Rail to Digital automated up to autonomous train operation

D29.2 – Investigation of solution approaches for DevOps and architectural evolvability from other domains

Due date of deliverable: 2023-12-01

Actual submission date: 2023-12-19

Leader/Responsible of this Deliverable: Deutsch, D.; SMO

Reviewed: Y

Document status		
Revision	Date	Description
01	2023-12-19	First issue

Project funded from the European Union's Horizon Europe research and innovation programme		
Dissemination Level		
PU	Public	X
SEN	Sensitiv – limited under the conditions of the Grant Agreement	

Start date: 2022-12-01 Duration: 42 months

ACKNOWLEDGEMENTS

This project has received funding from the Europe's Rail Joint Undertaking (ERJU) under the Grant Agreement no. 101102001. The JU receives support from the European Union's Horizon Europe research and innovation programme and the Europe's Rail JU members other than the Union.

REPORT CONTRIBUTORS

Name	Company	Details of Contribution
Arrizabalaga, Saioa	CEIT	DevOps, 2nd opinion, review
Figueroa, Santiago	CEIT	DevOps, 2nd opinion, review
Roelle, Harald	SMO	2nd opinion, review
Deutsch, Dominik	SMO	Architectural evolvability, 2nd opinion, review
Oertel, Norbert	SMO	2nd opinion, review

EXECUTIVE SUMMARY

Software needs to be adapted over its lifetime for several reasons, as discussed in the previous deliverable D29.1 [1]. To enable and support this and to avoid software erosion, an *evolvable* software architecture and appropriate processes are necessary. In this deliverable the working group investigates concrete solutions and approaches for DevOps and architectural evolvability from other domains as some of these might also be applied or adapted for the railway domain.

First, the working group focuses on DevOps as methodology that brings together practice of software development and operations to improve collaboration, communication, and efficiency throughout the entire software development process. Such considerations as the nature of systems, risk tolerance, regulatory compliance, and scale and complexity can vary significantly between Information Technology (IT) and Operational Technology (OT) environments due to domain-specific requirements. For example, just by considering the impact of a cyber-attack on a safety-critical environment, both safety and security constitute critical factors that differ between IT and OT DevOps approaches. For this reason, the working group analyzes the state of the art of DevOps on OT environments, including sectors like automotive, avionics and railway to analyze the main approaches for OT environments from a safety and security perspective. They also analyze how DevOps approaches OT environments from the perspective of leading safety (IEC 61508) and security standards (ISA/IEC 62443-4-1). Finally, a practical analysis is performed, where based on a selected user story from deliverable D29.1 [1] an attempt is made to approximate the set of associated DevOps stages and to characterize required tooling for each stage. Next steps will include to cover more user stories from D29.1 [1] by the DevOps methodology and to bring it closer to the railway domain considering the different standards.

In the second part of this deliverable, the working group summarizes and structures the results of a literature research on architectural evolvability. The focus is on the domains automotive, avionics and industrial automation. However, literature from other domains (e.g., enterprise IT) as well as domain-unspecific literature is also considered. First, some characteristics and current architectures of the mentioned domains are described as their general set-ups and conditions, some of which differ from the railway domain, must be considered when evaluating the found solutions. Subsequently, the solutions and approaches for evolvable software architectures found in the literature are listed and mapped to the non-functional properties and user stories from the previous deliverable D29.1 [1], which they primarily address. For the listing, the solution approaches are grouped into architectural patterns, techniques and methodologies, concepts and principles, concrete solutions and approaches, and metrics. Finally, a potential applicability within the railway domain is outlined. Next steps will be to analyze and evaluate the found solution approaches from the other domains in more detail with the aim of creating a catalogue of solution approaches for architectural evolvability that are applicable within the railway domain.

ABBREVIATIONS AND ACRONYMS

AAL	Architecture Analysis Language
ADL	Architecture Description Language
AFDX	Advanced Full Duplex Network
ANSI	American National Standards Institute
AI/ML	Artificial Intelligence/Machine Learning
ALMA	Architecture-Level Modifiability Analysis
AP	Adaptive Platform
API	Application Programming Interface
ARINC	Aeronautical Radio Incorporated
AUTOSAR	Automotive Open System Architecture
BPMN	Business Process Model and Notation
CAAS	Common Avionics Architecture System
CAN	Controller Area Network
CBSE	Component-Based Software Engineering
COTS	Commercial off-the-shelf
CP	Classic Platform
CPPS	Cyber-Physical Production Systems
CPS	Cyber-Physical System
DDS	Data Distribution Services
DEMO	Design and Engineering Methodology for Organizations
DevOps	Development and Operations (Methodology)
DoD	Department of Defense
DSL/DSML	Domain-Specific (Modeling) Language
EAS	Evolvable Assembly Systems
ECU	Electronic Control Unit
EDGSS	Emergency Diesel Generator Startup Sequencer
EPS	Evolvable Production Systems
FMS	Flexible Manufacturing System
FPGA	Field Programmable Gate Arrays
HAL	Hardware Abstraction Layer
HMS	Holonic Manufacturing System
HW	Hardware
ICS	Industrial Control System

ID	Identifier
ICT	Information and Communications Technology
IEC	International Electrotechnical Commission
IIoT	Industrial Internet of Things
IMA	Integrated Modular Avionics
IPC	Industrial PC
ISA	International Society of Automation
ISO	International Organization for Standardization
IT	Information Technology
JADE	Java Agent Development Environment
LAN	Local Area Network
LIN	Local Interconnect Network
MAS	Multi-Agent Systems
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDSD	Model-Driven Software Development
MILS	Multiple Independent Levels of Security and Safety
MOSA	Modular Open Systems Approach
NFP	Non-functional property
NFR	Non-functional requirement
OCI	Open Container Initiative
OEM	Original Equipment Manufacturer
OPC-UA	Open Platform Communication-Unified Architecture
OS	Operating System
OSA	Open System Architecture
OT	Operational Technology
OWASP	Open Web Application Security Project
PLC	Programmable Logic Controller
PLE	Product Line Engineering
QDSA	Quality-Driven Software Architecture
QoS	Quality of Service
RAAM	Recovering Architectural Assumptions Method
RACE	Robust and Reliant Automotive Computing Environment
RMS	Reconfigurable Manufacturing System

RTOS	Real-Time Operating System
R2DATO	Rail to Digital automated up to autonomous train operation
SAFe	Scale Agile Framework
SAOL	System Architecture Optimization Language
SBOM	Software Bill of Material
SDK	Software Development Kit
SOA	Service-Oriented Architecture
SOME/IP	Scalable Service-Oriented Middleware over Internet Protocol
SW	Software
S2C	Security Standard Compliant
TCP	Transport Control Protocol
TSP	Time and Space Partitioning
TTE	Time Triggered Ethernet
UAM	Urban Air Mobility
VM	Virtual Machine

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The *erosion* of software architecture leads to problems like the increase of cost for changes and extensions, the decrease of non-functional properties (NFPs), and the degradation of software quality [2]. While in some areas for individual software projects or new products everything can be developed from scratch, this is usually not possible in many other areas due to time and cost reasons. To be able to keep pace with technology development and remain competitive, for example, it may be necessary to (continuously) adapt an existing software architecture over time. But change will become more difficult over time unless the software system is designed to be evolvable [3].

Software architecture *evolvability* describes the software architecture's capability to accommodate changes [2]. It is important to note that the described capability cannot be achieved by focusing on single architectural aspects. Software evolvability is a *multifaceted quality* attribute [4]. But according to [5] "most studies focus on particular quality attributes such as adaptability, and do not cover the wide spectrum of evolvability subcharacteristics" and only "few studies explicitly address software evolvability". In this work multiple "sub characteristics" or "quality attributes", specifically the non-functional properties collected in D29.1 [1], are addressed.

The base for evolvability is a "good" architecture.

A "good" software architecture can be evaluated qualitatively (e.g., reviews) or quantitatively (e.g., metering) based on *(non-functional) properties* or *metrics* and achieved by applying appropriate *methodologies*, *concepts*, *design principles* or *patterns*.

But evolvability is not only about the software architecture. It needs to be addressed over the complete software lifecycle [5]. The *DevOps* approach can contribute to this. Deliverable D29.1 [1] already defines the concept and applicability of DevOps. This work proposes to study the DevOps paradigm in depth from the perspective of industrial environments (particularly in the railway domain) with special emphasis on both cybersecurity and safety.

The DevOps market has observed significant demand for agility in development, testing, and Information Technology (IT) operations. DevOps tools tightly integrate the design & development of applications with the deployment and delivery lifecycle, enabling enterprises to efficiently manage their IT resources. This also leads to reduced capital expenditure and increased market competitiveness by reducing software delivery cycles. In addition, the demand for automated defect detection and standardization of development stack is to create new market opportunities for development & operations tools (Figure 1).

The increasing importance of software and rising level of connectivity of Operational Technology (OT) products (e.g., safety-critical) such as vehicles, require continuously improving and adding functionality. However, it requires more guidance and a more structured approach for software assurance is needed through further standardization, and adoption of improved procedures and guidance to support certification of safety-critical assets. This is where DevOps emerges as a paradigm that promises to successfully impact all stages of the software lifecycle. However, in the OT sectors, security and safety are the cornerstones. OT software (particularly in safety-critical environments) demands a comprehensive security and safety case that usually refers to a set of documents with arguments and evidence showing the product’s security and safety, i.e., the security and safety artifacts. Hence, there are rigorous standards related to the design, implementation, and deployment of the software lifecycle, such as International Society of Automation/International Electrotechnical Commission (ISA/IEC) 62443-4-1, IEC 61508, and ISO 26262. Therefore, unlike the IT world in the context of OT, the value of DevOps is tied to security and safety concepts.

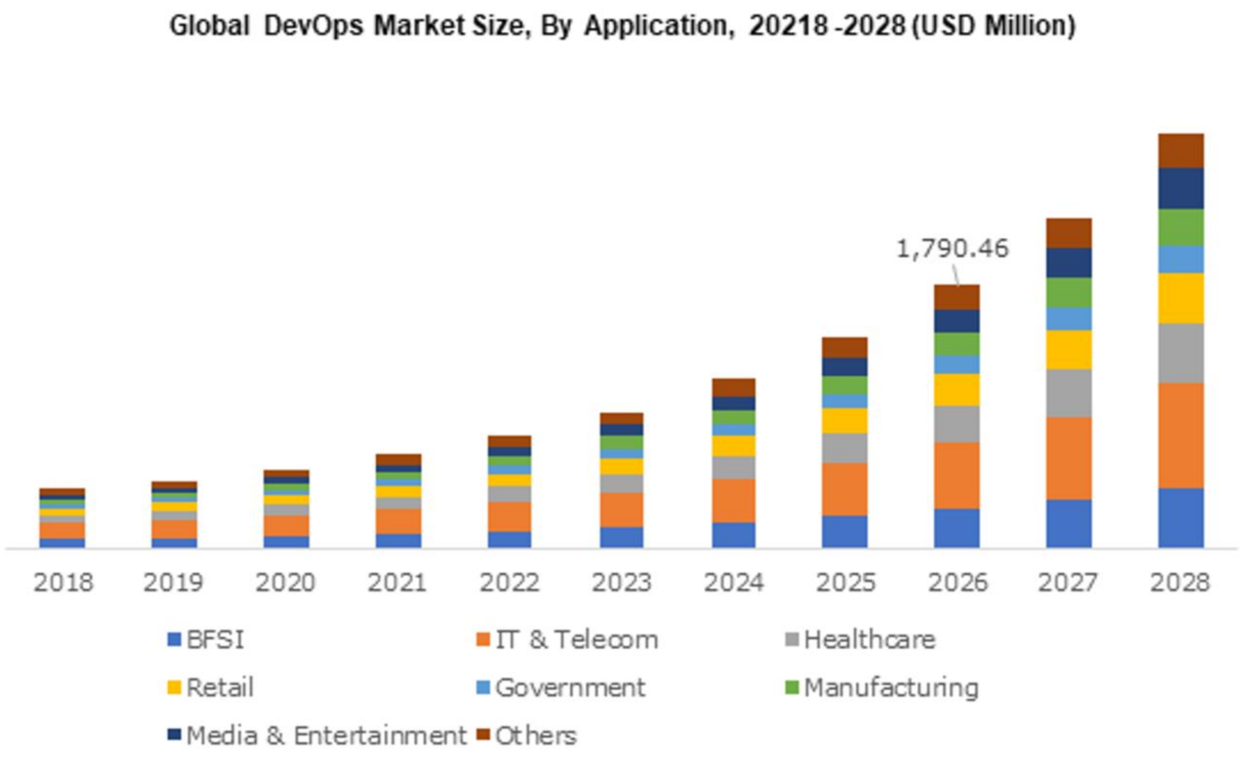

Figure 1. Global DevOps market size by application [6].

The concrete solutions and approaches for a good and evolvable software architecture as well as for DevOps processes depend heavily on the environment and required conditions (e.g., real-time, or safety-critical) and on the respective domain (e.g., enterprise IT, automotive, avionics or industrial automation). But many approaches can also be applied to different domains. So, the railway domain can be inspired by approaches from domains with fundamentally other conditions (such as enterprise IT) and adopt or transfer solutions from domains with similar requirements for real-time and safety (such as automotive, avionics or industrial automation).

2 DEVOPS

In this section we intend to start with a brief study that illustrates the usability of DevOps in different OT sectors. The value of security and safety in the OT context leads to the analysis of the security and safety in DevOps Lifecycle. Finally, a mapping of user stories defined in deliverable D29.1 [1] is developed around a secure DevOps approach.

2.1 DEVOPS IN DIFFERENT OPERATIONAL TECHNOLOGY APPLICATION DOMAINS

1) Automotive: There are contributions from automotive environments on DevOps. For example, a first approach specifies how the automotive (software) development lifecycle is expected to evolve towards a DevOps-oriented development process [7]. The authors have proposed a DevOps framework covering the full vehicle lifecycle ranging from development, to production, to operation. A second approach introduces the SafeOps concept that leverages the DevOps principles automation, feature-driven development, and monitoring during operations to fulfill the requirements of the ISO 26262 when iteratively extending and improving safety-critical products [8].
2) Avionics: There are also contributions from avionics environments on DevOps. For example, a first approach indicates that for Urban Air Mobility (UAM) manufacturers to succeed in the marketplace, it is necessary to leverage DevOps development practices and hardware virtualization to reduce overall lifecycle costs and, in turn, ensure the ability to meet the customer’s vehicle safety and cost requirements [9]. A second approach describes a study that resulted in a set of software architecture principles intended to help with sustainability-driven design and monitoring. The framework given focuses on the aviation industry in particular [10].
3) Department of Defense (DoD): DoD has established DevSecOps capabilities to deliver applications rapidly and in a secure manner, increasing the warfighters competitive advantage, bake-in and enforce cybersecurity functions and policy from inception through operations, enhance enterprise visibility of development activities and reduce accreditation timelines, ensure seamless application portability across enterprise, Cloud and disconnected, intermittent and classified environments and drive DoD transformation to Agile and Lean Software Development and Delivery [11]. As part of this process, several technologies, and tools from different environments (e.g., cloud) have been integrated. For instance, the use of Kubernetes at DoD reduced the software deployment effort from eight months to one week [12].
4) Railway: There are some shy approaches to DevOps integration in railway both from the business perspective and from the scientific literature. On the one hand, [13] refers to the application of DevOps to the propulsion system development process, along with advanced physical modelling techniques and innovative uses of AI/ML for automation. On the other hand, the scientific paper approaches DevOps from the perspective of Design-Operation Continuum methods to provide solutions in order to have a more efficient process which guarantees that (1) software updates are

performed safely and securely, (2) most of the faults are detected in the design phase before the software is deployed in the CPS and (3) problems that can emerge in operation can be reproduced in development in order to analyze and propose potential solutions [14]. The use case where the proposed taxonomic review is put into practice is Bombardier Transportation [14]. Although this use case illustrates a DevOps approach in railway environments, it is currently outdated.

2.2 SECURITY AND SAFETY IN DEVOPS LIFECYCLE OF OPERATIONAL TECHNOLOGY ENVIRONMENT

From the previous sections it can be summarized that DevOps is a technology that is progressively penetrating operational technology environments, and at the same time, its adoption is intrinsically tied to security. Figure 2 establishes a relationship between two of the most important safety and security standards: ISA/IEC 62443-4-1 and IEC 61508. Although they are not the only security and safety standards, they are two of the most widely known, hence the approach of studying them. In addition, it proposes a mapping of these two standards onto DevOps as a framework. Currently there is a defined DevOps standard (ISO/IEC 32675), but it is highly linked to IT environments. For this reason, the potential relationship from the OT point of view is considered on discontinuous lines.

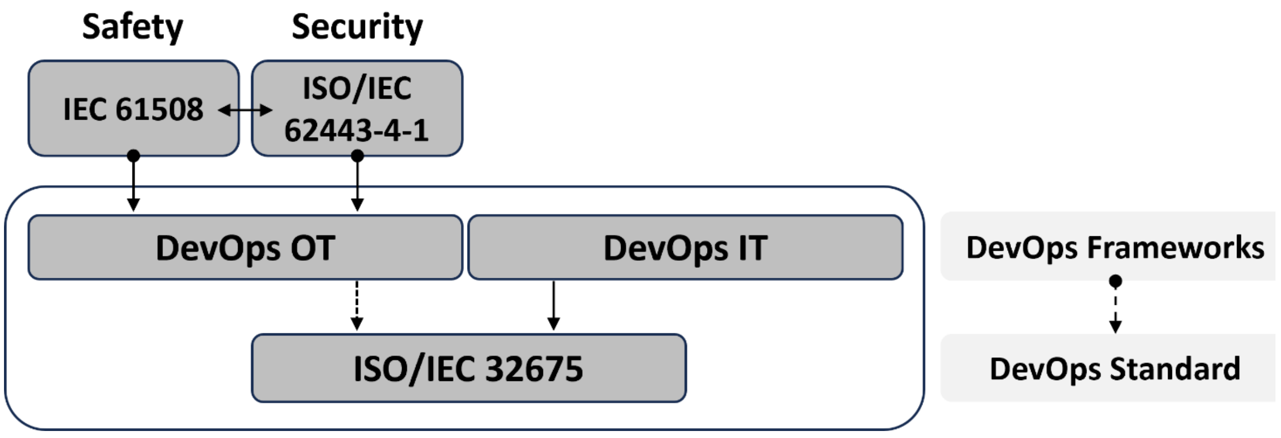

Figure 2. Framework to map security and safety standards over DevOps.

2.2.1 Comparison of IEC 61508 and ISA/IEC 62443-4-1

With the appearance of malware and nation state attacks on Industrial Control Systems (ICS), such as the Stuxnet (2010), Industroyer (2016) and TRITON (2017) attacks, safety system assets become targets [15]. More and more safety equipment OEMs (Original Equipment Manufacturer) are seeking to certify their products to both IEC 61508 Functional Safety requirements as well as ISA/IEC 62443 Cybersecurity requirements [15]. Development requirements concentrate on processes to ensure a good understanding of what it is going to be built, how it is going to be built, and that it was built correctly. IEC 61508 and ISA/IEC 62443-4-1 both have development process requirements. Furthermore, these requirements overlap a great deal, so separate assessment efforts would mean repeating assessment of common requirements. By identifying what process requirements are in common between IEC 61508 and ISA/IEC 62443-4-1 and showing that the IEC 61508 process requirements meet the ISA/IEC 62443-4-1 process requirements, the cost of developing procedures, and assessing procedures for compliance, can be reduced [15].

2.2.2 Security on DevOps from ISA/IEC 62443-4-1 perspective

Several studies demonstrate the integration of an industrial security standard such as ISA/IEC 62443-4-1 in DevOps environments. The first approach suggested a new way to achieve continuous and secure development in security domains, specifically in industrial and automation control systems [16]. To do so, the authors proposed a model-based approach, which consisted of merging visual representations of security norms and process models, in particular the Scale Agile Framework (SAFe) and the ISA/IEC 62443-4-1 standard, resulting in a Business Process Model and Notation (BPMN) model [17].

A second approach proposes a concept for a structured and systematic integration of security activities based on ISA/IEC 62443-4-1 standard into a DevOps pipeline. To achieve this, the security requirements, as described in the ISA/IEC 62443-4-1 standard, were mapped into a simple DevOps pipeline specification [18]. The work maps, in detail, the ISA/IEC 62443-4-1 security flows into the stages of the DevOps framework, in a similar way to that proposed by [17].

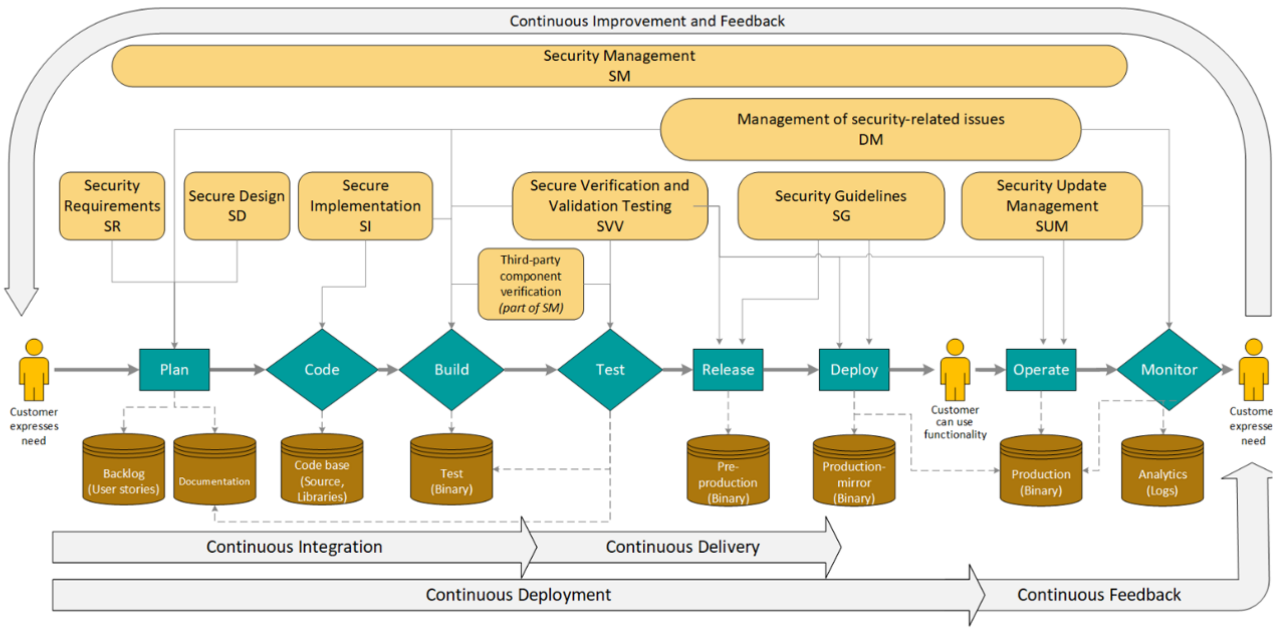

Figure 3. Security standard compliant DevOps pipeline for the IEC 62443-4-1 standard [19].

A third approach consists of integrating an instance of the ISA/IEC 62443-4-1 standard into pipelines [19]. Figure 3 shows the integration of the Security Standard Compliant DevOps Pipeline for the ISA/IEC 62443-4-1. Diagram shows the ISA/IEC 62443-4-1 standard practices (in yellow), the DevOps stages (in green). Solid vertical arrows depict in which

DevOps stage an ISA/IEC 62443-4-1 practice security activity can take place. Standard security activities impact several repositories (in brown) like backlog, code base and test, pre-production, and production environments. In addition, security standards demand an explicit repository for documentation and logs maintenance. Continuous practices (gray arrows) describe the flows to which the security activities also apply [15]. The main artefact of this approach is the Security Standard Compliant S2C DevOps Pipeline Specification. The methodology was tested in the context of a large industrial company that operates in the ICS market, providing that the automation extent of this standard is 31% *Complete*, it means that 31% of the ISA/IEC 62443-4-1 requirements can be fully automated. 38% of the ISA/IEC 62443-4-1 requirements are manual tasks that must be executed by a human expert, while the remainder has the potential to be at least partially automated with future tools and techniques [19].

A fourth approach integrates security into agile software development in strongly regulated industries, complexity increases not only when scaling agile practices but also when aiming for compliance with security standards. For that purpose, the authors present the framework S2C-SAFe and its evaluation by agile and security experts within Siemens' large-scale project ecosystem. They discuss benefits and limitations as well as challenges from a practitioners' perspective. The overall aim of their work is to improve product development lifecycle by integrating requirements of ISA/IEC 62443-4-1 into Scaled Agile Framework (SAFe), resulting in the "Security Standard Compliant Scaled Agile Framework" (S2C-SAFe) [20].

A fifth approach aims to investigate the evidence and identify its dependencies to develop and design an artefact model for DevSecOps. This artefact model has the possibility to measure security compliance with the ISA/IEC 62443-4-1 standard to ensure traceability in DevOps pipeline and evaluate the usability of it. This research provides the practitioners' understanding of the usability of the artefact model in the industry to meet the secure software development product lifecycle requirements according to the ISA/IEC 62443-4-1 standard. The results demonstrated the evidence of assessing the security compliance for DevSecOps workflow in DevOps pipeline [21].

2.2.3 Safety on DevOps from IEC 61508 perspective

Several studies demonstrate the integration of an industrial safety standard such as IEC 61508 in DevOps environments. However, the starting point of the relationship was not directly DevOps, which is a relatively recent concept, but rather agile development of safety critical software. SafeScrum [22], is a framework based on the Scrum process framework for incremental and iterative development as shown in Figure 4. To be compatible with requirements found in safety standards, in particular IEC 61508, SafeScrum proposes additional activities and roles. Requirements are kept in the product backlog in the form of user stories, and in SafeScrum, functional (not safety related) and safety-related requirements are kept separately. Simply put, functional user stories come from the users and safety-stories come from preliminary safety analyses. If a user story is presumed to be related to a safety story, a reference is inserted. Development is done in sprints, which are short and repeated work iterations. Each sprint starts with a sprint planning meeting where stories from the product backlog are prioritized, selected, and broken down into solution ideas and added to the sprint backlog. Development is done by a fixed team which has a short status meeting (known as the scrum) regularly, maybe every day, to share progress, plans and discuss any problems. Development of software should be done according to the principles of test-driven development, which also assures high test coverage and documentation of testing. Considering DevOps was built on the principles of agile practices, but extended them to include operations and automation, an agile safety case approach to have a DevOps process in place to ensure quick but safe patching (short- term response) has been implemented to satisfy both safety and security requirements when developing and operating autonomous vehicles [23]. A third approach demonstrates relation IEC 61508 - DevOps from a design assurance (IEC-61508 Compliant V&V Workflow), runtime safety and security as a fundamental aspect of the dependable DevOps continuum process. This work performs verification of an Emergency Diesel Generator Startup Sequencer (EDGSS) implemented on a Field Programmable Gate Arrays (FPGA) overlay architecture using model-based verification techniques [24].

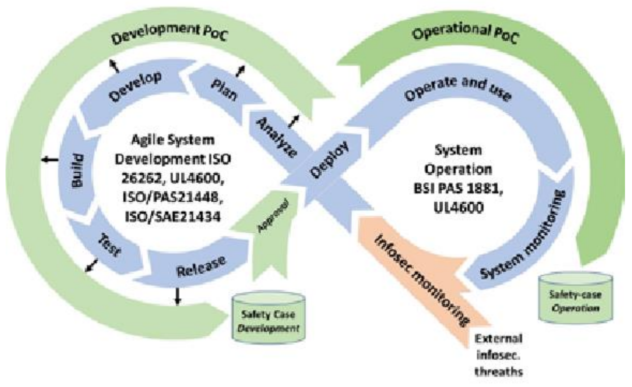

Figure 4. The SafeScrum process with a DevOps approach [22].

2.3 PRACTICAL APPROACH OF DEVOPS OVER RAILWAY DOMAIN

This section aims to give a practical view of how to approach DevOps in the railway context considering the gap identified in the State-of-the-Art study performed in Section 2.1. Considering that the positions of the ISA/IEC 62443-4-1 and IEC 61508 standards are not so far apart, we proceed to select from the previous work the most convincing approach, i.e., that the reference that aligned DevOps closer to ISA/IEC 62443-4-1 [19].

As shown in Figure 3, the first stage of the procedure is constituted by the definition of the product backlog, which refers to a prioritized list of functionalities which a product should contain. Therefore, the set of user stories defined in deliverable D29.1 [1] will be used as the product backlog [1]. To simplify the procedure in this deliverable, which will be extended in future deliverables, the practical approach to be evidenced consists of selecting a user story, narrowing its functionality, and moving it through the entire flow of Figure 3, providing the set of tools that in the literature refer to these stages. Table 1 above shows the selected user story.

ID	Actor	User Story	Driver 1	Driver 2
10007	Train Manufacturer	As a train manufacturer, I want to generate a security release of a train software in minimum time, so that train functionality is not changed, and re-homologation is not required.	Changeability	Verifiability

Table 1. User story selected [1].

The functional complexity evidenced by the second part of the user story, i.e., "so that train functionality is not changed, and re-homologation is not required" will be analyzed in future deliverables, hence the user story approach can be summarized in Table 2:

ID	Actor	User Story	Driver 1	Driver 2
10007	Train Manufacturer	As a train manufacturer, I want to generate a security release of a train software in minimum time.	Changeability	Verifiability

Table 2. User story abbreviated [1].

2.3.1 Context

There is a Software Bill of Material (SBOM) registry (e.g., CycloneDX data format) that integrates information related to versions, licenses, libraries, dependencies, author name, distributor name, open-source component classification of all development and infrastructure (e.g., K8s version, firmware version, etc.) lifecycle (Figure 5). Comprehensive, end-to-end SBOM management reduces risk and increases transparency in software supply chains.

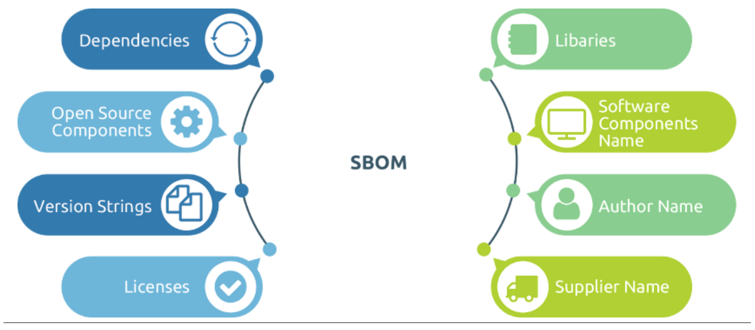

Figure 5. Definition of Software Bill of Materials (SBOM).

A vulnerability is identified in one of the dependencies used in the production application. It is necessary to rebuild the application and perform the delivery in the shortest possible time, updating in each case the SBOM. The following requirements are assumed:

1. Delivery is made on industrial railway devices (e.g., Ruggedcom VPE1400 [25]) that provides a virtualized environment to run a guest Linux operating system (e.g., RUGGEDCOM ROX II, a Rugged Operating System on Linux) and third-party applications.
2. GitHub is used as DevOps platform.

The following table (Table 3) relates the Figure 3 DevOps stages to different activities and purposes for which the user stories will be matched throughout the lifecycle. A set of open-source tools that help to materialize the integration are included.

DevOps Stage	Stage Definition	Repository
Plan	Based on the customer need, during this stage, requirements are represented as user stories. A repository that provides compliance evidence is the Backlog.	Backlog: The product backlog is a dedicated space for defining and prioritizing work a team will take on now and into the future. Jira Software ([26]) is a tool example that can be used for backlog purposes.
Plan	Based on the customer's need, other kinds of information (e.g., threat modeling and product security context). Other repository that provides compliance evidence is the Documentation repository.	Documentation: Continuous documentation can be provided by Readme files or Wikis. GitHub ([27]) is a tool example that can be used for documentation purposes.
Code	DevOps establishes that not only functionalities can be coded, but also infrastructure and environment configurations. Automation is possible for static code analysis.	DevOps recommends that functionalities and configurations should be stored in the same repository and integrated into the version control tool (e.g., GitHub). CodeCov ([28]) is a Code Quality tool example that generate and analyze Code Coverage reports.
Build	During this stage, the teams (not only developers) commit their code. The continuous integration tool triggers different security testing tools.	Compile and/or link is related with specific Framework (e.g., Java [29], .NET [30], Python [31]).
Build	During Build, security requirements testing can be done with development frameworks for behavior-driven development and unit testing.	Dependency vulnerability checking; scans your pull requests for dependency changes and will raise an error if any new dependencies have existing vulnerabilities, using, for instance, Dependency Review tool ([32]).
Test	In the test environment, automated and user acceptance tests can occur. If testing	Open Web Application Security Project (OWASP) ZAP Full Scan ([33]) is an example

DevOps Stage	Stage Definition	Repository
	is successful, code is merged, and the application is ready for testing. If testing is not successful, commit is not merged, and results of testing tools need to be synchronized with the security issues tracking tool to comply with the ISA/IEC 62443-4-1 standard. To achieve compliance, testing documentation should exist and be in the documentation repository. ISA/IEC 62443-4-1 standard describes four groups of testing: security requirements testing, threat mitigation testing, vulnerability testing and penetration testing.	of testing tool that runs the ZAP spider against the specified target (by default with no time limit) followed by an optional ajax spider scan and then a full active scan before reporting the results. The alerts will be maintained as a GitHub issue in the corresponding repository.
Test	In the test environment, automated and user acceptance tests can occur including behavior-driven development and unit testing.	Automated Tests tools run automated tests included in your code using framework-specific tools like pytest, dotnet test, etc for Framework specific (e.g., Java, .NET, Python).
Test	From the ISA/IEC 62443-4-1 standard point of view, this is relevant since it allows to maintain evidence of all security measures, security issues, licenses used and so on.	SBOM allows scanning the workspace directory and uploading SBOM artifacts. Anchore ([34]) is a SBOM tool example.
Release and deploy	During these phases, the functionalities that implement the customer's need are made available for customers. The release stage refers to Alfa/Beta releases and stored into a pre-production repository and during the Deploy stage to the production environment. Releases allow to gather early feedback about the implemented security measures. During these stages, penetration testing is performed with tools that partially automate the activities. Tools run tests, identify vulnerabilities, and a team member manually analyzes and tries to exploit them.	Considering the requirements assumed there are two very useful tools at this stage: ROXupgrade ([35]) and ROXflash ([35]).
Operate and Monitor	The product is available for Customer use. Security monitoring, security testing and compliance checks are highly automated. Monitoring activities are part of	Datadog ([36]) is an example of monitoring and observability tool that integrates an automatic architecture-mapping, monitoring, and troubleshooting all in one screen for

DevOps Stage	Stage Definition	Repository
	Maintenance and artifacts belong to the Analytics repository.	your streaming data pipelines to resolve errors quickly and avoid costly outages.

Table 3. User story mapping.

3 SOFTWARE ARCHITECTURE EVOLVABILITY

As discussed in deliverable D29.1 [1] as well as motivated in the introduction, there is a necessity for software architecture evolvability within the railway domain. In this chapter, the working group summarizes and structures the results of a *literature research* across other domains. For one thing characteristics and current architectures of these domains are described (see section 3.1), for another thing solution approaches for evolvable software architectures found in the literature are categorized and listed (see section 3.2).

The results will serve as a basis for deeper analyses and evaluation of selected approaches regarding their potential applicability within the railway domain in the next step (D29.3).

3.1 CURRENT ARCHITECTURES AND CHARACTERISTICS OF OTHER DOMAINS

The general set-up and conditions differ in part between the railway domain and the other domains examined. This must be considered when evaluating the found solutions. For this purpose, some central characteristics, properties, and architectures of other domains are described below. In particular, the domains automotive (see subsection 3.1.1), avionics (see subsection 3.1.2) and industrial automation (see subsection 3.1.3) were examined. However, other domains (especially non-safety-related IT systems) were also considered (see subsection 3.1.4).

3.1.1 Automotive

The automotive sector is characterized by its large number of electronic control units (ECU), some of which are purpose-built [37] [38]. There are also heterogeneous networking technologies and bus systems (e.g., Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay or Ethernet), mainly static communication paths [38] and a signal-based development [39] [40]. Functions and data are distributed over the in-vehicle network and the ECUs and there is no unified way to access data [37] [40]. Similarities with the railway domain are that the development is V-model-based as well as they also face issues and challenges regarding [38] homologation or certification [37] [39] and security and safety [39]. There is a trend for centralization [41] leading to a smaller number of processing units. Another trend is the inclusion of suppliers and third-party contributors within development and test procedures [42].

Standardization within the automotive industry is driven by the AUTOSAR (Automotive Open System Architecture) development partnership aiming for standardized basic system functions and functional interfaces and an open E/E system architecture [43]. The ubiquitous software platform in this industry is the AUTOSAR Classic Platform. It is “used for deeply embedded systems and application software with high requirements for predictability, safety, security and responsiveness” [43]. The architecture distinguishes at the highest level of abstraction between the three software layers application, runtime environment and basic software [43]. AUTOSAR provides an architecture to develop software components independent from specific base-hardware and operating system [44] and defines the language for designing and configuring automotive software architectures [45]. According to

[46], resolving connectors at design time via generators has enabled component-based software engineering (CBSE) for distributed control systems in the automotive domain. One concept of AUTOSAR is the definition of distributed embedded applications independently from a concrete deployment and the allocation to computing nodes in a later engineering step [46].

One disadvantage of current automotive software architectures is the mainly static communication so that changes in applications often require a change in the “communication matrix” that describes statically bound data channels [42]. To overcome this, OEMs steadily include more IP-based communication technologies, which support a better separation of software and hardware [42].

The next generation architecture is the AUTOSAR Adaptive Platform [47]. It consists of functional clusters grouped into services, where functional clusters must have at least one instance per (virtual) machine while services may be distributed in the in-car network [47]. In comparison to the Classic Platform, the Adaptive Platform dynamically links services and clients during runtime [47]. It is a solution for more performant ECUs and is designed to meet the requirements of highly automated vehicles [43]. For communication it supports DDS (Data Distribution Services) [37] and SOME/IP (Scalable Service-Oriented Middleware over IP) [37] [39], based on Ethernet-based topologies [38].

3.1.2 Avionics

A prominent characterization within the avionics sector is the time and space partitioning, i.e. partitioning of computation (time) and memory (space) or also of access or the backplane [48] [49]. They also rely heavily on the use of commercial off-the-shelf (COTS) components [50] [51] [52] as well as on standards like ARINC (Aeronautical Radio Incorporated) [53] [51] [48] [49] [52] [54] [46].

Like in the railway domain, “clean sheet” developments are rare due to the risks involved [55]. The strategy instead is that the original design must be evolvable. Also, safety and predictability are extremely important in this sector [49]. This is achieved, among other things, by partitioning [49].

Of the set of ARINC standards, the most common one with a wide acceptance through its use by a variety of COTS operating system vendors is the ARINC 653 RTOS standard [48]. It defines an operating environment for application software used within Integrated Modular Avionics (IMA) [49] for real-time and safety-critical applications [51] [46] using cyclic scheduling and a preemptive fixed priority-based policy [49] and enforces time and space constraints to be statically defined before execution [10]. The ARINC standards also define communication models for local communication, e.g., shared memory and message queues, as well as for distributed communication, e.g., Advanced Full Duplex Network (AFDX), DO-178B TCP/IP or Time Triggered Ethernet (TTE) [51].

3.1.3 Industrial automation

In the industrial automation, there are many different forms with different characterizations. These include distributed control [46], multi-agent systems [56], smart devices [56], intelligent sensors [57], autonomous robotics [57], cloud computing [57], Industrial PCs (IPCs) [46] or up-to-date approaches like Evolvable Production Systems (EPS) [56] [58] and Cyber-Physical Production Systems (CPPS) [46] [59] for example. But there is one characteristic what many solutions in industrial automation (at least the traditional ones) have in common: Usually Programmable Logic Controllers (PLCs), so special-purpose microcontroller which continuously runs a program cycle, are used to run control logic [46]. A standard for distributed control systems based on PLCs is the IEC 61499, where applications consist of function blocks that communicate with each other via events and data connections [12]. For communication between PLCs there are several industrial standards like Open Platform Communication-Unified Architecture (OPC UA) (using client/server model), Profinet (with cyclic sender/receivers) or DDS (based on publish/subscribe pattern) [46].

Former issues and challenges include the long time for system design, commissioning and setup, a complex and time-consuming reengineering, incompatibility between different vendors equipment and legacy systems, and inflexible centralized/hierarchical implementations [56]. According to [58] also an update of functional safety standards is required needing significant research activities.

Traditional architectures like the “industrial automation pyramid” focused on integration between hierarchical layers, such as American National Standards Institute ANSI/ISA-95 from ISO/IEC 62264 version of 2007 [57]. But for more flexible and adaptable production systems, the lower layers of the pyramid tend to collapse to more “autonomous” CPPS to enable Industry 4.0 [46].

Trends in the industrial automation domain include edge control using hypervisors, use of more IT-like technologies and “software-defined manufacturing” which led to research on potential enablers like industrial internet of things (IIoT), information and communications technology (ICT) or artificial intelligence (AI), and in general research in the direction of reconfiguration on process level [46].

3.1.4 Other domains

In other, non-safety-related areas (e.g., typical “IT systems”) the characterizations and architectures vary to a much greater extent. A few examples for architectural patterns are client-server, mobile agents [60], model-view-controller, hexagonal architecture, or the publish-subscribe pattern [61]. Some further architectural patterns or techniques such as service-oriented architecture (SOA), microservices and containerization [60] are widespread nowadays and are widely used in a variety of IT systems. The individual, concrete approaches that were found during the literature research will not be listed or described in detail here. However, these are included in the listings of the following sections.

Due to the mentioned characterizations and architectures, which differ (partially) from the railway domain, the approaches for evolvability found in the literature (see section 3.2) need to be analyzed in detail whether they are also suitable for use in the railway sector (see subsection 3.2.5). It is also possible to adapt or expand existing approaches or just adopt concepts from them.

The characterizations and architectures just mentioned (section 3.1), which differ (partially) from the railway domain, need to be considered when the approaches for evolvability found in the literature (see section 3.2) will be analyzed whether they are also suitable for use in the railway sector (see subsection 3.2.5).

3.2 SOLUTION APPROACHES FOR EVOLVABILITY FROM OTHER DOMAINS

In this section, the working group summarizes and structures the results of a literature research on evolvable software architectures. The focus is on the domains automotive, avionics and industrial automation. However, literature from other domains (e.g., enterprise IT) as well as domain-unspecific literature were also considered. For a better overview, the listings of the solution approaches are split into architectural patterns, techniques and methodologies (see subsection 3.2.1), concepts and principles (see subsection 3.2.2), concrete solutions and approaches (see subsection 3.2.3), and metrics (see subsection 3.2.4).

The various solution approaches found in the literature have been consolidated. For this purpose, the essential components (e.g., architectural patterns or concepts) of each solution were extracted and classified under appropriate generic terms in the listings. These generic terms are given in the first column “Approach” of each listing. The second column “NFP” (non-functional property) lists all non-functional properties from deliverable D29.1 [1] that are either explicitly mentioned by at least one concrete solution in the literature or classified by the working group as a property addressed by the respective approach. Properties that are not primarily addressed by the basic approach, but for which the approach can make a supportive contribution (e.g., only under certain conditions), are given in brackets. The third column, “User Story”, lists the user story IDs from deliverable D29.1 [1] that are addressed by the approach in the same way as described for the NFPs. Both the NRPs and the user story IDs are listed in the (rough) order in which the working group assesses the impact of the respective approach on the stated NRP or user story. The fourth column, “Reference”, lists the literature that discusses the respective approach in some way or uses it in one's own solution approach. In the "Detail" column of each listing, further individual information is provided.

3.2.1 Architectural patterns, techniques and methodologies

In Table 4, various architectural patterns, techniques and methodologies for evolvable software architectures found in the literature have been consolidated into a listing, using the columns as described in section 3.2.

Approach	NFP	User Story	Reference	Detail
Service-Oriented Architecture (SOA)	changeability maintainability compatibility extensibility (portability) (testability)	10006 10004 10005 (10012) (10007) (10013)	[62] [38] [60] [40] [63] [56] [3] [39]	Fine-grained (distributed) entities ("services") are combined to an application. Promotes reuse of functionality and engineering efficiency (smaller entities are easier to develop, adapt and maintain). Supports dynamicity at runtime (e.g., dynamic discovery of services or network communication established dynamically at runtime).
Microservices	extensibility maintainability changeability compatibility portability (testability) (traceability)	10006 10005 10007 10004 10015 (10012) (10007) (10013) (10009) (10008)	[3] [60] [64]	Compared to services in SOA, microservices are even more fine-grained and especially more independent (very few or no dependencies to other microservices = loose coupling) and self-contained (with own, decentral data storage). Each microservice can use different technologies (technology-independent implementation) and communicate across different platforms. Due to the strong independence, the potential reuse is reduced, but e.g., updates are easier (incl. security patches).
Virtualization	portability (maintainability) (safety) (security)	10001 10015 (10016) (10006) (10012)	[51] [54] [59] [65]	Allows to exchange HW platforms without affecting functional system behavior. Might help to increase non-functional properties like performance during lifetime or scaling capabilities. Might support for safety and security due to isolation, partitioning, or separation (e.g., allowing mixed critically).
Containers	portability testability extensibility changeability maintainability (configurability) (safety) (security)	10001 10009 10006 10012 10004 10015 10007 10005 (10010)	[39] [59] [60] [64] [66]	A (lightweight) form of virtualization. Self-contained, highly portable, isolated units of software. Provide a good basis for DevOps and engineering efficiency. Promote easy deployment, update, and exchange of SW modules. Implementations of Open Container Initiative (OCI) standard are e.g., Docker or Podman.

Approach	NFP	User Story	Reference	Detail
Model-Driven Development (MDD) / Software Development (MDSD) / Architecture (MDA) / Engineering (MDE)	traceability maintainability (verifiability) (changeability) (maintainability) (adaptability) (portability) (safety)	10011 10002 10004 (10000) (10003) (10006) (10005)	[38] [67] [46] [3] [45] [68] [5]	Systematic use of models throughout the software engineering life cycle. MDD helps to reduce complexity, supports reuse, and might also support (formal) verification or defining and evaluating Quality-of-Service (QoS) attributes. Code generation based on model-driven approaches allows to trace the actual relationships (e.g., between the architecture, documentation, and code)
Enterprise Modeling	traceability	(10011) (10002)	[3]	Similar to model-driven approaches, but on a more abstract, higher level. Allows to check changes and impact on a higher level (e.g., processes, …),
Product Line Engineering (PLE)	adaptability configurability maintainability traceability (verifiability) (changeability)	10000 10003 (10004)	[69] [54] [5]	Helps to manage SW over multiple projects based on a (proven) reference base and supports to handle new (tailored) projects and variability.
Component-Based Software Engineering (CBSE)	(maintainability) (adaptability) (changeability) (extensibility) (traceability)	(10006) (10005) (10000)	[46] [70]	Basic paradigm based on software entities to support reusability. One concrete form of this concept is, for example, SOA.
Normalized Systems Theory	changeability maintainability extensibility	10005 10006 (10015) (10004)	[3]	Method of software engineering aiming fine-grained modularity (based on separation of concerns). Small modules (e.g., from automated code generation) support reducing complexity and allow to accommodate change.
Axiomatic Design	verifiability traceability (maintainability) (changeability)	10018 (10009) (10011) (10016) (10015)	[55]	Method for structured design based on the assignment of requirements to solutions (with the help of matrices). Aims at reducing complexity, like principle “parts reduction principle”, trying to share closely related functions (e.g., in modules) and use duplicate parts as far as possible.

Approach	NFP	User Story	Reference	Detail
Quality-Driven Software Architecture (QDSA)	(maintainability) (adaptability) (extensibility)	(10003)	[71]	Method for software architects to ensure quality attributes in software architectures using a “quality tree” with scenarios.
Health Monitoring	diagnosability (maintainability)	10017	[72]	Components (on different architectural levels) provide built-in diagnostic functions. These are invoked by a “health monitoring” application that performs analyses (or also e.g., fault logging, troubleshooting or other maintenance tasks).
Distributed Processing	maintainability (changeability) (extensibility) (testability)	(10015) (10006) (10005) (10012) (10008) (10009)	[54]	Components of a system are computed on different entities (of various types and forms). One concrete form of this concept is, for example, SOA.
Feature Orientation	adaptability configurability maintainability traceability (changeability) (extensibility) (verifiability)	10003 10004 10014 10000 10011 (10002)	[69] [2]	Supports to handle new tailored projects and variability. Available in various forms and forms (e.g., feature activation or deactivation at runtime).
Cloud Computing	changeability extensibility maintainability portability (adaptability) (testability)	10005 10013 10007 10012 (10014) (10006) (10017)	[41]	Partial outsourcing of software calculations (e.g., non-safety-critical or non-latency-critical) to the cloud and sending the results to the train in the form of commands or information (e.g. travel recommendations).
Mockups	(testability)	(10003)	[3]	Can increase engineering efficiency. Also, can support to validate the customer's requirements (early feedback), but not to verify that the later SW system fulfils them.
Low Code / No Code	portability maintainability adaptability changeability (traceability)	10001 10003 10004 (10008) (10005)	[3]	Can increase engineering efficiency (shorter time-to-market by support to develop new SW faster and reduce maintenance cost). Can reduce complexity (using prebuilt components). Can support technical

Approach	NFP	User Story	Reference	Detail
				variability e.g., change of technologies (regenerate applications for other target platforms based on code generation and mode-driven approach).
Scripting Language	extensibility maintainability (compatibility) (portability)	10004 (10005) (10008)	[70]	Supplementally use scripting languages (e.g., for non-safety-related parts) to improve support for application development (as they tend to be flexible and must not be recompiled).
Mobile agents	portability changeability	10001 10005 10006	[60]	Allow to move computation incl. code, data, and state to servers (e.g., from one host to another). Also aimed for client customization (e.g., adding new features).
Multi-Agent Systems (MAS)	configurability changeability adaptability (extensibility)	10014 10003	[56] [73] [58] [74]	A decentralized group of autonomous, distributed agents that collectively solve a common problem using high-level (semantic) communication and interaction. New features and adaptions are easy possible but is likely to have (unpredictable) impact on the existing components/system.
Architecture Description Language (ADL)	maintainability changeability verifiability (extensibility) (portability) (safety)	10009 10005 (10002) (10009) (10001) (10011)	[75] [68]	Combining formal methods with concepts of components and connectors, applying principles like abstraction, correspondence, and type completeness, and allowing executable specifications (e.g., ArchWare or LEDA).
Domain-Specific (Modeling) Language (DSL/DSML)	(verifiability) (maintainability) (changeability) (adaptability) (portability)	(10011) (10002) (10003) (10009) (10000) (10004)	[46]	The achievable NFPs depend heavily on the specific domain, language, and concrete realization. But DSL-based validators and generators (based on automated model transformations) could reduce complexity via abstraction and automation [46].

Table 4. Architectural patterns, techniques and methodologies.

As one can see in Table 4, there is no single architectural pattern, technique or methodology that fulfills all or at least most of the NFPs at the same time. The same applies to the addressed user stories. For an overall architecture that covers as many as possible or all of the targeted NFPs and user stories, an appropriate combination must be found. However, it may be the case that some approaches counteract each other and cannot be combined.

3.2.2 Concepts and principles

The columns used as described in section 3.2 and as already done in subsection 3.2.1 for architectural patterns, techniques and methodologies (in Table 4), Table 5 now lists *concepts* or *principles* that support software architectures evolvability found in the literature from the different domains. Again, the mentioned NFPs and user story IDs are not always complete in the sense that not every property that is supported in any form by the approach is listed, but rather those that the working group considers to be the most relevant.

Approach	NFP	User Story	Reference	Detail
Abstraction	portability maintainability changeability adaptability extensibility	10001 10015 10006 10005 10003 (10000) (10002) (10012)	[69] [67] [46] [49] [70] [75]	Basic principle. Concrete forms are e.g., abstraction from target Operating System (OS) and hardware platforms (e.g., Hardware Abstraction Layer (HAL)), Application Programming Interfaces (APIs) or communication.
Separation Of Concerns	maintainability changeability extensibility (portability)	10012 10006 10005 (10015) (10003) (10002)	[3] [63] [46] [75] [76]	Every concern (e.g., drivers of change, technology, or data access) should be separated from other concerns. This is one principle of Normalized Systems and supported by e.g., MDD, CBSE or SOA.
Modularity	maintainability testability changeability extensibility (verifiability) (traceability)	10012 10002 10006 10005 10009 (10007) (10015)	[37] [69] [77] [72] [52] [78] [70] [2] [79]	Breaking down a problem into smaller, more manageable modules. Improves maintainability and supports changes without affecting other modules.
Layered Architecture	portability compatibility maintainability changeability	10015 10001 10005 (10008) (10002)	[63] [48] [57] [62] [69] [38] [37] [46] [80] [75] [54] [44]	Multiple horizontal layers (HW and SW) each with a defined responsibility (specific function/role) thus promoting separation of concerns. Reduces the impact/effects of system changes on other layers (incl. applications).

Approach	NFP	User Story	Reference	Detail
Decentralization	changeability extensibility (testability) (maintainability)	10006 10005 10015 (10002)	[70] [79]	Distribute resources and computation to different (logical or physical) locations (e.g., using Microservices). Can reduce change propagation.
Information Hiding	maintainability testability changeability traceability (extensibility) (portability)	10002 10006 10005 10015 (10001)	[69] [46] [67] [77]	Hide details of e.g., deployment, bus and network topology or underlying technologies (e.g., encapsulate infrastructure technology choices and provide interfaces for application software). Reduces the impact of system changes on other system parts.
Isolation	safety security extensibility (changeability) (verifiability) (testability) (portability)	10012 10006 10005 10002 (10015)	[46] [51] [54] [72] [70] [81] [82] [59] [65]	General concept with different types (e.g., temporal or spatial) and various variants, e.g., isolation of (system) services, functions, tasks, modules (e.g., trusted/untrusted), resources, or test isolation. Techniques include virtualization (e.g., hardware-assisted Trusted Execution Environments like ARM TrustZone or Intel SGX), programming languages (e.g., Modula or SPIN OS), or containers. Can support certification and homologation (e.g., regarding safety and security), secure user space, or prevent fault propagation (e.g., compromising other functionality).
Partitioning / Separation / Segmentation	[see “Isolation”]	“ “	[37] [38] [39] [63] [51] [54] [72] [49] [83] [42]	A kind of isolation. Partitioning of e.g., resource usage (like memory space, computation time, access, or backplane). Common form is “time and space partitioning” (e.g., by RTOS or hypervisor-based). Allows mixed-criticality tasks/services (e.g., safety and non-safety-critical) running on the same hardware (e.g., as standalone Virtual Machine (VM) or within an OS partition). Can facilitate security (e.g., by restricting data flow between partitions).

Approach	NFP	User Story	Reference	Detail
Commonality	compatibility (maintainability) (changeability)	10018 (10016) (10008) (10000)	[55] [73] [54] [72] [80]	Cluster components (HW and SW) based on similarity (e.g., via domain analysis) and share functionality among these (e.g., processes, technologies, interfaces, or infrastructure). Promotes design reuse and reduces component count (incl. spares). Supported by using (system) features, platforms, product families or (software) product lines.
Redundancy	safety (changeability) (portability)	(10015) (10006) (10016)	[39] [62] [63] [84] [51] [54] [70] [79] [73]	Can include both HW (e.g., lockstep processor) and SW (e.g., master/slave protocol). Different types, e.g., dislocality (SW units deployed on two distinct HW components) or dissimilarity (SW units deployed on HW of different type, different suppliers or using different processors, cores, etc.). Supports safety or availability (e.g., allows failure or removal of module without losing a system function).
Loose Coupling High Cohesion	maintainability testability changeability (extensibility) (portability)	10015 10006 10005 (10002) (10012)	[69] [40] [60] [39] [64] [2] [63] [70] [85]	Can be achieved by e.g., encapsulation, service grouping (SOA), decoupling of components and inter-process communications (e.g., from specific data transfer mechanisms) or using standardized interface pattern. System changes affect fewer modules and module changes affect fewer other modules.
Publish-Subscribe	maintainability portability	10003 (10012) (10001) (10008)	[39] [38] [80] [81] [83]	Enables location-independent and protocol-transparent communication. Decouples communication and application logic (loose coupling). Data consumer and producer does not know each other and can find each other dynamically (at runtime). Typically used in event-based systems (register for notification on data changes or other specific conditions). An example of a real-time middleware is DDS.

Approach	NFP	User Story	Reference	Detail
Design by Contract	verifiability testability (traceability)	10009 10002 10006 (10011) (10012) (10005) (10015)	[67]	A form of “assume/guarantee” (A/G) technique. Definition of interface assertions (e.g., mode-based contracts) and contract compatibility analyses. Might support (re-)homologation, (regression-)testing and impact analysis of changes.
Meta Modeling	traceability verifiability (changeability) (maintainability)	10009 10011 (10005) (10006) (10002)	[5] [46]	Document e.g., architectural design decisions and trace them to related requirements and implementation. Or use versioned components and meta model to identify compatibility-breaking updates (based on data types and timing) for example.
Interface Handling	compatibility maintainability (testability)	10008 10015 (10006) (10004)	[38] [40] [62] [44] [67] [46] [77] [54] [72] [70] [64] [79]	Prefer common interfaces for better integrability. Aim for backward-compatible interfaces. Interface assertions (cf. “Design by Contract”) support verification of interface compatibility. Define system interfaces between subsystems. Restrict access and visibility of interfaces (e.g., of services).
Namespaces	(testability) (traceability) (changeability)	10006 10005 (10002) (10009) (10011)	[46] [39] [59]	Isolate (incl. resource usage) and limit the visibility of e.g., processes, network interfaces, mount points, workspaces, microservices or containers.
Zones	compatibility maintainability changeability extensibility	10012 10015 10008 10013 (10007) (10006) (10005) (10002)	[37] [70]	Divide architecture (HW or SW) into zones to separate e.g., legacy software (for compatibility reasons), data (locally distributed), or “make” parts done by manufacturer from “buy” parts from suppliers or third-party components.
Self-Reconfiguration	configurability adaptability changeability	10014 10003	[46] [79]	Capability of an entity (e.g., system, device, or application) to change autonomously its configuration (e.g., to build ad-hoc ensembles for collaboration). This might be supported

Approach	NFP	User Story	Reference	Detail
				by multi-agent systems (MAS) and is incorporated by e.g., the Evolvable Production System (EPS) paradigm.
Commercial Off-The-Shelf (COTS)	compatibility maintainability portability	10008 10015 10001 10018 (10016)	[83] [72] [51] [50]	Using COTS components (e.g., Ethernet Local Area Network (LAN) HW like IP routers, or general-purpose processors) supports to develop a system cost-effectively, to keep it modern (e.g., due to market forces, third-party participation, and successive compatible generations of a product line) and handle obsolescence.
Open System Principle	compatibility maintainability portability	10008 10015 10001 10018 (10016)	[72] [78] [54] [78]	This system design principle is supported by e.g., a modular design, technology-independence, and the use of published, widely supported and controlled or consensus-based standards (especially for interfaces). It enables the use of commercial technology and products (COTS) and aims for interoperability and replacement or upgraded of HW or SW with alternate components. Open System Architectures (OSA) are e.g., AUTOSAR (for automotive) or Modular Open System Approach (MOSA) (for avionics).
Location Transparency	portability (changeability) (extensibility)	10001 (10003)	[54]	Entities (e.g., applications) don't know where other entities reside and thus are not affected when entities migrate to other HW. Therefore, objects or resources should be identified and accessed without knowledge about their location (e.g., using logical names and registries).

Table 5. Concepts and principles.

Just like with the approaches in subsection 3.2.1, to cover as many of the different NFPs and user stories as possible, suitable concepts and principles must also be selected and combined into an overall architecture.

3.2.3 Concrete solutions and approaches

This subsection names and shortly describes selected, concrete solutions and approaches from other domains. The architectural patterns, techniques and methodologies described in 3.2.1 as well as the concepts and principles described in 3.2.2 were, among others, extracted from these concrete solutions. For clarity, the listings are split into the domains avionics (see Table 6), automotive (see Table 7), industrial automation (see Table 8) and others (see Table 9).

First, Table 6 lists concrete solutions and approaches from the avionics domain. Safety, security as well as standardization (including use of COTS components) play a particularly important role in these.

Approach	Reference	Detail
Integrated Modular Avionics (IMA)	[48] [49] [50] [46]	Architecture supporting COTS. Truth-based scheme where each entity recognizes an internal failure and removes itself from the system. Uses lock-step processor, serial bus and time and space partitioning via ARINC 653 RTOS. Allows real-time, safety-critical, and certifiable applications.
Multiple Independent Levels of Security and Safety (MILS)	[39] [51]	Joint research effort to develop a high-assurance, real-time architecture for embedded systems. Supports non-bypassable, evaluable, available, and tamper resistant security at the RTOS level. Technical foundation is a separation kernel.
Common Avionics Architecture System (CAAS)	[54]	Open system architecture (incl. HW and SW) used for a helicopter product line based on e.g., variability isolation, connectivity, modularity, layering, partitioning, and redundancy with the vision for a scalable system, reducing cost, and addressing obsolescence and modernization issues.
Modular Open Systems Approach (MOSA)	[51] [78]	Driven by the need for COTS, it describes the reference for HW and SW standards (incl. network, middleware, third-party applications, security protocols, storage, or OS). Using Advanced Full DupleX (AFDX) or Time-Triggered Ethernet (TTE) it can even integrate safety critical and non-critical controls/systems onto the same network with mixed levels of redundant operation.
Microkernel Hypervisor RTOS VM	[51]	[51] describes a feasibility assessment for the “Microkernel Hypervisor RTOS Virtual Machine (VM) architecture” to “enable virtualization for a representative set of avionics applications requiring multiple guest OS environments”. These include legacy applications (on a legacy RTOS guest OS), safety-critical applications (on an ARINC 653 OS), and MILS applications (on a high assurance OS) for example. All executing on different VMs on a “Microkernel Hypervisor RTOS within a Multicore (X86 or Power PC) with

Approach	Reference	Detail
		hardware-based virtualization support". The paper addresses design issues, limitations/restrictions, and the feasibility of applying this approach.
XtratuM	[49]	[51] describes an approach for an embedded architecture using the bare-metal hypervisor "XtratuM" designed for safety-critical applications to extend the "trusted environment" from the HW level to the hypervisor. It is based on time and space partitioning (TSP) as defined in ARINC 653 and MILS, and includes an interrupt model, health monitoring, fault management. Executable entities (partitions) are executed on top of a VM.

Table 6. Concrete solutions and approaches from avionics domain.

In the following, Table 7 lists some concrete solutions and approaches from the automotive domain. Safety, security, and open standards also play an important role in these.

Approach	Reference	Detail
AUTOSAR Classic Platform (CP)	[43] [44] [45] [46]	This current software platform, which is ubiquitous in the automotive domain, was already described in subsection 3.1.1.
AUTOSAR Adaptive Platform (AP)	[43] [47] [37] [39] [38]	This next generation architecture for the automotive domain has also been described in subsection 3.1.1 already.
AutoFOCUS	[38]	Research prototype for model-driven development supporting formal verification capabilities. At a certain degree of required integrity, such formal techniques are highly recommended by the automotive functional safety standard ISO 26262. But contrary to evolutionary goals, according to [84] hard-coded optimization rules (such as AutoFOCUS) "restrict the engineers' flexibility".
Robust and Reliant Automotive Computing Environment (RACE)	[44] [41] [39] [44]	This project proposes a "single, scalable computing platform as a central vehicle controller" (centralized architecture) inspired by ARINC 653 with redundancy and fail-safe operation. The project also developed the open source publish-subscribe middleware "CHROMOSOME" that uses concepts from DDS (but only supports a subset of its QoS policies). RACE provides "a safety-critical execution environment with interfaces for verifying and testing the components" and is capable of "Plug & Play" what fits to the "concept of highly sensorised cars". Depending on the installed sensors and SW, a software component can change its behavior.

Table 7. Concrete solutions and approaches from automotive domain.

Table 8 lists selected solutions and approaches from the industrial automation domain. A prominent characteristic of these approaches is their high degree of flexibility and adaptability. In some cases, even at runtime, which is a very strong form of evolvability.

Approach	**Reference**	**Detail**
Flexible (FMS) / Reconfigurable (RMS) / Holonic (HMS) Manufacturing Systems	[74] [58]	FMS were a first step towards adaptive manufacturing systems through “flexibility at the machine and routing levels”. The later RMS promotes “modular and scalable manufacturing stations to achieve a faster response to change of markets and customers”. HMS are based on a concept of autonomous and co-operative building blocks, later expanded to “intelligent agents”.
Evolvable Assembly Systems (EAS)	[74]	The EAS project [74] builds upon and extends the “PRIME” project, a multi-agent architecture for plug and produce based on standard technology proposed in [86]. The philosophy of EAS is a “four-phase cycle” consisting of the phases (re-)configuration, operation, monitor, and definition or adaptation (external or internal). EAS proposes a distributed software architecture “based on the principles of decentralization, context-awareness and intelligent resources, that is implemented using intelligent agent technology and a data distribution service”.
RAMI 4.0	[87]	“RAMI” is a reference architecture model as an orientation aid for Industry 4.0. It is a three-dimensional consolidation of the most important aspects of Industry 4.0 and is intended to ensure that all participants have a common perspective and build a common understanding. To this end, it relies on international cooperation and strives for global interoperability.
Cyber-Physical Systems (CPS)	[57] [46] [66]	In manufacturing (Industry 4.0), these include intelligently networked field devices, machines, production modules and products that autonomously exchange information, trigger actions, and control each other independently. Such systems are characterized by a high number of software-controlled functions, sensors, and actuators. They focus on information and flexibilize the integration of different layers within the architecture.
Cyber-Physical Production Systems (CPPS)	[57] [46] [59]	More recent approaches focus on Cyber-Physical Production Systems (CPPS), which are special, more autonomous CPS that make use of AI technologies to reduce human supervision and where virtualization of applications is becoming more present to provide higher flexibility.
Evolvable Production Systems (EPS)	[56] [58]	According to [58], EPS was “one of the most promising emerging paradigms aimed at revolutionizing the manufacturing industry by incorporating adaptability, self-reconfiguration and intelligence at the shop-floor level” in 2015. The approach is process-oriented, based on

Approach	Reference	Detail
		"skills" and enables runtime modifications. It allows for modular, self-managing systems based on intelligent, agent-based distributed control and provides "Plug & Produce" at the level of sensors and actuators.

Table 8. Concrete solutions and approaches from industrial automation domain.

Finally, Table 9 lists selected, concrete solutions and approaches from other domains (e.g., enterprise IT) as well as from domain-unspecific literature (e.g., "embedded systems" in general). Even if these address non-safety-related areas, they can potentially also be applied or transferred to other areas, such as the railway domain.

Approach	Reference	Detail
Recovering Architectural Assumptions Method (RAAM)	[5]	Method "that makes assumptions explicit by recapitulating historical information of software system evolution", where assumptions are modeled invariability (design decisions that are assumed not to change). These can then help to assess the evolutionary capabilities of a system architecture or to provide what-if scenarios (what if an assumption proves to be invalid).
System Architecture Optimization Language (SAOL)	[84]	Specification language for both objectives (optimization goals) and constraints. Optionally extended by compatibility relations and concepts of dislocality and dissimilarity for redundancy. Can be used to decide on which execution unit to deploy which software application for example.
Architecture-Level Modifiability Analysis (ALMA)	[88] [89] [5] [70]	Method for quantitative architecture analysis that "analyzes modifiability based on scenarios that capture future events a system needs to adapt to in its lifecycle" [5]. Can be used for maintenance prediction (required effort for system modifications due to future changes), architecture comparison of multiple candidates, or for risk assessment for example.
Architecture Evolvability Analysis (AREA)	[90] [91] [70]	Method for systematic assessment with the goal to "provide quality attribute subcharacteristics values", "identify the weak parts of the system architecture related to evolvability", and "analyze the quality attribute subcharacteristics of the possible evolutions" of a system [70]. Can support to make architecture requirements and corresponding design decisions more explicit and documented. It includes a technical review that can be applied at different points during system life cycle (e.g., at design phase or for evaluating a legacy system that is changed).
ArchWare Architecture Analysis Language (AAL)	[68] [75]	AAL is a "formal property expression language designed to support automated verification" (e.g., using model-checking or theorem proving) and provides a framework that allows architects to specify and verify relevant properties of software architectures and styles.

Approach	Reference	Detail
Portability Layer	[77]	Use of a “portability layer” that encapsulates infrastructure technology choices and provides interfaces for application SW. This leads to openness for e.g., different OS vendors (like VxWorks).
Java Agent Development Environment (JADE)	[58] [74]	Platform for implementing an agent-based control architecture (cf. “Multi-Agent Systems (MAS)” in Table 4).
Open Service Gateway initiative (OSGi) Service Platform	[62] [82]	A “runtime framework that supports visibility constraints between OSGi bundles” which are “a Java archive or a Web application archive file” [62]. The visibility is declared using a manifest file.
Proteus	[85]	A “framework which is intended to support the development of adaptable software architectures using design patterns” illustrated by way of a home appliance control system. It presents ”how to analyze and use design patterns as potential adaptability enhancers in developing software systems”.
Lightweight Sanity Check for Implemented Architectures (LiSCIA)	[76]	Evaluation method “that can reveal potential problems as a software system evolves” and “helps to determine which quality criteria the system meets module”. Is covers the five categories “source group”, “module functionality”, “module size”, “module dependencies” and “technologies.
Design and Engineering Methodology for Organizations (DEMO)	[3]	Enterprise modeling technique (cf. “Enterprise Modeling” in Table 4) that focuses on creating an “ontological model” of an enterprise that “defines the products and services that the enterprise delivers through actors, including the underlying processes, information and business rules, independent of its technological implementation”. Used in [3] as starting point for a MDSD approach incl. automated and traceable transformation of DEMO models into working software.

Table 9. Concrete solutions and approaches from other domains.

3.2.4 Metrics

Architectural metrics are a means of analyzing and evaluating the quality of an architecture. Metrics allow to make quality attributes or non-functional properties (NFPs) measurable or assessable. This can be used, for example, to justify architecture design decisions or to evaluate them retrospectively. But metrics can also be used to determine specific properties of a system at runtime. Such could be used, for example, in combination with the DevOps approach, which relies on continuous feedback. Therefore, not only architectural metrics are listed in Table 10, but various others as well.

Approach	Reference	Detail
Interface complexity	[79]	Expresses "the relative difficulty of a given change to an interface, measured in percentage of original design effort, as a function of the percent change needed". Relies on interviews to develop the relationships. Can be used to "decide where modular boundaries (and therefore interfaces) should be defined in a given system."
Dependency complexity	[2]	When "a component is changed, its change has effects on other components through dependencies. The complexity of dependencies is the determining factor of maintaining architectures".
Design complexity	[69]	Proposes metrics "to measure complexity of the design".
Cyclomatic complexity	[64]	A metric that is "aggregated from measurements of individual methods" [92].
Cognitive complexity Architectural complexity Technical complexity	[64]	Mentions "metrics for cyclomatic complexity, cognitive complexity, architectural / technical complexity".
Functional cohesion	[63]	Mentions metrics "like functional cohesion".
Evolution ratio	[5]	"Amount of evolution in terms of software size".
Evolution speed	[5]	"Indicator of an organization's capability for software system's evolution".
Implementation change logs Software life span Software size	[5] [91]	Mentions metrics that "base on implementation change logs" or "on software life span and software size".
Module number	[91]	Refers to "computation of metrics using the number of modules in a software system".
Maintainability Binary Size	[69]	Refers to an approach uses metrics to measure the non-functional properties "maintainability", "binary size" and

Approach	Reference	Detail
Performance		"performance" in software product lines", which are "used to compute optimized software product line (SPL) configurations according to user-defined non-functional requirements".
Code metrics	[64]	Measure attributes like lines of code (LOC), test coverage, cyclomatic complexity, clone coverage, defect resolution time, CI/CD pipeline duration, or count of defects per service, failed tests, code smells, endangered requirements, outdated dependencies or rule violations.
IOSA	[5]	Impact of each "scenario profile" is measured through "impact on the software architecture" (IOSA).
ADSA	[5]	Impact of each "scenario profile" is measured through "adaptability degree of software architecture" (ADSA).
Visibility matrix	[79]	A matrix where element's row is the "visibility fan out" (VFO), the number of dependencies it has on other elements, and element's column is the "visibility fan in" (VFI), the number of elements that depend on it.
Costs associated	[79]	Proposes to add a cost function to represent and measure cost and time of a given change in a parameter (not interface). Defining a cost threshold, these metrics "could then be used to explore the tradespace to see what subset is reachable with the given resources".
Process-oriented	[5] [91]	Analyzes "the degree of software architecture adaptability through intuitive decomposition of goals and intuitive scoring of goal-satisfying level of software architecture" [5].
Downtime	[59]	Refers to authors that state that downtime is a relevant metric "since it used in service level agreements (SLA)".
Isolation	[65]	Defining "measures of spatial and temporal isolation that could point out isolation issues", e.g., a "developer could use the existing performance isolation metrics".
Product line maintainability	[69]	Refers to a metric with which product lines are measured using the metric "maintainability index".

Approach	Reference	Detail
Service maintainability	[64]	Mentions “service-oriented maintainability metrics” and its applicability for microservices.
Service complexity	[40]	Refers to “a metric to measure the complexity of composite services” defined by Liu and Traore.
Service grouping	[40] [62]	Proposes a “set of metrics for evaluation of architectural design candidates in distributed safety-critical SOA” [40].
Coupling and cohesion (for service groups in SOA)	[40]	Proposes “a set of metrics to measure the coupling and cohesion quality attributes of derived service groups of architectural designs for a distributed SOA”.
Latency Container performance	[59]	Refers to an approach that “tried to address” the gap to “measure the different types of latency” by “studying operating system-level metrics, as well as metrics to specifically evaluate the timeliness of tasks running in the system, and adapted those to assess the RT performance of containers”.

Table 10. Metric approaches from other domains.

Further, a more in-depth discussion and overview of approaches for quality evaluation at software architecture level including metric-based, but also experience-based and scenario-based, approaches can be found in [5].

According to [79] “no perfect evolvability measure currently exists”. As reasons why metrics fall short, it is mentioned that some “measure a different ility (such as adaptability or complexity)”, some are “time intensive and are potentially unreliable (such as relying on interviews)” and others “require very highly developed models before they can be applied”. Nevertheless, metrics, as described at the beginning of this subsection, can make an important contribution to improve the quality of software architectures and thus also promote the evolvability of them.

3.2.5 Applicability of approaches within railway domain

This subsection outlines the applicability of the approaches consolidated in subsections 3.2.1 to 3.2.4. It is limited to fundamental considerations, such as addressing specific issues and possible difficulties. A more in-depth analysis and evaluation of the applicability of specific approaches will be part of the next deliverable (D29.3).

Basically, the concepts and principles consolidated in subsection 3.2.2 (see Table 5) are more likely to be applicable in the railway domain because of their higher abstraction than the concrete, partly domain-specific solutions shown in subsection 3.2.3. Nevertheless, also the applicability of the former, as with all approaches collected in this deliverable, must be evaluated individually.

However, the fact that an approach can be *implemented* within the railway domain does not mean that it can be *applied* sensibly. For example, “cloud computing” or “mockups” will undoubtedly be technically feasible, but whether there are also use cases within the railway domain where these approaches can be used sensibly still needs to be considered. From CPPS (see Table 8) one can possibly learn aspects regarding flexibility, but on the one hand it is questionable whether this form and degree of flexibility is necessary within a train, and on the other hand this approach from industrial automation as a whole obviously does not fit the conditions within the railway domain. Interesting approaches, which the working group could not clearly classify into “certainly applicable” or “certainly not applicable” right away, are e.g., SOA or microservices. These have promising aspects, but whether the respective holistic architectural pattern also fits the conditions and structures of a train control system, for example, still needs to be examined in more detail.

Further, solution approaches cannot always be *combined* with each other without restrictions. It is possible that approaches counteract or even exclude each other. A simple example is the decision for a centralized or decentralized architecture. Even if these can possibly be combined within an overall system, but at a specific level, only one of the two concepts can be applied. In addition, the objectives (e.g., specific NFPs) of individual approaches can also contradict each other. For example, “self-reconfiguration” (which e.g., enables strong flexibility and promotes engineering efficiency) on the one hand, and the “design by contract” technique (which can e.g., support homologation) on the other. While a safety-critical control function of a train based on self-reconfiguration is rather not eligible regarding homologation, the design by contract approach would rather not be applicable to all system components of a train with a justifiable effort and counteracts engineering efficiency. In such cases, it must then be evaluated individually which combination of appropriate approaches represents the more suitable software architecture overall.

In addition, the *context* will have to be considered during the evaluation and classification even within the railway domain, e.g., which (sub-)system is involved and what requirements are placed on it. One example is IT and OT parts, which may both be present within the overall software architecture of a train. Especially when virtualization or partitioning is used to operate (sub-)systems or applications with different requirements together (e.g., on the same HW or within an OS partition). Such different requirements could be e.g., the NFPs security or safety (“mixed-critically”), but also QoS attributes (e.g., latency). If, in such a context, certain approaches are not applicable to individual units (e.g., subsystems, applications, or partitions) due to specific requirements, they may still be applicable to other units. This means that varying solution approaches could be applied in different parts of a train to meet distinct requirements (e.g., NFPs) in a targeted and needs-based manner. However, it should be noted that the more different approaches and technologies are used, the more complex the overall system becomes, which in turn could counteract evolvability.

Finally, if it is not possible to make a clear classification for individual solution approaches whether they can be applied sensibly within the railway domain or if they are classified as not applicable, it is still possible to draw inspiration from these and, for example, to adapt certain aspects of it.

4 CONCLUSIONS

In this deliverable, the *problem* space worked out in the previous deliverable D29.1 [1] was addressed by examining *solution* approaches for DevOps and software architecture evolvability from other domains.

The increasing importance of software and rising level of connectivity of safety-critical products is continuously improving and adding functionality. *DevOps* development principles support such kind of continuous deployment. However, safety-critical products must meet security and safety standards. Therefore, the first step was to analyze how it is possible to approach the DevOps concept from the perspective of cybersecurity and safety, taking as a basis the two main standards, i.e., ISA/IEC 62443-4-1 and IEC 61508, which constitute the legacy. After an exhaustive review of the state of the art of DevOps in industrial environments, no relevant work has been found that approximates the paradigm in this environment. This constituted the motivation to develop a practical approach-based on the mapping of one of the user stories defined in deliverable D29.1 [1] on the numerous DevOps stages. For each of the stages, a set of tools was defined to enable the security properties defined in each case.

As the variety of non-functional properties (NFPs) and user stories worked out in D29.1 [1] has already indicated, the literature research has confirmed that *software architecture evolvability* is a multifaceted quality attribute with a wide spectrum of sub characteristics. But of the approaches found, each addresses only a subset of these characteristics or NFPs. The focus differs depending on the domain. While a high degree of safety and reliability plays a major role in the avionics domain, for example, recent approaches from industrial automation often focus more on a high degree of flexibility, even at runtime. However, it has been shown that the utilized architectural patterns, techniques, and methodologies overlap and are often based on the same or similar concepts and principles. The working group has extracted these from the various approaches found from the different domains, classified them, and provides them in form of structured listings, including a mapping to the NFPs and user stories from D29.1 [1], within this deliverable. On this basis, the working group can evaluate which of the existing solutions and approaches from the other domains can also be applied within the railway domain in the next step (D29.3).

The working group concludes that there is no ready-made, “off-the-shelf” solution, neither for DevOps nor for architectural evolvability, that fits the conditions in the railway domain and can entirely fulfill all the requirements and user stories defined in D29.1 [1]. However, there are approaches or at least aspects of them which obviously or very likely can also be used. The trick will be to make a suitable selection of applicable approaches which can be combined and then to integrate them into the overall architecture and processes to build a suitable solution that enables the desired NFPs and user stories.

REFERENCES

[1] Ceit and Siemens Mobility GmbH, "Flagship Project 2 - Rail to Digital automated up to autonomous train operation, D29.1 – List of user stories and requirements," Europe's Rail Joint Undertaking, 2023.

[2] T. Wang and B. Li, "Analyzing Software Architecture Evolvability Based on Multiple Architectural Attributes Measurements," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, Sofia, Bulgaria, 2019.

[3] M. R. Krouwel, "On the Design of Enterprise Ontology-Driven Software Development," Ph.D dissertation, Maastricht University, Maastricht, 2023.

[4] D. Rowe, J. Leaney and D. Lowe, "Defining Systems Evolvability - A Taxonomy of Change," in *IEEE International Conference on the Engineering of Computer-Based Systems*, 1998.

[5] H. P. Breivold, I. Crnkovic and M. Larsson, "A systematic review of software architecture evolution research," *Information and Software Technology,* vol. 54, no. 1, pp. 16-40, January 2012.

[6] "DevOps Market size worth over $30 Bn by 2028," 22 03 2022. [Online]. Available: https://www.gminsights.com/pressrelease/devops-market.

[7] C. Schmittner, J. Dobaj, G. Macher and E. Brenner, "A preliminary view on automotive cyber security management systems," in *DATE '20: Proceedings of the 23rd Conference on Design, Automation and Test in Europe*, Europe, 2020.

[8] P. Munk and M. Schweizer, "DevOps and Safety? SafeOps! Towards Ensuring Safety in Feature-Driven Development with Frequent Releases. In Computer Safety, Reliability, and Security.," in *SAFECOMP 2022 Workshops : DECSoS, DepDevOps, SASSUR, SENSEI, USDAI, and WAISE*, Munich, Germany, 2022.

[9] M. Johnson, D. Cummings, B. Leinwand and C. Elsberry, "Continuous Testing and Deployment for Urban Air Mobility," in *AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, San Antonio, TX, USA, 2020.

[10] S. Gupta, P. Lago and R. Donker, "A framework of software architecture principles for sustainability-driven design and measurement," in *In Proceedings of the 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, Stuttgart, Germany, 2021.

[11] H. Yasar, "Waterfall to DevSecOps in DoD," 2019. [Online]. Available: https://apps.dtic.mil/sti/trecms/pdf/AD1085204.pdf. [Accessed 2023].

[12] The Linux Foundation, "With Kubernetes, the U.S. Department of Defense Is Enabling DevSecOps on F-16s and Battleships," 2020. [Online]. Available: https://www.cncf.io/case-study/dod.

[13] "DevOps for Railway Propulsion System Design," 2019. [Online]. Available: https://www.aidoart.eu/aidoart/use-cases/3.

[14] J. Ayerdi, A. Garciandia, A. Arrieta, W. Afzal, E. Enoiu, A. Agirre, G. Sagardui, M. Arratibel and O. Sellin, "Towards a Taxonomy for Eliciting Design-Operation Continuum Requirements of Cyber-Physical Systems," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, Zurich, Switzerland, 2020.

[15] D. Butler, "Comparing the IEC 62443 Software Engineering Process to IEC 61508: Where Do They Overlap?," 2018. [Online]. Available: https://www.exida.com/blog/comparing-the-iec-62443-software-engineering-process-to-iec-61508.

[16] S. Gautham, A. V. Jayakumar, A. Rajagopala and C. Elks, "Realization of a Model-Based DevOps Process for Industrial Safety Critical Cyber Physical Systems," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, Victoria, BC, Canada, 2021.

[17] F. Moyon, K. Beckers, S. Klepper, P. Lachberger and B. Bruegge, "Towards continuous security compliance in agile software development at scale," in *In 2018 ieee/acm 4th international workshop on rapid continuous software engineering (rcose)*, 2018.

[18] R. M. Soares, "Large Scale Agile Software Development compliant to IEC 62443-4-1 Artefact Design and Tool support," University Institute of Lisbon, Lisbon, 2019.

[19] F. Moyón, R. Soares, M. Pinto-Albuquerque, D. Mendez and K. Beckers, "Integration of Security Standards in DevOps Pipelines: An Industry Case Study.," in *PROFES 2020: Product-Focused Software Process Improvement*, Turin, Italy, 2020.

[20] F. Moyón, D. M. Fernández, K. Beckers and S. Klepper, "How to integrate security compliance requirements with agile software engineering at scale?," 2021.

[21] P. Bitra and C. S. Achanta, "Development and Evaluation of an Artefact Model to Support Security Compliance for DevSecOps," https://www.diva-portal.org/smash/get/diva2:1531206/FULLTEXT02, 2021.

[22] K. Hanssen, G. T. Stålhane and T. Myklebust, "SafeScrum – Agile Development of SafetyCritical Software," *Springer,* p. 2018.

[23] T. Myklebust, G. T. Stålhane and K. Hanssen, "Agile Safety Case and DevOps for the automotive industry," in *Proceedings of the 30th European Safety and Reliability Conference and 15th Probabilistic Safety Assessment and Management Conference*, 2020.

[24] S. Gautham, A. V. Jayakumar, A. Rajagopala and C. Elks, "Realization of a Model-Based DevOps Process for Industrial Safety Critical Cyber Physical Systems," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, Victoria, BC, Canada, 2021.

[25] Siemens AG, "Electrical components for the railway industry," Siemens AG, 2018. [Online]. Available: https://assets.new.siemens.com/siemens/assets/api/uuid:3c66c0dc-294c-40bb-b3fd-204777ad79ec/dfcp-b10083-00-7600-ws-railway-components-144.pdf.

[26] Atlassian, "Jira Software Cloud support," [Online]. Available: https://support.atlassian.com/jira-software-cloud/.

[27] Microsoft, "GitHub," [Online]. Available: https://github.com/. [Accessed 2023].

[28] GitHub Action, "Codecov GitHub Action," [Online]. Available: https://github.com/marketplace/actions/codecov. [Accessed 2023].

[29] "Building and testing Java with Maven," [Online]. Available: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-maven.

[30] "Building and testing .NET," [Online]. Available: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-net.

[31] "Building and testing Python," [Online]. Available: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python.

[32] GitHub Action, "Dependency Review," [Online]. Available: https://github.com/marketplace/actions/dependency-review.

[33] Github Action, "OWASP ZAP Full Scan," [Online]. Available: https://github.com/zaproxy/action-full-scan.

[34] GitHub Action, "Anchore SBOM Action," [Online]. Available: https://github.com/marketplace/actions/anchore-sbom-action.

[35] Siemens, "Ruggedcom Rox II v2.9," [Online]. Available: https://cache.industry.siemens.com/dl/files/700/109481700/att_863595/v1/ROXII_v2.9_RX1500_User-Guide_WebUI_EN.pdf.

[36] Datadog, "Datadog," [Online]. Available: https://www.datadoghq.com/.

[37] S. Kugele and M. Broy, "Architecture as a Backbone for Safe DevOps in Automotive Systems," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, Macau, China, 2022.

[38] P. Obergfell, S. Kugele and E. Sax, "Model-Based Resource Analysis and Synthesis of Service-Oriented Automotive Software Architectures," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, Munich, Germany, 2019.

[39] S. Kugele, D. Hettler and J. Peter, "Data-Centric Communication and Containerization for Future Automotive Software Architectures," in *2018 IEEE International Conference on Software Architecture (ICSA)*, Seattle, WA, USA, 2018.

[40] V. Cebotari and S. Kugele, "Playground for Early Automotive Service Architecture Design and Evaluation," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, Las Vegas, NV, USA, 2020.

[41] C. Prehofer, K. Schorp, S. Kugele, D. Clarke and M. Duchon, "Towards a 3-tier architecture for connected vehicles," in *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, Vienna, Austria, 2014, 2014.

[42] P. Obergfell, S. Kugele, C. Segler, A. Knoll and E. Sax, "Continuous Software Engineering of Innovative Automotive Functions: An Industrial Perspective," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Hamburg, Germany, 2019.

[43] "Automotive Open System Architecture," AUTOSAR, [Online]. Available: http://www.autosar.org.

[44] S. Shafaei, F. Müller, T. Salzmann, M. H. Farzaneh, S. Kugele and A. Knoll, "Context Prediction Architectures in Next Generation of Intelligent Cars," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI, USA, 2018.

[45] A. Bucaioni, A. Di Salle, L. Iovino, S. Kugele and Y. Dajsuren, "Joint Workshop on Model-Driven Engineering for Software Architecture (MDE4SA) and International Workshop on Automotive System/Software Architectures (WASA)," in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, L'Aquila, Italy, 2023.

[46] K. Telschig, "Evolving distributed embedded applications during operation," Ph.D dissertation, Universität Augsburg, Augsburg, 2023.

[47] "Adaptive Platform," AUTOSAR, [Online]. Available: https://www.autosar.org/standards/adaptive-platform.

[48] R. Black and M. Fletcher, "Next generation space avionics: a highly reliable layered system implementation," in *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*, Salt Lake City, UT, USA, 2004.

[49] A. Crespo, I. Ripoll and M. Masmano, "Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach," in *Eighth European Dependable Computing Conference*, Valencia, Spain, 2010.

[50] Z. S. Mor, N. Asghar and G. Inalhan, "Avionics Architecture Design for a Future Generation Fighter Aircraft," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, San Diego, CA, USA, 2019.

[51] T. Gaska, B. Werner and D. Flagg, "Applying virtualization to avionics systems — The integration challenges," in *29th Digital Avionics Systems Conference*, Salt Lake City, UT, USA, 2010.

[52] W. D. Ivancic, "Modular, Cost-Effective, Extensible Avionics Architecture for Secure, Mobile Communications," in *2006 Aerospace Conference*, Big Sky, Montana, 2006.

[53] SAE ITC, "ARINC Standards," ARINC Industry Activities, [Online]. Available: https://aviation-ia.sae-itc.com/product-categories/arinc-standards.

[54] P. Clemente and J. Bergey, "The U.S. Army's Common Avionics Architecture System (CAAS) Product Line: A Case Study," Defense Technical Information Center, Pittsburgh, 2005.

[55] A. S. J. van Heerden, M. D. Guenov and A. Molina-Cristóbal, "Evolvability and design reuse in civil jet transport aircraft," *Progress in Aerospace Sciences,* vol. 108, pp. 121-155, July 2019.

[56] G. Candido, A. W. Colombo, J. Barata and F. Jammes, "Service-Oriented Infrastructure to Support the Deployment of Evolvable Production Systems," *IEEE Transactions on Industrial Informatics,* vol. 7, no. 4, pp. 759-767, 2011.

[57] M. Saturno, V. Pertel, F. Deschamps and E. Rocha Loures, "Proposal of an automation solutions architecture for Industry 4.0," *DEStech Transactions on Engineering and Technology Research,* vol. 14, no. 2, pp. 185-195, 2018.

[58] A. Rahatulain and M. Onori, "Production System Innovation Through Evolvability: Existing Challenges and Requirements," *Journal of Machine Engineering,* vol. 15, no. 3, pp. 50-64, 2015.

[59] R. Queiroz, T. Cruz, J. Mendes, P. Sousa and P. Simões, "Container-based Virtualization for Real-time Industrial Systems — A Systematic Review," *ACM Computing Surveys,* vol. 56, no. 3, pp. 1-38, 2023.

[60] T. Salah, M. Jamal Zemerly, C. Y. Yeun, M. Al-Qutayri and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, Barcelona, Spain, 2016.

[61] "Architectural pattern," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Architectural_pattern. [Accessed 05 12 2023].

[62] V. Cebotari and S. Kugele, "On the Nature of Automotive Service Architectures," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Hamburg, Germany, 2019.

[63] S. Kugele, P. Obergfell, M. Broy, O. Creighton, M. Traub and W. Hopfensitz, "On Service-Orientation for Automotive Software," in *2017 IEEE International Conference on Software Architecture (ICSA)*, Gothenburg, Sweden, 2017.

[64] J. Bogner, J. Fritzsch, S. Wagner and A. Zimmermann, "Assuring the Evolvability of Microservices: Insights into Industry Practices and Challenges," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA, 2019.

[65] L. De Simone and G. Mazzeo, "Isolating Real-Time Safety-Critical Embedded Systems via SGX-Based Lightweight Virtualization," in *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Berlin, Germany, 2019.

[66] C. A. Garcia, M. V. Garcia, E. Irisarri, F. Pérez, M. Marcos and E. Estevez, "Flexible Container Platform Architecture for Industrial Robot Control," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin, Italy, 2018.

[67] S. Kugele, D. Marmsoler, N. Mata and K. Werther, "Verification of component architectures using mode-based contracts," in *2016 ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, Kanpur, India, 2016.

[68] F. Oquendo, B. Warboys, R. Morrison, R. Dindeleux, F. Gallo, H. Garavel and C. Occhipinti, "ArchWare: Architecting Evolvable Software," in *Software Architecture, First European Workshop, EWSA 2004*, St Andrews, UK, 2004.

[69] A. Grewe, C. Knieke, M. Korner and A. Rausch, "Automotive Software Product Line Architecture Evolution: Extracting, Designing and Managing Architectural Concepts," *International Journal on Advances in Intelligent Systems,* vol. 10, no. 3 & 4, pp. 203-222, 2017.

[70] H. P. Breivold, I. Crnkovic and M. Larsson, "Software Architecture Evolution through Evolvability Analysis," *Journal of Systems and Software,* vol. 85, no. 11, pp. 2574-2592, November 2012.

[71] P. Hruschka, "Quality Driven Software Architecture," in *Software Quality. Process Automation in Software Development*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2012, pp. 10-13.

[72] J. Borky, R. Lachenmaier, J. Messing and A. Frink, "Architectures for next generation military avionics systems," in *1998 IEEE Aerospace Conference Proceedings (Cat. No.98TH8339)*, Snowmass, CO, USA, 1998.

[73] J. Engle and T. Moseman, "An affordable and flexible architecture for deep space exploration," in *2016 IEEE Aerospace Conference*, Big Sky, MT, USA, 2016.

[74] J. Chaplin, O. Bakker, L. de Silva, D. Sanderson, E. Kelly, B. Logan and S. Ratche, "Evolvable Assembly Systems: A Distributed Architecture for Intelligent Manufacturing," *IFAC-PapersOnLine,* vol. 48, no. 3, pp. 2065-2070, 2015.

[75] R. Morrison, G. Kirby, D. Balasubramaniam, K. Mickan, F. Oquendo, S. Cimpan, B. Warboys, B. Snowdon and R. Greenwood, "Support for evolving software architectures in the ArchWare ADL," in *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, Oslo, Norway, 2004.

[76] E. Bouwers and A. van Deursen, "A Lightweight Sanity Check for Implemented Architectures," *IEEE Software,* vol. 27, no. 4, pp. 44-50, 2010.

[77] H. P. Breivold, I. Crnkovic and P. J. Eriksson, "Analyzing Software Evolvability," in *2008 32nd Annual IEEE International Computer Software and Applications Conference*, Turku, Finland, 2008.

[78] T. Gaska, "Optimizing an incremental modular open system approach (MOSA) in avionics systems for balanced architecture decisions," in *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, Williamsburg, VA, USA, 2012.

[79] J. Beesemyer, D. Fulcoly, A. Ross and D. Rhodes, "Developing Methods to Design for Evolvability: Research Approach and Preliminary Design Principles," in *9th Conference on Systems Engineering Research*, Los Angeles, CA, 2011.

[80] K. Schmid, I. John, R. Kolb and G. Meier, "Introducing the PuLSE Approach to an Embedded System Population at Testo AG," in *ICSE '05: Proceedings of the 27th international conference on Software engineering*, St. Louis, USA, 2005.

[81] M. Gagliardi, R. Rajkumar and L. Sha, "Designing for evolvability: building blocks for evolvable real-time systems," in *Proceedings Real-Time Technology and Applications*, Brookline, MA, USA, 1996.

[82] M. Lahami and M. Krichen, "Test Isolation Policy for Safe Runtime Validation of Evolvable Software Systems," in *2013 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Hammamet, Tunisia, 2013.

[83] L. Sha, R. Rajkumar and M. Gagliardi, "Evolving dependable real-time systems," in *1996 IEEE Aerospace Applications Conference. Proceedings*, Aspen, CO, USA, 1996.

[84] S. Kugele, G. Pucea, R. Popa, L. Dieudonné and H. Eckardt, "On the deployment problem of embedded systems," in *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, Austin, TX, USA, 2015.

[85] L. Chung, K. Cooper and A. Yi, "Developing adaptable software architectures using design patterns: an NFR approach," *Computer Standards & Interfaces,* vol. 25, no. 3, pp. 253-260, 2003.

[86] N. Antzoulatos, E. Castro, D. Scrimieri and S. Ratchev, "A multi-agent architecture for plug and produce on an industrial assembly platform," *Prod. Eng. Res. Devel. 8,* p. 773–781, 2014.

[87] Plattform Industrie 4.0, "RAMI 4.0," [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/rami40-eine-einfuehrung.pdf.

[88] P. Bengtsson, N. Lassing, J. Bosch and H. van Vliet, "Architecture-level modifiability analysis (ALMA)," *Journal of Systems and Software,* vol. 69, no. 1-2, pp. 129-147, 2004.

[89] N. Lassing, D. Rijsenbrij and H. van Vliet, "How well can we predict changes at architecture design time?," *Journal of Systems and Software,* vol. 65, no. 2, pp. 141-153, 2003.

[90] H. Pei-Breivold and I. Crnkovic, "An Extended Quantitative Analysis Approach for Architecting Evolvable Software Systems," in *Computing Professionals Conference Workshop on Industrial Software Evolution and Maintenance Processes (WISEMP10), IEEE*, 2010.

[91] H. P. Breivold, I. Crnkovic, R. Land and M. Larsson, "Analyzing Software Evolvability of an Industrial Automation Control System: A Case Study," in *2008 The Third International Conference on Software Engineering Advances*, Sliema, Malta, 2008.

[92] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner and G. Saake, "Measuring Non-Functional Properties in Software Product Line for Product Derivation," in *2008 15th Asia-Pacific Software Engineering Conference*, Beijing, China, 2008.