Requirements Engineering in Rail Transit Production: an Experience Report

Fernanda Buonanno*, Domenico Di Leo†, Paolo di Paolo*, Roberto Pietrantuono†, Stefano Russo †‡
*AnsaldoBreda S.p.A., Via Argine, 425, 80147 Napoli Italy
†DIETI, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125, Napoli, Italy
‡Critiware s.r.l., Incipit, Via Cinthia, Complesso Univ. Monte S. Angelo, 80126, Napoli, Italy
{*buonanno.fernanda,dipaolo.paolo*}*@ansaldobreda.it*, {*domenico.dileo,roberto.pietrantuono,sterusso*}*@unina.it*

***Abstract*—Software is an increasing part of train control systems, calling for the integration of sound software design techniques into consolidated industrial systems' engineering processes. Although requirements engineering is a traditional software engineering area, its relevance for critical embedded systems is underestimated. We present the experience of a public-private collaboration between University of Naples and Ansaldo Breda, a leading company in the field of rail transit systems. The experience is focused on requirements engineering as a driver to improve the development process in order to better support, in the long term, software quality and safety assurance activities, at the same time with a proper cost/quality trade-off (higher quality costs are compensated through reuse over a product line).**

I. Introduction

Software has become a non-negligible part of on-board train control equipments. This demands for innovation in system engineering processes in the field, with the aim of giving evidence of control software quality and safety levels, while reducing costs. We report the experience within a public-private collaboration between the University of Naples and Ansaldo Breda (AB), a leading company in the market of rail transit systems, with a product line of technologically advanced rolling stocks[1]. Companies like AB are called to build high-quality control software, compliant to reference standards of railway and software industry (such as CENELEC 50128 [1], IEEE 830), with contained cost and within stringent time to market. This pushes to explore a modernization of the internal software development life cycle to strengthen key aspects like reusability and software integrity levels assurance.

The experience we describe focuses on the integration of requirements engineering [2] techniques into AB's consolidated industrial process. The cooperation started in 2010 and is continuing through the Critiware spin-off company[2]. Our experience highlights the role of proper requirements engineering as main driver for addressing the issues related to development cost, product quality, and standard compliance. Indeed, requirements have "*a crucial importance ... in critical software systems engineering*" [3]; their importance is however underestimated, if we just admit that - as pointed out again in [3] - "*most tools in common use today still represent a requirement as a simple, unadorned string*".

[1]www.ansaldobreda.it
[2]www.critiware.com

II. Main issues and proposed solutions

A. System overview

We focused on the *MLA* (Automated Light Metro) product line, namely what UNI 8369 defines as *Light Rail Transit*[3]. This AB's rail transit platform has been deployed in several sites all over the world, such as Riyadh (SA), Milan and Brescia (I), Saloniki (GR), Taipei (RC). The main subsystems of this platform are: **Train Control System (TCS)**, **Traction Control Units (TCUs)**, and **Monitoring and Diagnostic System (MDS)**.

The TCS is the on-board train automation unit managing logic functions, contactors and electrovalves during the traction, braking and coasting status. The MDS is the on-board unit which collects and displays diagnostic data from each subsystem of the train. The TCU is the on-board controller of the propulsion system with capability to manage propulsion inverters and braking choppers, and to move the dedicated contactors. Communications among entities of the vehicle is carried out through dedicated buses (Multifunctional Vehicle Bus, MVB), a serial RS485 bus, and an Ethernet bus. These units have a considerable part of software, e.g., the TCS sizes up 10,000 LoC. Fig. 1 depicts the architecture of the TCS.

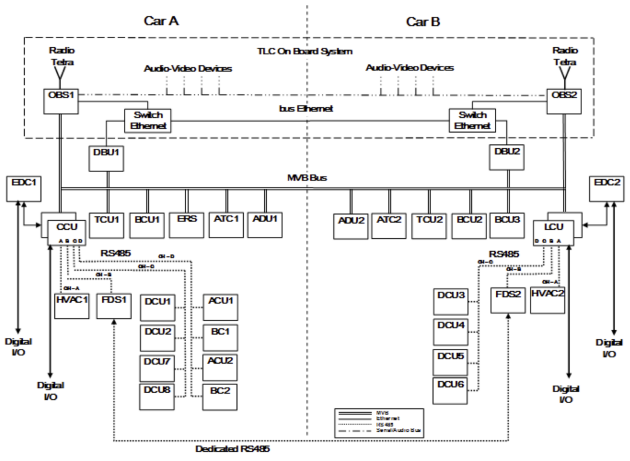

Fig. 1. The Architecture of the TCS.

[3]http://store.uni.com/

B. Development process

The AB software development process adheres to the V-Model (Fig. 2), prescribed by the CENELEC EN 50128 standard [1]. This model, adopted in several critical industrial domains, has key benefits in accounting for verification and validation (V&V) at early stages, as soon as requirements are elicited, and supporting model-based V&V [4].

Requirement engineering starts with user requirements described in the *contractual documents* with the customer, containing technical specifications and references to standards and norms to be respected. From these, system requirements are first specified, and a high-level system architecture is devised. A primary work of the design team is the allocation of system requirements to subsystems, and the derivation of sets of subsystem requirements. Then, the team identifies which tasks should be realized by software, ending up with the *software requirements specification* (SRS) for each subsystem. SRSs are used by designers to derive the software architecture, which is finally used, along with requirements, as input for the other phases of the life cycle process.

C. The impact of requirements

The innovation needs initially identified were a rapid identification of reusable modules of software artifacts across instances of the MLA product line, and the improvement of traceability of requirements, so as to favour reuse, as well as an easier compliance with the certification standards. As we jointly started to analyze the development process from the upper system level down to the code, we realized that the primary area of intervention to achieve both goals should have concerned requirements specification and management. The principal warning was the high heterogeneity of requirements in terms of writing style, structuring, classification, granularity, relations, and semantics (i.e., separation of *what* from *how*).

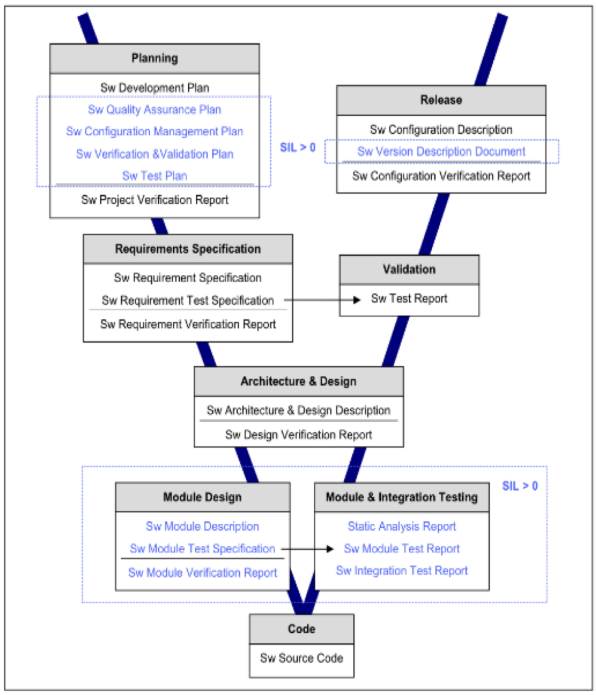

Fig. 2. The company "V-model" for software development.

Differences in specifying requirements may be due to several reasons. A relevant one in our case is related to the different stakeholders involved, which include - besides the AB engineers: the customer requiring the train; the partner railway company Ansaldo STS, which for some projects acts as main contractor, and whose requirements refer to the whole rail transportation system (including signaling); external consultants supporting AB in control software development.

In this context, system requirements, expressed in natural language, of an MLA project is the result of: *i)* user needs, expressed in the contractual documents, properly analyzed and specified at system level; *ii)* standards rules and regulations to be respected; *iii)* more basic, stable, and "reusable" requirements coming from domain knowledge and consolidated practices within the company; *iv)* constraints coming from COTS (Commercial Off-the-Shelf) component providers.

The different points of view often result in relevant differences not only in the *style* by which a requirements is expressed, but also in the meaning of requirements, in their abstraction level, and in their granularity. This often leads to consistency problems among requirements and ambiguity when read from different actors in the development chain.

Moreover, the coarse-grained way requirements are typically specified by system engineers does not support "design for reuse". This pushes development engineers to proceed, practically, by starting from the most resembling project, instead from the platform, and applying customizations; in this way the product becomes a "customization of customizations", with some parts derived from similar projects, some other parts derived from the platform.

Issues in requirements specification lead to problems of reusability, traceability (up to code), testability (each specification problem reflects into a problem in the test specification), and maintainability (e.g., change and impact analysis are more difficult with non-homogeneous requirements specifications). All these issues result into high costs of customization of the platform to user needs and, more importantly, may affect confidence in the evidences of the satisfaction of non-functional requirements, including safety. For instance, testability and traceability (which is highly recommended by certification standard [5], [6]) are essential goals in this domain.

D. Improvement Actions

In order to improve quality, traceability and reusability of requirements across the product line, first we envisaged their organization into platform-level, independent from a specific project, and product-specific requirements, obtained through a customization of the former according to contractual specifications. Then we defined procedures and practices for requirements classification, structuring, traceability, and management, experimented on an internal pilot case study, namely

an instance of the MLA system. In particular, we acted on:

- a classification of requirements per typology;
- a clear separation of abstraction levels, so as to favour the distinction of *what should be specified where*, and the consequent identification and respect of roles;
- rules for a structured use of natural language, with a common representation for all the types of requirements (system, subsystem, interface, data, software), previously left to the individual decision of project managers;
- a proper granularity of requirements' textual description;
- using terms and definitions, in all artifacts, consistently among different projects.

The main implemented actions are the following.

Abstraction. The difficulty in reusing requirements was mainly due to the intersected customizations and the absence of common abstraction criteria across different projects. Hence we deemed important to proceed with step-by-step refinements in specifying requirements [2]. We suggested distinguishing requirements at different abstraction levels - high-level and low-level requirements (HLR, LLR) - in order to separate the *platform-level* (more abstract) and *project/product-dependent* (more specific) set of requirements, and derive the latter from the former by customization. Table I shows an example of HLR with a coresponding LLR for the TCS subsystem.
This choice also favours the separation of concerns and responsibilities (since a different expertise is required for the two types of requirements): at higher level, a broader system-level and cross-project knowledge is required, to specify requirements in a way that can be potentially reused in other projects, whereas at lower level a much closer view to the technical perspective of the specific subsystem being realized, as well as to the programming domain, is required. A better separation between *what* is required from *how* it is realized in a specific project is also obtained in this way.

Structure of a requirement. We defined a tailored scheme to structure the requirements. We included the fields: *ID, Name, Description, Input, Output, Precondition, Postcondition, Type (functional, performance, interface)* (Table I).

Relations among requirements. A high-level functional requirement represents a functionality requested to the system, that will be expressed as a flow of actions to perform. We encountered a number of situations in which a requirement expressed a basic functionality with successive requirements expressing "variations", or "specialization" detailing it. Therefore we found it beneficial to introduce of a *hierarchical* relation among requirements. Note that this is different for abstraction levels; we can have a hierarchy between requirements at the same level of abstraction, i.e., indicating both details close to the implementation but, for instance, indicating a normal and an abnormal scenario.

We have formalized further relations among requirements, that connect requirements referring to different subsystems. For instance, it happens that a requirement for subsystem A describes the interactions with subsystem B; a reference in A's requirement should be put in the requirement for B in order to trace the dependence relationship.

TABLE I
STYLE OF HIGH- AND LOW- LEVEL REQUIREMENTS.

Attribute	*HLR*	*LLR*
ID	SHR-TCMS-01	SLR-TCS-01
Type	Functional	Functional
Link to	SYS-REQ01	SHR-TCMS-01
Name	Traction release by TCS	Traction release by TCS
Preconditions	...	...
Input	Coasting and Propulsion Reset commands by ATC	Coasting and Propulsion Reset commands on MVB by ATC through dataset 707/708
Output	Coasting commanded, service brake released, and breaking train line set to off. BCU shall send back the status of service and holding brake.	Coasting $->1$ on MVB, Service Brake $->0$, Train Line $->1$. BCU shall send back the status of service and holding brake.
Postconditions	...	...

Linking artifacts. Traceability establishes the (bidirectional) relationships among artifacts of the development process [5], [6]. We implemented traceability relations among elements at different level of abstraction, both vertically and horizontally.

Considering the separation in levels of abstraction introduced, we have relations between:

- Each system functional requirement with the interface requirements possibly mentioned in the text;
- Each high-level software requirement with the corresponding(s) system requirements that it implements;
- Each high-level software requirement with other HLRs of other subsystems that it mentions in the text;
- Each low-level software requirement with the corresponding(s) HLR that it implements;
- Each low-level software requirement with other LLRs of other subsystems that it mentions in the text.

This way, a requirement needs to specify only what concerns its level of abstraction and its subsystem, citing with a link actions that are described elsewhere. This prevents from having in the same document requirements with very different levels of abstractions, or mixing interface with functional requirements, or with other subsystem's requirements.

Form and style. We introduced a set of rules to specify requirements unambiguously, with the same form (e.g., verbs in active form, sentences shorter than a limit, using common meaning of terms, using properly terms like *shall*, *must*, *should*, avoiding qualitative terms, decompose complex requirements in more one-goal requirements, etc.)

Tool support. The usage of a requirements management tool, namely IBM Rational DOORS[4], by all actors and across all development phases was promoted. Through knowledge transfer and training activities within the collaboration, the adoption of the tool has been consolidated; it is currently being used with success, providing support for specification, traceability, impact analysis, sharing (acting as common requirements repository) and reuse.

[4]www-03.ibm.com/software/products/us/en/ratidoor/

E. Benefits

The major observed benefits from the implemented actions can be grouped as follows.

Reusability. The separation of abstraction levels, the correct identification of hierarchical relations and linking among requirements, and the common structure and classification criteria for specification definitely help in separating out platform-level from project-specific needs, easing the implementation of a shared product line requirements database. The expectation is that next projects can be developed by simply customizing the baseline, and that new common features can be seamlessly specified at platform level.

Testability. A well defined organization of requirements eases the derivation of test specifications and test cases. Specifically, it is possible to apply specification-based testing techniques with a contained efforts since key elements of the test specification such as the inputs and test dependencies are easy to identify. Furthermore, the usage of a requirements management tool allows the derivations of test specification partially automatically, thus supporting the test designer in repetitive and likely error-prone tasks. Test specification support baed on requirements is an in-progress activity.

Traceability. Traceability is crucial for many quality control and assurance activities. The definition of clear items subject to traceability is fundamental. Defining a homogenous form for requirements (in size, style, structure, classification, level of abstraction), and vertical and horizontal relations with other requirements, make tracing trees clear to understand and much less prone to mistakes. Starting from this, traceability can be implemented and verified up to source code files and test cases for each requirement.

Maintainability. Actions are expected to bring further benefits from the perspective of maintenance. In fact, the implementation of links and relations among requirements and the introduction of a support tool ease the impact analysis whenever a change occurs, identifying requirements to be taken care of and regression tests to carry out.

Automation. The systematic usage of a requirements management tool allows certainly to reduce cost of performing repetitive tasks; in the case of AB, the automation support has been extended through the implementation of plugins to enhance the linking between different phases of the lifecycle. For instance, several scripts have been implemented with various goals, such as: to relate entries in the issue tracker with the requirement in DOORS, to synchronize files in the software versioning and revision control (SVN) system with objects in DOORS, to generate IEEE-compliant SRSs, to automatically generate a *Software Change Record* whenever a change is required through the issue tracker.

We expect all above benefits to be amplified by two further ongoing innovations. The former, technological, concerns the integration of the several different support tools involved - for requirements management, model creation, code generation, testing, issue tracking. For instance, the (possibly custom) integration of DOORS with the company testing tools (besides SVN and tracking tools) will clearly ease traceability and maintenance activities (including issues and change management). The second innovation, methodological, is the introduction of model-driven engineering (MDE) techniques in the company V-model. Similarly to what described in [4] for the air traffic control domain, MDE will support automation in test cases specification and generation from the requirements, early verification, and traceability.

III. Lessons learnt and next steps

For critical embedded systems, software requirements engineering tremendously impacts all development phases, as well as V&V activities (e.g., specification based testing, unit testing, integration testing). Badly specified requirements not only lead to unintended software, but also to ineffective testing. Sound requirements engineering practices are essential if the embedded software is meant to undergo a certification process.

In a traditional industrial setting, where software is part of very complex systems - although an increasing part - the consolidation of sound requirements management practices may take a long time and requires the full involvements of all the actors. Unfortunately, it may not be possible to have the commitment of all actors at the same time; in our experience, the definition of adequate company practices benefited from a progressive refinement over time. The V&V team plays an important role in this sense, as it can devise testing activities as soon as the requirements are available, and can give valuable feedback to proper requirements specification practices in the application field. Moving towards common agreed platform requirements, automation support, structured use of natural language, and traceability, will definitely provide valuable results in terms of software quality and integrity level assurance.

Our next steps will concern the improvement of the test (cases) specification process starting from requirements, so as to customize the V-model also along its "right" branch through model-driven testing techniques.

Acknowledgment

This work has been partially supported by the project CECRIS (CErtification of CRItical Systems) - FP7 - Marie Curie (IAPP) number 324334 (www.cecris-project.eu) - and by the project "Embedded Systems in Critical Domains" (CUP B25B09000100007).

References

[1] CENELEC EN 50128, Railway applications - Communications, signalling and processing systems, *http://www.cenelec.eu*

[2] E. Hull, K. Jackson, J. Dick. Requirements Engineering, 3rd ed., 2010. Springer-Verlag.

[3] X. Larrucea, A. Combelles, J. Favaro, Safety-Critical Software (Guest editors' introduction), IEEE Software, 30 (3), May/June 2013.

[4] G. Carrozza, M. Faella, F. Fucci, R. Pietrantuono, S. Russo. Engineering Air Traffic Control Systems with a Model-Driven Approach, IEEE Software, 30 (3), May/June 2013.

[5] DO - 178B. Software Considerations in Airborne System and Equipment Certification.

[6] C. Esposito, D. Cotroneo, N. Silva. Investigation on Safety-Related Standards for Critical Systems. Proceedings 1st International Workshop on Software Certification, WoSoCER 2011, IEEE CS Press.