Ref. Ares(2023)7794059 - 21/12/2023

Rail to Digital automated up to autonomous train operation

D29.1 – List of user stories and requirements

Due date of deliverable: 2023-06-01

Actual submission date: 2023-07-13

Leader/Responsible of this Deliverable: Roelle, H.; SMO

Reviewed: Y

Document status		
Revision	Date	Description
01	2023-07-13	Filed to TMT (reviewed by WP members)
02	2023-12-19	Revised after TMT comments and reviewed by all WP members

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	User stories, DevOps, 2nd opinion, review
Figueroa, Santiago	CEIT	User stories, DevOps, 2nd opinion, review
Roelle, Harald	SMO	User stories, Architecture drivers, 2nd opinion, review
Oertel, Norbert	SMO	User stories, Architecture drivers, 2nd opinion, review

Disclaimer

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The content of this document reflects only the author's view – the Joint Undertaking is not responsible for any use that may be made of the information it contains. The users use the information at their sole risk and liability.

EXECUTIVE SUMMARY

“Software eats the world!” This is a rather drastic statement which appears to be inline with recent developments in the rail domain. There is a clear trend to shift to software defined train control functions and there is an increasing number of IT systems on-board the trains.

In contrast to classical, wayside non-rail IT systems (e.g., Netflix deploying to their main application in the magnitude of minutes!), the software on trains has a very long life-time, but still needs to be able to respond to frequent updates due to security fixes and upcoming new features required by customers. In addition there are onboard domains where life-time has become much shorter than with others (e.g. passenger entertainment vs. core traction control).

In this deliverable the working group has collected a number of user stories, which will serve as a guidance for the upcoming investigation into how we can make the architecture of our SW systems more responsive to change, to be more ‘evolvable’.

Further, the working group has discussed reasons for why architecting for evolution is actually a necessity. Partly because the amount of software systems is continuously growing, but also because it is simply not feasible to predict the future. Considering every possible variability is not possible for a number of good reasons – the SW architecture needs to be evolvable at some point in the future when a certain change in requirements or constraints is happening.

The group has collected best practices from other domains, the DevOps approach, which addresses similar problems in a holistic way. DevOps considers, besides architecture, also organization and processes.

The constraints and non-functional requirements between this domain and the railway domain appear to be different, however, the main drivers for evolvability remain the same. The working group sees a large potential of re-interpreting and applying DevOps principles and related architectural styles to the rail domain in order to

- improve engineering efficiency,
- reduce time-to-market for new features and changes,
- and overall increase the evolvability of on-train SW systems.

For the further work on “architecting for evolution” the working group has refined and documented the definition of scope, collected user stories and architecture driving non-functional properties, as set out in this deliverable.

Next step will be to analyze DevOps processes more deeply and map these onto existing processes in the rail domain.

ABBREVIATIONS AND ACRONYMS

ATO	Automatic Train Operation
R2DATO	Rail to Digital automated up to autonomous train operation
TOC	Train Operating Company[1]
SW	Software
HW	Hardware
DevOps	Development and Operations (Methodology)

[1] The term “TOC“ was chosen intentionally. To make clear that in commercial setups where responsibility for the track and responsibility for the trains is within different companies, that the interests of train running company are addressed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

2 INTRODUCTION

Rail industry and rail operators face significant challenges that result in conflicting time scales for operations (long, with the need for adaptions and changes over decades) on the one hand, and realization of new features and fixes (short, shall be brought into the field much faster than today) on the other hand. For the long timescale we propose to explore ways to improve the evolvability of architectures and for the quick realization we propose to focus on ideas from the DevOps approach. Both must be done consistently with requirements from standards and regulations, harsh environment, safety-criticality and long lifetime. To determine the right granularity of being technology-independent while preserving applicability in practice will be the key question.

In this work package we constrain ourselves to on-board systems. With these there are special limitations, especially in contrast to IT-systems, involved:

- For systems in the rail domain in general, see 3.3
- In contrast to wayside systems, on-board systems are not always connected to the wayside with high bandwidth/quality or not connected at all (e.g. by quality of LTE/5G links, or trains being completely shut down when not in service). This gives additional constraints, especially when homogeneity across a whole fleet is required or at least desired.

So, constraining to on-board systems causes no harm as this is the most challenging case. Being specific for the rail domain, it shall be easy to transfer and apply the results for the wayside.

On the Rail-industrial DevOps aspect:

- Tackling realization time in rail industry products is not only about architecture and runtime environments but also about development process or in general how the assets are realized while will show up on train. Additionally, lead times are heavily influenced by certification and homologation.
- There is much to learn from “ordinary” IT business. The advantages of agile processes, combined with the ideas of continuous development, cont. build, cont. test, cont. deployment, continuous integration mark a significant leap there. Taking these ideas even further into operations led to the concept of DevOps.
- Also in the railway industry, the DevOps idea as such has a great potential.
- But the basic idea of expanding continuous-X into operations needs adequate adaptation for rail business’ surroundings.

On the Architecting4evolution aspect:

- Avoiding disruptive changes of rail sector’s long living systems implies they need to be intrinsically evolvable.

- Change and evolution should be explicitly tangible in a software architecture – to make it manageable.

Design goals for evolution include:

- Design modules and services as autonomous units.
- Encapsulate uncertainty, risk and change.
- Minimize need for explicit adaption.

3 DEFINITION OF CORE CONCEPTS

3.1 EVOLUTION OF ARCHITECTURE

"Design for evolution" is key for designing systems with long lifetime and "never-goes-obsolete" requirements.

We strongly believe in this statement, and to understand the reason behind, one can look at a widely used alternative: The universal and generic approach that tries to build all variability and flexibility explicitly in. Unfortunately, this has significant shortcomings:

- It introduces much unnecessary variability, because over a long lifetime the actual needed features and properties are hard to estimate beforehand.
- It lacks variability for the same reason.
- Not hitting the necessary extent and kind of variability, this introduces accidental complexity and technical debt.
- With unneeded complexity and technical debt, non-functional properties like performance and robustness are unnecessarily impaired.

Therefore, for a software architecture to be evolvable, it needs to support change, for example:

- Ease of adding new functions and taking out old functions.
- Ease of migration to new versions, or new technologies.
- Ease of supporting new deployment scenarios.

Change is considered mostly harmful. To make change manageable, change and evolution must be explicitly addressed in the software architecture. To achieve this goal, we believe that the following architectural design goals shall be considered:

- Single Responsibility Principle – there should be a single reason for changing a SW component, module, service or interface. The scope or granularity of this 'single reason' scales with the scope of the unit under consideration.
- Open-closed principle – Be open for extension of SW assets and interfaces but closed for modification.
- Interface design shall follow the Liskov Substitution Principle, such that an entity may be replaced by a functionally extended version of itself.
- Be aware of implicit and explicit coupling and reduce the grade of coupling where possible.
- Encapsulate uncertainty, risk and change. Most interesting is not the choice between A and B, but the fact that there -is- a choice between A and B.

3.2 DevOps

DevOps (the combination of Development and Operations) is a methodology, a set of tools and practices with a focus on reducing time to market for product features, even when the product as a such is already deployed and available on the market. It has gained much attention in the IT-industry in past years, even a standard emerged about (ISO/IEC/IEEE 32675).

The combination of development and operations into a wholistic methodology shortens the feedback cycle between operating a product and how it is used, and the development responsible for developing new product features and fixing any existing issues. The methodology does not per se imply a specific architectural style. Certain architectural styles have however got momentum due to the DevOps movement, e.g., microservices based architectures.

Microservice based architectures organize an application into a collection of individual services or microservices. These microservices are highly maintainable and testable, are loosely coupled, independently deployable, organized around business capabilities and owned by a small team. They therefore allow to replace and evolve parts of the architecture during operations, ideally with zero downtime. The applicability of this architectural style depends on the application's business logic, considering elements such as application complexity and rapid, frequent and reliable delivery over a long period of time.

DevOps is not limited to Microservices architectures, as the methodology can also be applied to monolithic architectures.

When discussing the architectural aspects of DevOps, one must not forget that architecture is entangled with adequate forms of organization and processes as depicted in Figure 1.

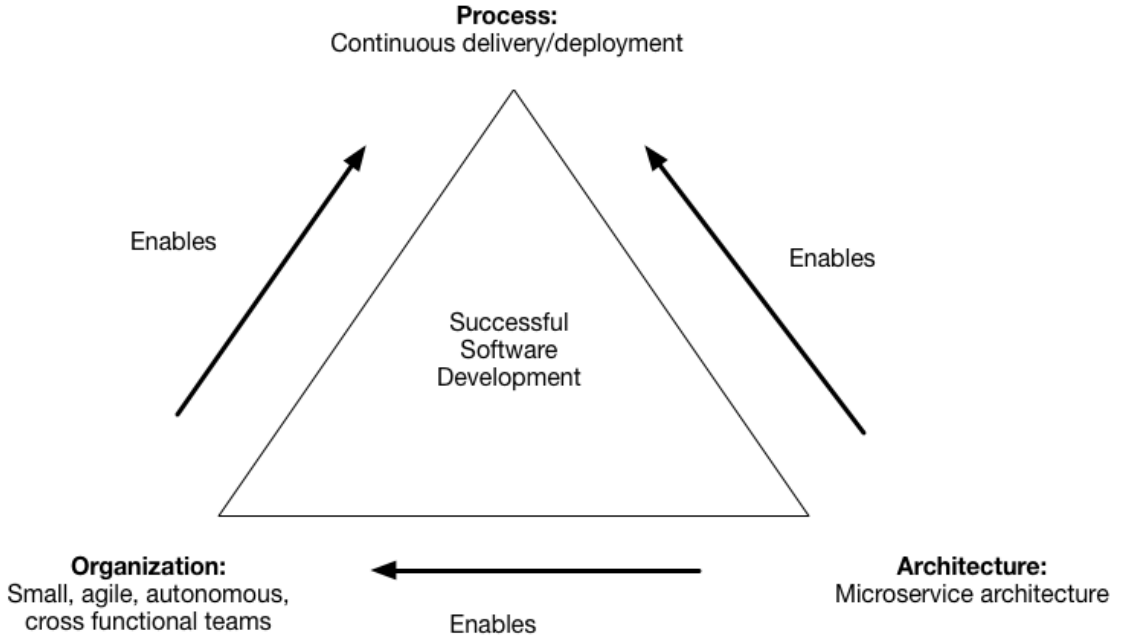

Figure 1 Successful Software Development Triangle [1]

The book DevOps Handbook ([2]) establishes a relation between process and DevOps in the "Successful Software Development" triangle context. The Team Topologies book ([3]) describes a high-performance IT organization is a loosely coupled network of small, autonomous, and empowered teams. Each team is relatively small, ideally five to nine people. That number is chosen because research shows that small teams promote trust, which is essential for high productivity. Also, each team should be long-lived because it takes time for a team to become highly effective. But mitigating the risk on team stability is a risk that general management / human resources need to address and cannot be dealt with in this work package.

Key to DevOps processes is process automation. This is especially important for development, build, deployment and testing processes, which requires adequate infrastructure and tools.

3.3 Software Engineering in the Rail Domain

SW Engineering in the rail domain is fundamentally different from the engineering of cloud-based services or classical IT systems. The key differences are:

- It is not only SW engineering, but SW engineering embedded into an overall system engineering context.
- The development process is dominated by V-model and formal homologation.
- The resulting SW system needs to be designed to live for a very long time, typically in the order of 30 years. Frequent updates are required to fix security issues or to meet new customer requirements which emerge over time.

- The systems engineering aspect increases the number of drivers for evolution, e.g. due to obsolescence of underlaying HW components.
- HW selection and compute performance is limited due to rail specific constraints (passive cooling, temperature ranges, etc.).
- Trend to increased number of IT systems and even more frequent updates and extensions of functionality.
- SW in the rail domain is not a product business, but a multi-project solution business.
- It is a low-volume business compared to the automotive domain.

Despite these key differences we see many of the same drivers for DevOps as described in section 3.1. Therefore, we see a large potential of re-interpreting and applying DevOps principles and related architectural styles in order to

- improve engineering efficiency,
- reduce time-to-market for new features and changes,
- and overall increase the evolvability of on-train SW systems.

This motivates to investigate IT-DevOps principle and methodologies and how these could be adapted to the rail domain.

4 SCOPING OF THE WORK PACKAGE

The scope of the work package is defined to be:

- We are limiting ourselves to the SW-Engineering aspect - constraints stemming from the overall systems engineering context will be considered.
  - SW development, deployment, and testing processes.
  - SW architecture with a focus on the key design principles as laid out in section 3.1.
  - Process automation and required tooling.
- We are limiting ourselves to on-train SW systems, in contrast to wayside SW systems.
- Derived from the user stories in section 5.2 we will consider the following scenarios or design aspects:
  - Distributed system aspects
  - SW deployment scenarios
  - SW configuration and parameterization
  - SW updates

    - SW monitoring
- Not reinventing the wheel – deep dive into what can be re-used and /or adapted from existing process frameworks, technologies and methodologies!

5 LIST OF RELEVANT USER STORIES

5.1 STRUCTURE OF USER STORIES

The user stories presented in this chapter follow an easy to comprehend structure:

1. ID: Identification number of the respective user story for future reference.
2. Actor: The acting role of the user story, respectively the mainly interested role. One or more of the following stakeholders:
    - Train manufacturer
    - Train operating company (TOC)
    - Homologation body
    - Maintainer
3. User Story: The user story itself, written in the form: "As a <role>, I want <goal/desire> so that <benefit>"
4. Driver 1 and 2: One or two main drivers/motivation of the stakeholder for submitting the story. These drivers in most cases are desirable or constraining quality attributes; in addition, they may also be business drivers.

5.2 USER STORIES

ID	Actor	User Story	Driver 1	Driver 2
10000	Train Manufacturer	As a train manufacturer, I want to be able to keep multiple projects' SW aligned with a reference base without impacting on-train components of other stakeholders, e.g., TOCs, so that I can minimize internal maintenance efforts while keeping already achieved quality.	Maintainability	Engineering Efficiency
10001	Train Manufacturer	As a train manufacturer, I want to efficiently port the train SW to new HW platforms, so that I can evolve with improved non-functional properties and deal with HW obsolescence over time.	Obsolescence Management	Portability
10002	Train Manufacturer	As a train manufacturer, I want to have support for efficient impact analysis of changes, so that I can provide the necessary information to homologation bodies.	Verifiability	Engineering Efficiency
10003	Train Manufacturer	As a train manufacturer, I want to have support to efficiently tailor the SW and system architecture, so that I can efficiently handle new train projects with varying train topologies.	Adaptability	Engineering Efficiency

ID	Actor	User Story	Driver 1	Driver 2
10004	Train Manufacturer	As a train manufacturer, I want to have well defined and documented variation points, so that I can implement project specific extensions without changing the overall system and SW-architecture.	Extensibility	Maintainability
10005	Train Manufacturer	As a train manufacturer, I want to be able to change parts of the SW architecture so that (re-)homologation can be kept to a minimum or is not needed at all.	Changeability	Compatibility
10006	Train Manufacturer	As a train manufacturer I want to exchange SW components without impacting other components, so that test effort can be minimized and re-homologation is not required.	Changeability	Verifiability
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.	Maintainability	Verifiability
10008	Train Manufacturer	As a train manufacturer, I want to integrate third-party components with standardized interfaces, so that I don't have to adapt to 3rd party specific interfaces.	Engineering Efficiency	Compatibility
10009	Train Manufacturer	As a train manufacturer, I want to verify functional and non-functional features of the SW before deploy it on the train.	Testability	Verifiability

ID	Actor	User Story	Driver 1	Driver 2
10010	Train Manufacturer	As a train manufacturer, I want to automate the deployment of the SW on every train and have the tools to monitor and observe the deployment process.	Changeability	Diagnosability
10011	Homologation Body	As a homologation body, I want to see concise, precise, traceable, and comprehensible documentation, so that I can efficiently decide on homologation aspects of the project.	Homologation	Traceability
10012	TOC	As a TOC, I want to be able to deploy SW/applications on a train without breaking homologation status or negatively influencing other components, so that I can swiftly roll out applications satisfying upcoming customer demands.	Extensibility	Homologation
10013	TOC	As a TOC, I want to receive security patches quickly and want to apply them short term without harming vehicle operation, so that my trains are always in line with security's state-of-the-art.	Maintainability	Minimal service outage
10014	TOC	As a TOC, I want to change the system behavior in a timely manner without a SW update, so that I can adapt to changing operational needs.	Configurability	Changeability
10015	Maintainer	As a maintainer, I want to replace hardware or software with functionally and non-functionally equivalent parts without compromising the overall system behavior, so that I can handle cases of obsolescence.	Compatibility	Maintainability

ID	Actor	User Story	Driver 1	Driver 2
10016	Maintainer	As a maintainer, I want to efficiently exchange devices of the system with spare parts, so that I can repair diagnosed defects.	Maintainability	Compatibility
10017	Maintainer	As a maintainer, I want to have precise diagnostic information, so that I can accurately replace defect components.	Diagnosability	Traceability
10018	Maintainer	As a maintainer, I want to have a low variety of spare parts, so that required logistics and warehousing is cost efficient.	Maintainability	Compatibility

Table 1 List of user stories

6 MAIN ARCHITECTURAL DRIVERS

Architectural drivers from DevOps's current application domains (mostly IT) differ from those in the railway industry. E.g., micro services architectures support IT industry's drivers quite well, while using micro services in the railway domain needs an in-depth analysis whether the different/additional drivers are equally well served, and constraints can be met.

As can be concluded from section 3.3, the main difference between the development of cloud-based IT-systems and on-board train software are non-functional requirements and constraints. These affect all aspects of building, delivering and operating on-train SW, especially the development process, verification and validation processes (testing), the homologation process and operational qualities of the final product in service and maintenance.

This is also reflected in Table 1, which in most cases lists quality attributes (non-functional properties) as main drivers for submitting the user story.

The main business goals which need to be addressed, are:

- **Engineering efficiency and obsolescence management** for the train manufacturer
    - Improved time to market
    - Lower cost
    - Lower effort and cost for software maintenance
    - Quick response to security issues and new feature requests
    - Efficiently deal with obsolescence issues
- **Increased flexibility** for the train operating company (TOC)
    - Offer new value-added services quickly
    - Lower cost
    - Lower effort and cost for integrating such services into the train
    - Reduced manual maintenance intervention
    - Increased availability due to reduced maintenance downtimes

Some ubiquitous non-functional properties are not explicitly mentioned as drivers in the list of user stories. These include, but are not limited to:

- **Security**: All activities must maintain an adequate level of security. Since trains are critical infrastructure and the number of cyberthreats are continuously increasing this is of increasing importance as an architectural driver.

- **Performance**: For any activity that is constrained by a certain execution time or requires certain performance properties. In contrast to wayside IT systems, the compute performance on-train is severely limited due to constraints like power consumption and heat dissipation.
- **Safety and homologation**: For any activity that is constrained by a certain safety level, the architecture needs to support and protect this in an adequate way. Towards operations this also means an adequate support for homologation.

The non-functional properties listed in Table 1 are (in alphabetical order):

- **Adaptability**: Be able to adapt the software platform quickly to a concrete project context.
- **Changeability**: Efficiently change the software to add new features, project specific customizations, bug- and security-fixes. Change must be managed, and the architecture needs to be prepared for evolutional change pressure.
- **Compatibility**: Standardized interfaces to 3rd party components and systems, as well as customer IT systems on-board, ensure that efforts for adaptation are kept low.
- **Configurability**: Change defined characteristics of the SW system during operations, ideally without the need for a SW update.
- **Diagnosability**: Besides the typical usage of diagnostics information for maintenance purposes it is important to increase the feedback from operations to the development to answer questions like “Am I still fit for evolution, or am I already close to the limits?”
- **Extensibility**: Efficiently add new SW systems without affecting the installed SW base and the requirement for re-homologation.
- **Maintainability**: The SW system needs to be maintained for decades. At the same time maintainability needs to consider constant and frequent change due to the increasing trend of digitalization and emerging new technologies.
- **Portability**: The SW system needs to be portable to newly emerging HW platforms quickly and easily.
- **Testability**: Efficient and automated deployment of SW into a test environment ensures a short feedback loop back to software development.
- **Traceability**: Change needs to be traceable to manage change over the lifetime of the software, and to facilitate efficient impact analysis and swift homologation.
- **Verifiability**: Efficiently and reliably proof that the SW system fulfils regulatory and customer requirements.

The property of “minimal service outage” is addressed by the sum of the above properties.

7 CONCLUSIONS

DevOps as used in the IT-industry needs a heavy re-interpretation for the railway domain. It is not immediately applicable in its current form (see section 3.3).

In addition, DevOps as defined by ISO/IEC/IEEE 32675 neither delivers such an industrial interpretation but only high-level definitions. It lacks concrete technologies/tools etc. which must be defined for a railway specific application.

WP29 cannot apply DevOps to all railway domain processes. WP29 will focus on the SW engineering process guided by architectural principles (section 3.1) and scenarios/aspects derived from the user stories (section 5.2).4.

Architectural drivers from DevOps's current application domains (mostly IT) differ from those in the railway industry. E.g., micro services architectures support IT industry's drivers quite well, while using micro services in the railway domain needs an in-depth analysis whether the different/additional drivers are equally well served, and constraints can be met.

REFERENCES

[1] Chris Richardson, Microservices patterns, https://microservices.io/book.

[2] Gene Kim, Jez Humble, Patric Debois, John Willis, Nicole Forsgren, The DevOps Handbook, Second Edition, https://itrevolution.com/product/the-devops-handbook-second-edition/

[3] Matthew Skelton and Manuel Pais, Team Topologies, https://teamtopologies.com/