An Industrial Roadmap for Continuous Delivery of Software for Safety-critical Systems

Marc Zeller, Daniel Ratiu, Martin Rothfelder, Frank Buschmann

An Industrial Roadmap for Continuous Delivery of Software for Safety-critical Systems

Marc Zeller, Daniel Ratiu, Martin Rothfelder, Frank Buschmann
Siemens AG, Corporate Technology
81739 Munich, Germany
{forename.surname}@siemens.com

***Abstract*—Traditionally, promoted by the internet companies, continuous delivery is more and more appealing to industries which develop systems with safety-critical functions. Since safety-critical systems must meet regulatory requirements and require specific (re-)certification processes in addition to the normal development steps, enabling continuous delivery of software in safety-critical systems requires to solve specific challenges. In this paper, we describe relevant challenges for realizing continuous delivery in software-intensive safety-critical systems, and sketch a roadmap how to implement such a continuous delivery pipeline using an incremental approach.**

***Index Terms*—safety, agile, DevOps, continuous delivery**

I. Introduction

DevOps mixes the development and operations phases of a software product by promoting high frequency software releases which enable continuous innovation based on feedback from operations. DevOps uses continuous integration and test automation to build a pipeline from development to test and then to production (so-called *continuous delivery*). While companies are already implementing agile practices and continuous delivery in "non-critical" software development, safety-critical software is nowadays still developed using classical waterfall or V-model-based development processes. Safety-critical systems must meet regulatory requirements and shall comply to safety standards (such as IEC 61508). This requires (re-)certification processes for safety compliance after each change of the system.

However, also in the area of safety-critical systems, the need for accelerating the delivery of software is essential to reduce the time-to-market for new features and to respond faster to changing customer/market demands or technical concerns like deploying security patches. Hence, there is an increasing need to build continuous delivery pipelines also for software in safety-critical systems.

Working Hypothesis: As working hypothesis for this paper, we assume that an initial version of the system is certified and deployed in the field. We further assume that the updated functionality is exclusively realized in software.

R-Scrum [1] and SafeScrum [2] are existing approaches to develop safety-critical systems using agile methods, but do not show how to build a continuous delivery pipeline. Also [3] only presents challenges how to enable agile development of safety-critical systems in large organizations. First ideas to realize a continuous delivery pipeline for safety-critical software are outlined in [4] and [5]. In contrast to existing work, the objectives of this paper are twofold: (1) to describe relevant challenges within the safety engineering life-cycle for performing fast and incremental software changes/update in safety-critical systems, and (2) to sketch a roadmap to speed-up the development of safety-critical software-intensive systems from in industrial point of view using an incremental approach.

II. Background: Generic Safety Engineering Life-cycle

The engineering of safety-critical systems includes various aspects as described in safety standards. The goal of safety engineering is to identify failures that cause hazardous situations and to provide a sound argumentation that the system is sufficiently safe. This argumentation is based on evidence gathered during the system engineering and assessment process.

The first step in the safety engineering life-cycle is the **item definition**, in which the item (along with its purpose and functionality) considered by the safety engineering process is defined, and dependencies between the item and its environment are described. Based on a clear system definition, a **Hazard Analysis & Risk Assessment (HARA)** is performed. This analysis tries to identify potential hazards that can be caused by the system and to asses the associated risks. This step requires a very clear understanding about the context of the system and the potential interactions between the system and environment. The HARA is performed by domain experts. As the next step, a **system architecture** is defined and a **safety concept** is derived. The safety concept is defined as the specification of the safety requirements, their allocation to system elements, and their interactions necessary to achieve safety goals. Moreover, the potential causes and the cause-effect-relationships must be evaluated. Therefore, different **safety analysis techniques** are used that evaluate the risk that arises from potential failures that have been identified as causes for hazards. As the system development continues, the system architecture is refined in the form of a **software architecture**. In the same way, as the system is incrementally refined over the different development phases, the safety analyses of the refined development artifacts refine the safety concept accordingly. Afterwards, the specified system is realized and verified against the specification. The **verification** techniques (such as source code verification, unit testing, integration testing, etc.) are defined or recommended by safety standards. At the end

of the development a so-called **safety case** is compiled to argue that the system is safe. This safety argumentation spans over heterogeneous artifacts (e.g. hazards, requirements, code, safety analyses, tests, etc.) that need to be glued together in a consistent manner. During the entire development, experts from different engineering disciplines must work together. The resulting artifacts, processes, and tools are subject to **independent reviews and independent safety assessments** which eliminate eventual judgement biases of single experts. This is always a time-consuming process in parallel to the actual development. During **operation**, safety-critical systems must be monitored for potentially unsafe behavior. On the occurrence of unsafe behavior, analyses shall be performed to see if an action is required. Moreover, any change needs to be analyzed, answering the question whether the unsafe behavior is in expected bounds or the system needs to be fixed either due to a software bug (less likely) or due to a incomplete system level requirement. This analysis is referred to as the **Change Impact Analysis (CIA)**.

III. Challenges in Enabling Continuous Delivery for Safety-critical Systems

While continuous monitoring and measurement (often called supervision in the safety domain) is a standard design feature of safety-critical systems, continuous delivery is challenging in safety-related applications with strict regulatory requirements and safety guidelines. Since safety is a system-level property, the continuous delivery process must be lifted from software to system level and people from different engineering disciplines must be included in the continuous delivery pipeline. Moreover, additional development steps are required in safety-critical systems, such as the HARA, the safety analysis of the architecture, the safety augmentation, and the certification (see Fig. 1). These steps are required by all safety standards and must be performed within the delivery delivery pipeline. Thereby, the following challenges related to

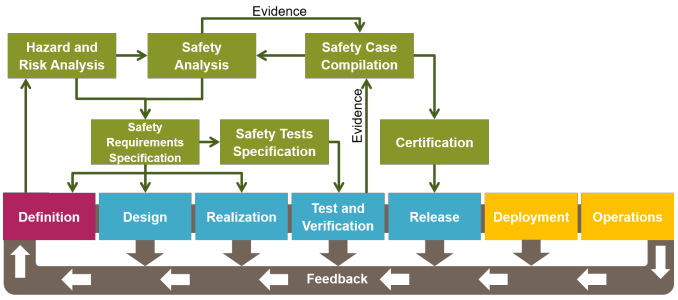

Fig. 1. Continuous delivery pipeline with additions for safety-critical systems development

the safety engineering activities need to be addressed:

a) Hazard Analysis & Risk Assessment: Today, the HARA is a manual process which requires the assessment of potential system hazards typically documented in spreadsheets. To enable continuous delivery for safety-critical systems, we need to speed-up the HARA process. This means, that we need automation for both the identification of hazards and the assessment of the risk associated to hazards. Moreover, it must be possible to identify new hazards and to adapt the risk associated with known hazards of the system.

b) Safety Analysis: The safety analysis techniques used today in industrial practice, such as *Fault Tree Analysis (FTA)* and *Failure Mode and Effect Analysis (FMEA)*, are performed manually by experts on system level. To enable continuous delivery, we need to speed-up the safety analysis process by increasing the level of automation. The changes of the software and their influence on the system safety must be reflected in the analysis automatically. Failure modes specific to highly integrated software-intensive systems such as failures due to feature interactions, emerging features, or not-wanted interactions must be integrated in the safety analysis. Moreover, the safety analyses must be linked to the system (or software) design so that changes to the system (or software) architecture can be synchronized with the safety analysis models.

c) Safety Tests: Since the predictable behavior of a system in the presence of faults is crucial to its safe operation, testing of safety-critical systems addresses questions such as: Are fault reactions correctly and effectively implemented? Is the timing of these reactions sufficient? Consequently, safety-critical systems require the testing of safety mechanisms that consider the fault models of the system components. Typically, tasks related to the testing of safety mechanisms are manually done by experts. In the context of a continuous delivery pipeline, the synthesis of tests for the specified failure mitigation mechanisms must be automated. Moreover, we need the possibility to inject faults into the productive system under test in virtual/real production environment without side-effects and to automatically execute the generated tests.

d) Safety Argumentation: Today, a safety case that describes the argumentation and references all the work products created during the safety life-cycle is manually captured in documents. Often, these work products are spread across various tools. The disparate information sources result in high accidental complexity and keeping the artifacts consistent is time-intensive and error-prone. Checking that the safety argumentation is complete and consistent with the configuration of the system, which is planned to be released, is expensive and mostly a manual process done through reviews. In order to build a continuous delivery pipeline for safety-critical systems, the creation and maintenance of the safety argumentation must be automated. Therefore, detailed traceability between safety arguments and evidences created during the development must be provided.

e) Orchestration of Different Engineering Disciplines: Since safety is a system-level property, the assessment of safety-critical software in terms of safety must be conducted on system level. Therefore, not only software engineers must be involved in the continuous delivery process, but also experts from other engineering disciplines as well as safety experts. Moreover, external assessors must be incorporated into the agile development process. In this way, at the end of each sprint/iteration, not only correct software but also the

necessary assets and documentation for the independent safety assessment and certification can be delivered.

f) Change Impact Analysis (CIA): CIA spans across heterogeneous artefacts at different abstraction levels and is today performed manually. Since safety experts need to have detailed knowledge about the system components and their interactions, the CIA is a time-consuming and error-prone activity. In context of continuous delivery, the impact of changes to the systems' safety must be automatically analyzed and prove that the change request has no influence in terms of safety. Therefore, we need to be able to determine, which activities of the safety engineering (such as the HARA or the safety analyses) must be updated.

IV. Roadmap Towards Continuous Delivery for Safety-critical Software

To automate the delivery process of safety-critical, software-intensive systems, we have to create a "*continuous (re-)certification machine*" that reduces certification efforts of an update. Therefore, we need to automate the activities of the safety engineering life-cycle and solve the challenges described in Sec. III.

In this section, we present a set of research questions guided by three different use case scenarios (Scenario 1 being the easiest, Scenario 3 the most difficult to implement).

Scenario 1: Bug-fixing in code: In this scenario, the change within the safety-critical system is represented by a bug-fix in the code. This can be for instance a bug-fix related to a security issue, which does not changes the functions of the system. After the bug is fixed, additional steps must be performed. First, the conformance of the code to the existing requirements must be ensured. Therefore, the existing tests need to be executed and passed successfully. To ensure the confidence required by the assigned Safety Integrity Level (SIL), the test coverage must be checked and if required, new tests must be defined and executed. Moreover, it must be ensured that the code complies with the coding guidelines (e.g. MISRA-C guidelines) and that the system is robust (e.g. by performing static analyses as required by safety standards). Finally, to determine the consequences of bug-fixing on the system safety, a code review of the updated system must be performed. In order to realize such a bug-fixing scenario in the context of continuous delivery, the above-mentioned steps need to be executed tool-supported and if possible automatically. In the context of test automation, a lot of prior work exists. However, today the evaluation of the consequences of bug-fixing in terms of a safety is performed manually. Therefore, the following research questions must be answered:

(1) How to prove that the artifacts of the safety engineering life-cycle do not need to be updated as a consequence of a bug-fix? We need to realize a deep integration between various artefacts (or models) which are relevant for the safety engineering [6]. A first step in this direction is to establish semantically rich and fine-grained traceability across development artifacts.

(2) Does a bug fix introduce new failure modes at software-level (e.g. loss of precision of arithmetic computations)? And how do these software-level failure modes propagate at system-level? If a bug-fix requires to update safety engineering artifacts (see 1), we need to determine how to adopt the safety engineering artifacts such as the system safety analysis accordingly. To automate this task, we can leverage model-based safety analysis techniques which are interlinked with system design artifacts [7], [8] to determine potential new failure modes on software level and automatically determine how new software-level failure modes propagate at system-level [9].

(3) Can the freedom of interference after the bug-fix still be guaranteed? In order to determine freedom of interference among safety functions, the features and their interaction or not-wanted interactions need to be represented by semantically rich models. Based on these models an automated analysis must determine whether a certain feature influences other ones or not.

Scenario 2: Change of software requirement: In this scenario, the change within the safety-critical system to be delivered originates from a software requirement change. An example of changed software requirement is for instance the replacement of a library with a newer version, the fixing of an inconsistency in software requirements, or the use of fixed-point arithmetic to speed-up processing time. The following steps must be undertaken *before* any change is implemented in the code: First, it must be ensured that new software requirements are a refinement of the system-level requirements (requirements verification). Second, the completeness and consistency of the software requirements must be checked in the presence of a change. Furthermore, it must be evaluated, if new software-level faults could be introduced by the updated requirements (e.g. loss of precision when arithmetic operations are performed). The following steps must be done *after* the change is implemented: The conformance of the code to the updated requirements must be ensured. Hence, test cases must be updated accordingly. Moreover, existing tests must be executed and passed successfully. To ensure the confidence required by the assigned SIL, test coverage must be checked and if needed, new tests must be added. Since a software requirement was changed, the traceability between the code and the corresponding software requirements must be re-established. Finally, a review of the changed code must be performed in order to determine the consequences of the new requirement on the system safety. In addition to the research questions from scenario 1, the following new research questions must be answered:

(4) Does the change result in new hazards at system-level? Detailed knowledge about the system, its environment, and the interfaces between system and environment is required to automate the HARA process and identify new hazards. A possible approach could be to simulate the system including its different sub-systems (software, hardware, etc.) in context of its environment to identify the effects of component failures on the behavior of the system.

(5) How to automatically verify the completeness and consistency of changed software requirements? To answer this

question, we need to model requirements at sufficient level of detail. However, verifying completeness and consistency of requirements is ongoing research in the requirements engineering community. Assessing completeness is especially important since incomplete (safety) requirements can lead to unmitigated hazards.

(6) How to automatically verify the refinement of low-level software requirements w.r.t. system-level requirements? In order to ensure the consistency and completeness of software requirements, we need to show that the new software requirements are a refinement of the existing system-level requirements. In addition to model requirements at sufficient level of detail, we need to link the requirements between the abstraction levels. Hence, approaches such as contract-based design may be applied to automate the verification.

Scenario 3: Change of system-level requirements: In this scenario, the change within the safety-critical system to be delivered originates from a change in the system-level requirements or a new system-level requirement. An example for such a scenario could be adding a new feature to the system. The following steps must be done *before* a change is implemented in the code: First, new hazards and failure modes induced by the modified / new system requirement(s) must be investigated. Second, the completeness and consistency of the system-level requirements in the presence of the change must be checked. Moreover, software requirements which satisfy the changed / new system-level requirements need to be derived. Thereby, it must be ensured that the new software requirements are a correct refinement of the system requirements. Afterwards, the completeness and consistency of the software requirements in the presence of the change must be checked. The following steps must be done *after* the change is implemented: The conformance of the code to the updated software requirements must be ensured. Hence, test cases must be updated accordingly. Moreover, existing tests must be executed to check if they pass successfully. To ensure the confidence required by the assigned SIL, the test coverage must be checked and if needed, new tests must be added. Since any change in the set of system-level requirements leads to an adaptation of the software requirements, the traceability between the code and the corresponding software requirements must be re-established. Finally, a review of the changed code must be performed in order to determine the consequences of the changed / new system-level requirement on the ssystem safety. Apart from the research questions in scenario 1 & 2, also the following research question must be answered:

(7) Does the modified/new system requirement(s) lead to new hazards and failure modes? Similar to (2) and (4), we must determine new hazards or failure modes which result from the modified/new system requirement. However, in this scenario this must be done on a system level, comprising both software and hardware related failures as well as their potential hazardous behaviour on system level. In contrast to (2) this also includes the identification of random failures in hardware. Therefore, a detailed machine-readable specification of the system itself (especially the hardware/software interfaces) and its environment are required.

(8) How to automatically perform checks w.r.t. consistency and completeness of the changed system-level requirements? This require semantically rich models to specify the requirements at system level as well as refinement links between abstraction levels. The current practice of "merely tracing" (textual) requirements is inadequate for automated checks. Therefore, future research must focus on modeling requirements in a formal way at sufficient level of detail and link the requirements between abstraction levels.

V. Summary and Future Work

Safety-critical systems must meet regulatory requirements and shall comply to safety standards. Moreover, the continuous delivery process of a new software version must be lifted from software to system level and people from different engineering disciplines must be involved. Thus, applying agile development and the concepts of continuous delivery in context of safety-critical, software-intensive systems requires to solve specific challenges. In this paper, we describe the challenges which need to be solved in order to realize continuous delivery of software in safety-critical systems. Moreover, we sketch future research directions to overcome the mentioned challenges using an incremental approach. Thereby, we start with the "most simple" scenario (bug-fixing in code) and name guiding future research questions, which must be answered to realize such a scenario. Further scenarios (change of software or system requirement) raise additional research questions which must be answered by future research.

Furthermore, due to the automation of the safety delivery process, the delivery pipeline itself becomes a subject to regulatory compliance and the qualification of the tools is necessary. Moreover, there is a need to integrate assessors as stakeholders into the continuous delivery process to fully leverage the benefits in regulated environments. These topics also require further research.

References

[1] B. Fitzgerald, K. Stol, R. O'Sullivan, and D. O'Brien, "Scaling agile methods to regulated environments: An industry case study," in *35th Int. Conf. on Software Engineering*, 2013.

[2] G. K. Hanssen, T. Stålhane, and T. Myklebust, *SafeScrum®-Agile Development of Safety-Critical Software*. Springer, 2018.

[3] J.-P. Steghöfer, E. Knauss, J. Horkoff, and R. Wohlrab, "Challenges of scaled agile for safety-critical systems," in *Product-Focused Software Process Improvement*, 2019, pp. 350–366.

[4] S. Vost and S. Wagner, "Keeping continuous deliveries safe," in *IEEE/ACM 39th Int. Conf. on Software Engineering Companion*, 2017.

[5] F. Warg, H. Blom, J. Borg, and R. Johansson, "Continuous deployment for dependable systems with continuous assurance cases," in *IEEE Int. Symposium on Software Reliability Engineering Workshops*, 2019.

[6] D. Ratiu, M. Zeller, and L. Kilian, "Safety.Lab: Model-based domain specific tooling for safety argumentation," in *Computer Safety, Reliability, and Security*, 2015, pp. 72–82.

[7] S. Sharvia, S. Kabir, M. Walker, and Y. Papadopoulos, "Model-based dependability analysis: State-of-the-art, challenges, and future outlook," in *Software Quality Assurance*, 2016, pp. 251 – 278.

[8] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software*, 2003, pp. 37–46.

[9] H. Jahanian, "Failure mode reasoning," in *4th Int. Conf. on System Reliability and Safety (ICSRS)*, 2019, pp. 295–303.