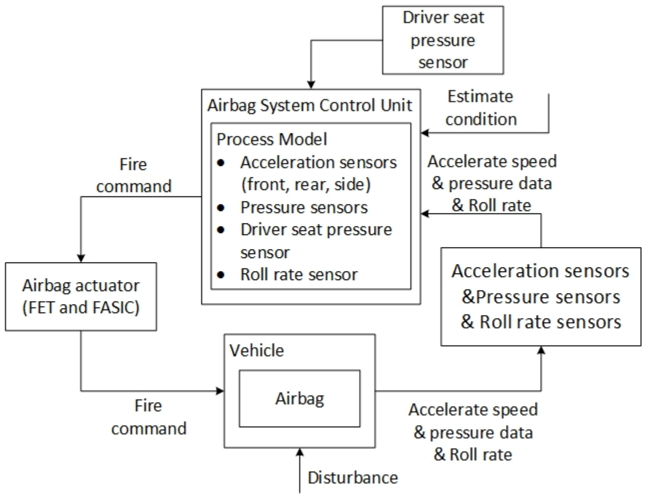
Driver seat pressure sensor
Airbag System Control Unit
Estimate condition
Process Model
Acceleration sensors (front, rear, side)
Pressure sensors
Driver seat pressure sensor
Roll rate sensor
Fire command
Accelerate speed & pressure data & Roll rate
Airbag actuator (FET and FASIC)
Acceleration sensors &Pressure sensors & Roll rate sensors
Vehicle
Airbag
Fire command
Accelerate speed & pressure data & Roll rate
Disturbance

Figure 3: Airbag System Control Structure

for the airbag squibs. The Firing Application Specific Integrated Circuit (FASIC) controls the airbag squib. Only if the control unit receive the signals from the sensors and FET has enough electrical power, the FASIC will ignite the airbag squib. The sensors we considered in the architecture are: (1) two acceleration sensors and two pressure sensors to detect front or rear crashes (x direction and -x direction acceleration); (2) driver seat pressure sensor to detect the present of occupants; (3) angular rate or roll rate sensors to detect rollover accidents.

The input of this step is a certain amount software code with architecture during sprint. The output is a safety control structure diagram with a process model, see Figure 3

STPA step 1: Identify Unsafe Control Actions:
Based on the control structure, we identify an example of potentially hazardous control actions which would violate system design constraints using STPA step1. The Unsafe Control Actions (UCA) are formulated into four general hazardous types (Providing, Not Providing, Providing too late or too soon, Applying too short or too long), illustrated in Table 1. Then we formulate the safety requirements from UCAs, see Table 2.

The input of this step is the control structure diagram with control actions. The output is the initial safety requirements based on UCAs.

STPA step 2: Identify causal factors
We identity each UCA with the causal factors, which could violate the safety requirements of each UCA from STPA step 1. For example, we focus on the UCA.1, and summarise the possible causal factors in the process model of the micro-controller, see Table 3.

In the micro-controller process model of the airbag system, we formulate four types interaction interface variables that affects the safety of these control actions: (1) acceleration speed; (2) pressure intensity; (3) driver seat pressure intensity; (4) roll rate. We determine the status of each variable, which could lead to the UCA. The casual factors can also be derived from the sensor model, as listed in Table 3.

Table 1: STPA Step 1 Unsafe Control Action

Airbag System Control Unit				
Cotrol Action (CA.)	Not Provided	Provided	Too Soon /Late	Too Long /Short
Send fire command	UCA.1: Not sending fire command is hazardous if there was a critical crash. [H.1]	UCA.2: A fire command is needlessly sent to the FET and FASIC, thus causing an unintended deployment of the airbag. [H.2]	UCA.3: The fire command for the airbag in case of a crash is delayed, thus causing the airbag to be ignited too late. [H.1][H.3]	/

Table 2: Initial Safety Requirements from STPA Step 1

Related UCAs	Corresponding Safety Constrains
UCA.1	SSR.1: The airbag system control unit shall provide fire command when there was a critical crash.
UCA.2	SSR.2: The airbag system control unit shall not send fire command when there was no crash or non-critical crash.
UCA.3	SSR.3: The airbag system control unit must send the fire command T <= 45ms, when there was a critical crash.

5.2.3 *Safety Requirements Verification in Sprint*

Mapping safety requirements to a formal specification

Based on the textual safety requirements in Table 2 and connecting with the causal factors in Table 3, the safety expert in S-Scrum team, who has the experience of Modex, translates them into a formal specification in LTL in parallel with the code development process. The verification of SSR.3 is beyond the scope of software verification. Thus, we focus on the first two safety requirements. See examples:

SSR.1 = []((acceleration >= safe acceleration threshold & & pressure intensity >= safe pressure threshold & & driver state == present) -> (send fire command))
SSR.2 = []((acceleration <= safe acceleration threshold | | pressure intensity <= safe pressure threshold | | driver state == present) - > !(send fire command))

Verifying and testing of the safety requirements

After the development team built enough code, we verify the safety requirements specification in LTL by transforming the software code into a PROMELA input model for the SPIN model checker. After the automatic proceedings in the SPIN model checker, we record the results in the safety product backlog. The safety requirements which are not satisfied will be discussed by the S-Scrum team to further build in and prioritization or give the reasons to remove the requirements from the safety product backlog.

6. CONCLUSION AND FUTURE WORK

In this paper, we extend Safe Scrum by integrating a novel safety analysis and safety verification approach based on STPA and propose a novel development process named S-Scrum. By doing this, we can apply an agile development process for safety-critical systems with a safety-guided unstable architecture. First, we focus on the unstable architectures which prevent using agile methods into safety-critical systems. Second, we integrate a novel safety analysis method STPA for safety-guided design to confront the changing architectures. The verification of the safety requirements is performed by the model checker SPIN, which seamlessly supports the continuous safety-critical software development. Finally, the traditional RAMS validation in Safe Scrum is replaced by STPA for keeping the agility in S-Scrum. Rather than a hybrid combination between safety standards and agile development processes, we believe that it would be a good direction from the standpoint of the nature of agile methods to solve safety-related problems.

However, we have not validated S-Scrum in a realistic project. A student project at our institute will be started for the continuous exploration and investigation of S-Scrum.

7. REFERENCES

[1] A. Abdulkhaleq and S. Wagner. Experiences with applying STPA to software-intensive systems in the automotive domain. *STAMP Conference at MIT, Boston, USA*, 2013.

[2] A. Abdulkhaleq and S. Wagner. A software safety verification method based on system-theoretic process analysis. In *Computer Safety, Reliability, and Security*, pages 401–412. Springer, 2014.

[3] A. Abdulkhaleq and S. Wagner. A controlled experiment for the empirical evaluation of safety analysis techniques for safety-critical software. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, page 16. ACM, 2015.

[4] A. Abdulkhaleq and S. Wagner. Integrated safety analysis using systems-theoretic process analysis and software model checking. In *Computer Safety, Reliability, and Security*, pages 121–134. Springer, 2015.

[5] A. Abdulkhaleq, S. Wagner, and N. Leveson. A comprehensive safety engineering approach for software-intensive systems based on STPA. *Procedia Engineering*, 128:2–11, 2015.

[6] B. Antoine. *Systems theoretic hazard analysis (STPA) applied to the risk review of complex systems: an example from the medical device industry*. PhD thesis, Massachusetts Institute of Technology, 2013.

Table 3: STPA Step 2 Causal Factors for UCA.1

UCA.1 Not sending fire command is hazardous if there was a critical crash	
Hazardous Scenario	Associated Causal Factors Leading to UCA.1
(1) Process Model ABS Enabled: [Yes, No] Driver Present: [Yes, No] Acceleration Speed >= Threshold: [Yes, No] Roll Rate >= Threshold: [Yes, No]	(1)Airbag system control unit does not detect the driver is present, thus the airbag system control unit does not been enable. (2)Acceleration threshold is incorrect and allows the vehicle get an abnormal acceleration without sending a signal.
(2) Sensors Acceleration Sensors Pressure Sensors Driver Seat Pressure sensor Roll Rate Sensors	(1)Pressure sensor is unable to detect pressure due to road and weather conditions (2)Noise is not adequately filtered and a rapidly acceleration is not real time detected.

[7] B. Boehm and R. Turner. Observations on balancing discipline and agility. In *Proc. Agile Development Conference (ADC)*, pages 32–39. IEEE, 2003.

[8] M. Cohn. *Succeeding with agile: software development using Scrum*. Pearson Education, 2010.

[9] I. E. Commission et al. Functional safety of electrical/electronic/programmable electronic safety related systems. *IEC 61508*, 2000.

[10] G. Eleftherakis and A. J. Cowling. An agile formal development methodology. In *Proc. 1st South-East European Workshop on Formal Methods*, pages 36–47, 2003.

[11] C. H. Fleming and N. G. Leveson. Including safety during early development phases of future air traffic management concepts. *Eleventh USA/Europe Air Traffic Management Research and Development Seminar (ATM)*, 2015.

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

[13] X. Ge, R. F. Paige, J. McDermid, et al. An iterative approach for development of safety-critical software and safety arguments. In *Agile Conference (AGILE), 2010*, pages 35–43. IEEE, 2010.

[14] C. Ghezzi, C. Menghi, A. M. Sharifloo, and P. Spoletini. On requirements verification for model refinements. In *Requirements Engineering Conference (RE), 21st IEEE International*, pages 62–71. IEEE, 2013.

[15] T. L. Hardy. *Essential Questions in System Safety: A Guide for Safety Decision Makers*. AuthorHouse, 2010.

[16] G. J. Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[17] T. Lauritsen and T. Stålhane. Safety methods in software process improvement. In *Software Process Improvement*, pages 95–105. Springer, 2005.

[18] N. Leveson. *Engineering a safer world: Systems thinking applied to safety*. Mit Press, 2011.

[19] J. Madison. Agile architecture interactions. *IEEE, Software*, 27(2):41–48, 2010.

[20] K. S. Rubin. *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley, 2012.

[21] S. Shafiq and N. M. Minhas. Integrating formal methods in XP-a conceptual solution. *Journal of Software Engineering and Applications*, 2014.

[22] T. Stålhane and T. Myklebust. Agile safety analysis-FMEA applied to user stories. The 16th International Conference on Agile Software Development (XP), Helsinki, 2015.

[23] T. Stålhane, T. Myklebust, and G. Hanssen. The application of Safe Scrum to IEC 61508 certifiable software. *Haettu*, 1:2014, 2012.

[24] I. Standard. *ISO26262: Road vehicles-Functional safety*. 2011.

[25] D. Turk, R. France, and B. Rumpe. Limitations of agile software processes. 2002.

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

[27] H. M. Works, U. T. Logic, and S. U. a Test-Harness. Modex 2.1 user guide.