Journal of Systems Architecture 77 (2017) 83–100

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Automotive Architecture Framework: The experience of Volvo Cars☆

Patrizio Pelliccione[a,*], Eric Knauss[a], Rogardt Heldal[a,c], S. Magnus Ågren[a], Piergiuseppe Mallozzi[a], Anders Alminger[b], Daniel Borgentun[b]

[a] Chalmers University of Technology | University of Gothenburg, Department of Computer Science and Engineering, Sweden
[b] Volvo Cars, Sweden
[c] Bergen University College, Norway

ARTICLE INFO

Article history:
Received 21 September 2016
Revised 10 February 2017
Accepted 18 February 2017
Available online 21 March 2017

Keywords:
Architecture framework
Software architecture
Automotive domain
Systems of Systems
Continuous integration and deployment
Automotive ecosystem

ABSTRACT

The automotive domain is living an extremely challenging historical moment shocked by many emerging business and technological needs. Electrification, autonomous driving, and connected cars are some of the driving needs in this changing world. Increasingly, vehicles are becoming software-intensive complex systems and most of the innovation within the automotive industry is based on electronics and software. Modern vehicles can have over 100 Electronic Control Units (ECUs), which are small computers, together executing gigabytes of software. ECUs are connected to each other through several networks within the car, and the car is increasingly connected with the outside world. These novelties ask for a change on how the software is engineered and produced and for a disruptive renovation of the electrical and software architecture of the car. In this paper we describe the current investigation of Volvo Cars to create an architecture framework able to cope with the complexity and needs of present and future vehicles. Specifically, we present scenarios that describe demands for the architectural framework and introduce three new viewpoints that need to be taken into account for future architectural decisions: Continuous Integration and Deployment, Ecosystem and Transparency, and car as a constituent of a System of Systems. Our results are based on a series of focus groups with experts in automotive engineering and architecture from different companies and universities.

1. Introduction

Today's automotive industry is concerned with much more than the traditional view of assembled mechanical parts controlled by a human for transportation. During the past twenty years vehicles have become more and more robot like, interpreting and exploiting input from various sensors to make (semi-)autonomous decisions and finally commit actions that were previously made by humans.

Accordingly, the role of software is continuously changing. Initially, it was introduced in cars to optimize the control of the engines. Since then, the growth of software within the car has been exponential for each year and today not a single function is performed without the involvement of software. 80%–90% of the innovation within the automotive industry is based on electronics [1], as mentioned for instance by Peter van Staa - Vice-President Engineering of Robert Bosch GmbH at the European Technology Congress in June 2014 [1]. A big part of electronics is software.

Considerable parts of the software is safety-critical, with human life at stake if the system is not performing as expected. Thus, the focus is gradually switching over from human control of the mechanics to software and electronics supporting decision-making and even taking over the control. The development is similar to the more popularized history of air plane development and the invention of auto pilots and "fly by wire", with some striking differences. The higher complexity of the environment where the vehicle moves has slowed down the development of automated vehicle control. The most advanced cars have more or at least comparable amounts of software than fighter airplanes.[1] Vehicles are also

☆ This work was partially supported by the NGEA and NGEA step 2 Vinnova projects led by Volvo Cars and by the Wallenberg Autonomous Systems Program (WASP).

* Corresponding author.
E-mail addresses: patrizio.pelliccione@gu.se, patrizio.pelliccione@gmail.com (P. Pelliccione), eric.knauss@gu.se (E. Knauss), heldal@chalmers.se (R. Heldal), magnus.agren@chalmers.se (S. Magnus Ågren), mallozzi@chalmers.se (P. Mallozzi), anders.alminger@volvocars.com (A. Alminger), daniel.borgentun@volvocars.com (D. Borgentun).
URL: http://www.patriziopelliccione.com (P. Pelliccione)

http://dx.doi.org/10.1016/j.sysarc.2017.02.005
1383-7621/

[1] As said by Alfred Katzenbach, the director of information technology management at Daimler:http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code.

produced in higher volumes than airplanes, which puts harder requirements on the technology cost.

This evolution of the automotive industry creates new challenges regarding the electrical and software architecture development and maintenance. The architecture of a modern car has to cope with a large amount of concerns, including safety, security, variability management, networking, costs, weight, etc. Also, the increasing amount of people involved in the software development projects imposes additional challenges to the architecture teams, as the development and design literally cannot be controlled, or even understood, in detail by a single group any more. The development is inevitably parallelized; this obviously also holds for the large amounts of externally developed software, which is integrated as black box functionality. The important integration work is done in an iterative manner by developers and test teams, focusing on vital systems first and then gradually establishing various functionalities. Architects might be not involved in integration, however, the architecture for sure influences the integration.

In this paper we report a current initiative of the Volvo Cars to renovate the electrical architecture. The work is part of a Vinnova FFI Swedish project, called Next Generation Electrical Architecture (NGEA),[2] which mainly focuses on the following areas: (i) continuous integration and deployment; (ii) cars as constituents of a System of Systems; (iii) reducing the number of ECUs through an architecture that allows the identification of key functions to be implemented in domain nodes; and (iv) strategies to improve the automotive ecosystem so to enable rapid communication with suppliers and flexible development. The reason for the NGEA project to choose these topics were that they are strategically important for automotive industry and it is important to find a way of handling them. In this paper we largely extend beyond a short paper [2]: the paper in [2] only sketches preliminary results.

Within the project we are investigating the possibility to create a Volvo Cars Architecture Framework. We believe that an architecture framework [3], together with its multiple viewpoints, is the instrument to manage the increasing complexity of modern vehicles. It aims at ensuring that descriptions of vehicle architectures can be compared and related across different vehicle programs, development units, and organizations, thus increasing flexibility and innovation, while reducing development time and risks. However, the definition and description of an architecture require a cost in terms of human and financial resources. Moreover, sometimes it is not evident that investing effort and money on properly defining and describing the architecture will result in saving money later in the development. For instance, as described later in the paper, we identified discrepancies between the as-intended and the as-implemented architecture. This motivates the shift towards "just-in-time" architecting and on continuous integration and deployment.

We build on existing architecture frameworks in the automotive domain, i.e., [4,5] and we base our work on the conceptual foundations provided by the ISO/IEC/IEEE 42010:2011 standard [3]. The standard addresses architecture description, i.e. the practices of recording software, system and enterprise architectures so that architectures can be understood, documented, analysed, and realized.

The paper is organized as follows. Section 2 introduces the concept of architecture framework and explain a template to document architecture frameworks. Section 3 gives an overview of the state of the art on architecture framework in the automotive domain. Section 4 analyses the state of practice in the context of Volvo Cars and reports some of the lessons learned. Section 5 introduces the architecture framework we are defining within Volvo Cars in the context of the NGEA project, including an overview of stakeholders as well as critical scenarios that define demands and expectations towards the architecture framework. Sections 6–8 detail three new and challenging viewpoints of the framework, namely the *Continuous Integration and Deployment* viewpoint (Section 6), the *Ecosystem and Transparency* viewpoint (Section 7), and the *System of Systems: vehicle point of view* viewpoint (Section 8). Finally the paper concludes in Section 9 with final remarks and a discussion about future work.

2. Architecture framework

Within this paper we refer to the definition of architecture suggested by the ISO/IEC/IEEE 42010:2011 standard [3], which defines the architecture as: "*< system> fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*". Therefore, with the term architecture we are not referring to a specific view of the system and since we are focusing on the automotive domain, the architecture will refer also to physical components and their behavior in an abstract way. An architecture description is a "*work product used to express an architecture*" [3].

An architecture framework is a coordinated set of viewpoints, conventions, principles, and practices for architecture description within a specific domain of application or community of stakeholders [3]. More specifically, it is determined by:

- a set of architecture-related concerns;
- a set of stakeholders holding those concerns;
- a set of architecture viewpoints which frame (i.e., cover) those concerns;
- a set of model correspondence rules to impose constraints between types of models and then demonstrate that constraints are satisfied by the architecture.

Then, an architecture framework establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community, developing architecture modelling tools and architecting methods, and establishing processes to facilitate communication, commitments and interoperation across multiple projects and/or organizations [3].

As shown in Fig. 1, an architecture framework is a prefabricated knowledge structure, identified by *architecture viewpoints*, that architects use to organize an architecture description into *architecture views*. The terms architecture view and architecture viewpoint are central to the standard [3]: *"A viewpoint is a way of looking at systems; a view is the result of applying a viewpoint to a particular system-of-interest"*. An *architecture viewpoint* encapsulates notations, conventions, methods, and techniques to be used according to specific model kinds framing particular concerns and for a particular audience of system stakeholders. The concerns determine what the model kinds must be able to express: e.g., functionality, security, reliability, cost, deployment, etc. A model kind determines the notations, conventions, methods and techniques to be used. Viewpoints, defining the contents of each architecture view, are built up from one or more model kinds and correspondence rules linking them together to maintain consistency. Viewpoints, like patterns and styles, are a form of reusable architectural knowledge for solving certain kinds of architecture description problems derived from best practices. Viewpoints have been originally proposed in the 1970s (in Ross' Structured Analysis) and then they have been refined in [6]. Architecting methods often define one or more viewpoints, e.g. [7–10].

Many existing practices express architectures through collections of models, and models are further organized into cohesive

[2] http://www.vinnova.se/sv/Resultat/Projekt/Effekta/2009-02186/Next-generation-electrical-architecture.

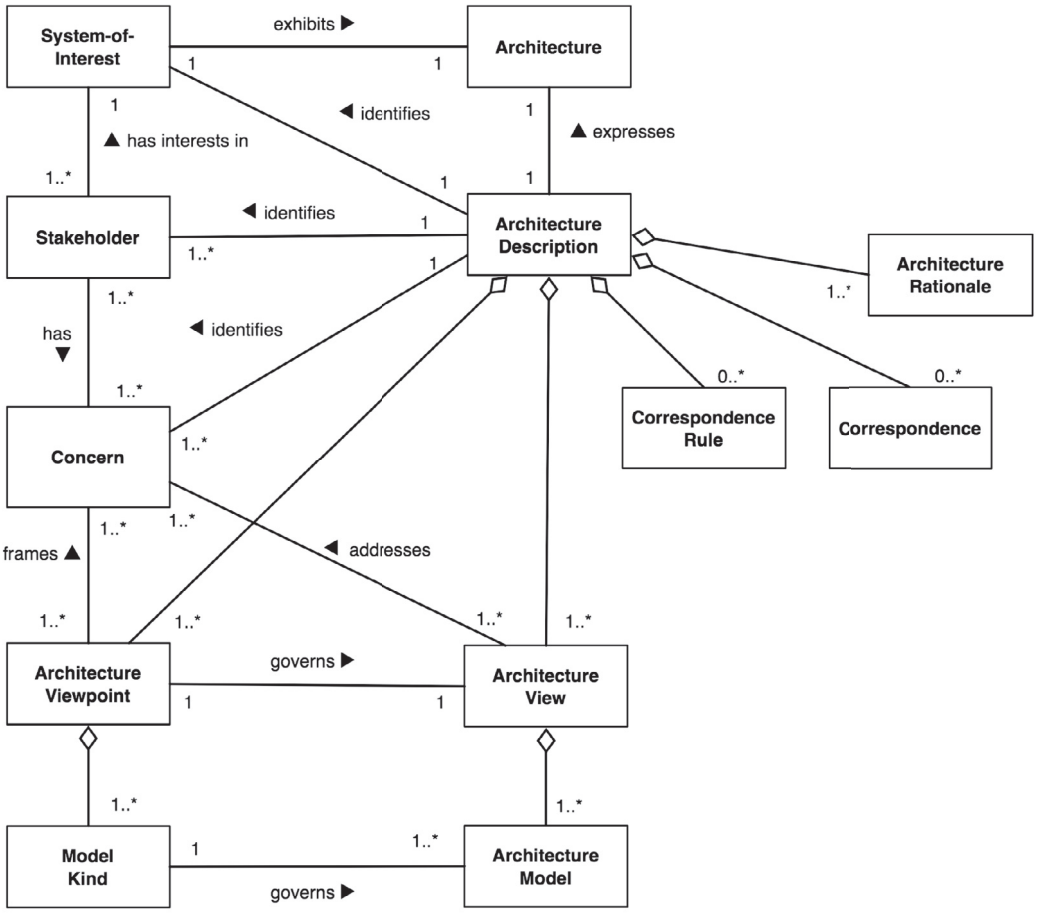

Fig. 1. Terms and concepts beyond the standard (figure taken from [3]).

groups, called *views*. A view can be defined as a "*work product expressing the architecture of a system from the perspective of specific system concerns*" [3]. As noted in the standard, the cohesion of a group of models is determined by specific concerns, which are addressed by that group of models. Viewpoints refer to the conventions for expressing an architecture with respect to a set of concerns.

For further discussion of the content model and architecture frameworks mechanism, see [11]. Recent frameworks include the ISO Reference Model for Open Distributed Processing, GERAM (Generalized Enterprise Reference Architecture and Methodology) [12], DODAF (US Department of Defense Architecture Framework) [13], TOGAF [14], and MODAF [15]. For an extensive survey of frameworks, see [16].

In this paper we use the template called *Architecture Viewpoint Template* for specifying architecture viewpoints in accordance with ISO/IEC/IEEE 42010:2011, *Systems and software engineering—Architecture description*.[3]

In the following we report the template sections and an excerpt of the guidelines; please refer to the links above for a complete description of the template:

- *Viewpoint Name*: name of the viewpoint. If there are any synonyms or other common names by which this viewpoint is known or used, record them here.
- *Overview*: Provide an abstract or brief overview of the viewpoint. Describe the viewpoint's key features.
- *Concerns and stakeholders*: Architects looking for an architecture viewpoint suitable for their purposes often use the identified concerns and typical stakeholders to guide them in their search. Therefore it is important (and required by the Standard) to document the concerns and stakeholders for which a viewpoint is intended.
  - **Concerns**: Concerns name "areas of interest" in a system. Concerns may be very general (e.g., *Reliability*) or quite specific (*e.g., How does the system handle network latency?*). Concerns identified in this section are critical information for an architect because they help one deciding when this viewpoint will be useful. When used in an architecture description, the viewpoint becomes a "contract" between the architect and stakeholders that these concerns will be addressed in the view resulting from this viewpoint. It can be helpful to express concerns *in the form of questions* that views resulting from that viewpoint will be able to answer. E.g., *How does the system manage faults?* or *What services does the system provide?*
  - **Typical stakeholders**: Provide a listing of the typical stakeholders of a system who are in the potential audience for views of this kind. Typical stakeholders would include those likely to read such views and/or those who need to use the results of this view for another task.
  - **Anti-concerns [Optional]** It may be helpful to architects and stakeholders to document the kinds of issues for which this viewpoint is *not appropriate or not particularly useful.*

[3] The template is called *Architecture Viewpoint Template*, which is released under copyright © 2012–2014 by Rich Hilliard. The template is licensed under a Creative Commons Attribution 3.0 Unported License. The terms of use are here: http://creativecommons.org/licenses/by/3.0/.

Identifying the "anti-concerns" of a given notation or approach may be a good antidote for certain overly used models and notations.

- *Model Kind name:*[4] Identify the model kind
  - *Conventions*: Describe the conventions for models of this kind. Conventions include languages, notations, modeling techniques, analytical methods and other operations. These are key modeling resources that the model kind makes available to architects and determine the vocabularies for constructing models of the kind and therefore, how those models are interpreted and used. The Standard does not prescribe *how* modeling conventions are to be documented. The conventions could be defined:
    (I) by reference to an existing notation or language (such as SADT [17] or UML [18] or an architecture description language such as EAST-ADL [19] or SysML [20]) or to an existing technique;
    (II) by presenting a metamodel defining its core constructs;
    (III) via a template for users to fill in;
    (IV) by some combination of these methods or in some other manner.
  - *Operations*: Specify operations defined on models of this kind.
  - *Correspondence rules*: Document any correspondence rules associated with the model kind.
- *Operations on views*: Operations define the methods to be applied to views and their models. Types of operations include:

  Construction methods are the means by which views are constructed under this viewpoint. These operations could be in the form of process guidance (how to start, what to do next); or work product guidance (templates for views of this type). Construction techniques may also be heuristic: identifying styles, patterns, or other idioms to apply in the synthesis of the view.

  Interpretation methods which guide readers to understanding and interpreting architecture views and their models.

  Analysis methods are used to check, reason about, transform, predict, and evaluate architectural results from this view, including operations which refer to model correspondence rules.

  Implementation methods are the means by which to design and build systems using this view.
- *Correspondence rules*: Document any correspondence rules defined by this viewpoint or its model kinds. Usually, these rules will be across models or across views since constraints within a model kind will have been specified as part of the conventions of that model kind.
- *Examples [Optional]*: Provide helpful examples of use of the viewpoint for the reader (architects and other stakeholders).
- *Notes [Optional]* Provide any additional information that users of the viewpoint may need or find helpful.
- *Sources*: Identify sources for this architecture viewpoint, if any, including author, history, bibliographic references, prior art, etc.

In this paper, we apply the viewpoint template above to describe three new viewpoints that we propose. At the current state of our work, many details still need to be evaluated, thus we for example share model kind candidates as well as our experience on what kind of information these model kinds would need to provide, rather than presenting a clear definitions of the modelling notations to be used.

[4] In the Standard, each architecture view consists of multiple architecture models. Each model is governed by a *model kind* which establishes the notations, conventions and rules for models of that type.

3. State of the art in Automotive Architecture Frameworks

Architecture frameworks have not been standardized in the automotive domain and automotive industry. However, some attempts exist and different types of architecture viewpoints and views have been introduced recently as part of Automotive Architecture Frameworks.

3.1. Automotive Architecture Framework (AAF)

A first attempt towards a standardized architectural foundation and automotive-specific architecture framework is the Automotive Architecture Framework (AAF) proposed in [4]. The purpose of AAF is to describe the entire vehicle system across all functional and engineering domains and drive the thought process within the automotive industry. AAF conforms to the ISO 42010 international standard [3], and therefore it is defined in terms of a set of viewpoints and views.

AAF distinguishes between two sets of architecture viewpoints and views: (i) mandatory and general viewpoints and (ii) optional viewpoints. The following viewpoints are presented according to the viewpoint catalog in [8]. The mandatory viewpoints and their respective views include: (i) Functional viewpoint - functional decomposition, functional architecture; (ii) Logical viewpoint - logical decomposition, logical architecture; this viewpoint is only mentioned in [4] but not detailed in [21]; (iii) Technical viewpoint - physical decomposition, technical architecture; (iv) Information viewpoint - perspective of information or data objects used to define and manage a vehicle; (v) Driver/vehicle operations viewpoint - vehicle environment; (vi) Value net viewpoint - OEM stakeholders and those of its suppliers and engineering partners. The optional viewpoints suggested by the AAF are: (i) Safety, (ii) Security, (iii) Quality and RAS - Reliability, Availability, Serviceability, (iv) Energy, possibly including performance, (v) Cost, (vi) NVH - noise, vibration, harshness, and (vii) Weight.

3.2. Architectural Design Framework

The Architectural Design Framework (ADF) [22] is developed by Renault and includes operational, functional, constructional, and requirements viewpoints. Although AAF and ADF are related they have some differences.

- *Operational Viewpoint* - this is the more abstract viewpoint. The system is observed from a black box and user perspective [22].
- *Functional Viewpoint* - system functions identified in the views associated to the operational viewpoint are allocated to SysML Blocks.
- *Constructional Viewpoint* - this viewpoint describes a further allocation (or grouping) of system functions into physical components.
- *Requirements Viewpoint* - this viewpoint is orthogonal to other viewpoints. Each requirement view has a relationship with views of other viewpoints. The suggested process is quite waterfall-ish. Stakeholder requirements are built at the very beginning after the identification of actors and system boundary. These requirements are the starting point for identifying High-Level Requirements. The Technical Requirement view is typically built after the whole set of Operational Views is available.

3.3. Architecture Framework for Automotive Systems

The Architecture Framework for Automotive Systems (AFAS) is proposed in [5] through an analysis of AAF, ADF, and of Architecture Description Languages [23,24] tailored to automotive domain, like EAST-ADL [19] and AML [25]. It contains an elaboration and

unification of the viewpoints proposed in AAF and ADF and then proposes additional viewpoints, e.g.:

- *Feature viewpoint* - to be used to support the product line engineering.
- *Implementation viewpoint* - it describes the software architecture of the Electrical/ Electronic (E/E) system in the vehicle [19].

4. Software development challenges within Volvo Cars

In a previous paper we studied within Volvo Cars, from the architecture point of view, the challenges that OEMs are facing in the last years [26]. To better understand some of the organizational issues with having different parts of the organization responsible for different parts or layers in the architecture, we decided to conduct nine in depth interviews with focus on the roles of architects and how organizational factors affect them. These interviews have been carried out at Volvo Cars and Volvo Group Truck Technology (VGTT), extending the knowledge about these specific companies, hence providing a detailed understanding of two independent but nevertheless similar automotive companies. In [26] we followed a lightweight grounded theory-type approach since we started from a general research question and we involved few people in order to collect more information. Based on the analysis of the first data and on the first emerging ideas from the data, we then refined the research questions, carefully planned the study and accordingly selected the people to be interviewed. In addition to initial workshops in which we collected initial ideas, we used interviews as data generation methods. Specifically, we used semi-structured interviews: we defined a list of themes to be covered and questions we aimed at asking. However, in some interviews we changed the order of questions according to received answers and to the flow of the "conversation". Each interview was around one hour long, we collected field notes and recorded audio. Each interview begins with introduction, clarification about the purpose of the study, asking permission to record and giving assurances of confidentiality of the information. More information about the research method can be found in [26].

4.1. Gap between prescriptive and descriptive architecture

We identified that there is not always an obvious connection between architecture (or top-level design) and design; it seems also that this connection vanishes over time, once development requires changes on the system design. The architecture is communicated as large documents, which are supposed to be read by stakeholders. However, this not always corresponds to the reality. Volvo Cars works also in cross functional teams and other type of communications are also used to communicate the architecture. More analysis is needed to understand the root of the problem. Moreover, maintenance of the architecture, while the design evolves, is demanding and not always performed in all parts. This shows a discrepancy between the planned architecture defined according to a V-Model process, and the architecture that is actually emerging from the system development.

We identified that the architecture has a temporal aspect, that means: at any given point in time the system has only one architecture, however, the architecture will change over time. We call as *prescriptive architecture* the architecture that captures the design decisions made prior to the system's construction.[5] This architecture is the as-conceived or as-intended architecture. We can call as *descriptive architecture* the architecture that describes how the system has been built. This architecture describes the as-implemented or as-realized architecture. We observed that there is often a discrepancy between the as-intended and the as-implemented architecture. This causes what is called *architecture degradation*. The degradation might show up in two different ways: (i) *architectural drift* when the descriptive architecture includes changes that are not included in, encompassed by, or implied by the prescriptive architecture, but which do not violate any of the prescriptive architecture's design decisions; (ii) *architectural erosion* is the introduction of architectural design decisions into a system's descriptive architecture that violate its prescriptive architecture.

When looking at the causes of architecture degradation, we can summarize the various reasons in the following three items:

1. During the development some important directives of the as-intended architecture have been violated due to time constraints, mistakes, misunderstanding, etc.
2. Some of the architectural choices of the as-intended architecture were based on assumptions (might be implicit) that then were identified as imprecise or wrong,
3. Some of the architectural choices of the as-intended architecture were made under uncertainty and during development these choices were judged as suboptimal.

4.2. Organization and process challenges

The first item of the causes of architecture degradation discussed in the section above (Section 4.1) might be solved by improving communication as mentioned before or by finding ways to guarantee the preservation of the important directives of the as-intended architecture that should not be violated for any reason.

On the organizational side, we found the need of improving the communication between different groups, for instance by making teams more cross-functional. Today there are several levels between architects and designers/developers and at some of these levels the connection is not very tight.

On one side this seems to be unavoidable since the company is big and there is the need of structuring the organization in sub-organizations (departments, groups, or teams). However, there is the risk that sub-organizations will grow up independently and decoupled from each other; this might cause silos thinking. For instance, this might lead designers/developers to think that the architects are sitting in their cloud above, without having any connection to the reality. Architects might also feel a frustration because they are not aware of everything that is happening; as a consequence, a big part of their work is to just keep up with what is happening in the construction groups. Espousing the terminology in [23,28], architects should be both "Introvert" architects (conceptually related to the internal focus of [29]), i.e., focused on more constructive work and definition of the architecture, and "Extrovert" architects (conceptually related to the external focus of [29]), i.e. devoted to communicating the architectural decisions and knowledge to the other stakeholders.

The different organizations have different competencies, attitude, and characteristics. However, new problems emerge. In fact, we can confirm that we found many architecture antipatterns. We found the *Goldplating* antipattern [29] since architects seem to be not really engaged with developers. They are doing a good technical job, however, their output is not really aligned with the needs of the developers and in the end they are often ignored. Another antipattern that we found is the *Ivory tower* [29]: the architecture team looks isolated sitting on a separate floor from the development groups and do not engage with the developers and the other stakeholders on a daily basis. This creates tensions in the organization.

It emerges the need to explore both organizational and technical possibilities for tighter cooperation between architecture lev-

[5] A discussion in depth about prescriptive and descriptive models might be found in [27].

els, and to measure effects such improvements would have. On the technical side, one partial solution might be to define specific viewpoints and to automatically generate corresponding architecture views from the design. Each of the views should focus on showing only what is relevant for respective stakeholders. Moreover, both architects and designers/developers need ways to perform early validation of their solution and to sketch and try different visions of how the future systems should look like; this will permit to understand the effect of design decisions on the architecture. As mentioned before this can be only a partial solution since it cannot solve the architecting problem since this solution only focus to visualize through specific viewpoints and views something that already exist. Other solutions are needed to solve the support just-in-time architecting as mentioned above, as well as to enable stakeholders different from the architects, such as developers, to improve the architecture when actually needed.

4.3. Towards "just-in-time architecture"

The second and third items of the causes of architecture degradation discussed in Section 4.1 call for a "*just-in-time architecture*" or agile architecting [30] as well as to enable stakeholders different from the architects, such as developers, to improve the architecture, e.g. by fixing wrong assumptions or making decision deliberately postponed.

One hypothesis made from some practitioners is that when developing large and complex systems "a clear and well-defined architecture facilitates and enables agility". This hypothesis implies that some upfront specification is needed when building complex products like cars. However this hypothesis is not completely true when the product to be realized is not clearly defined and companies want to go fast to the market (as for example observed by Waterman et al. [31]. In these situations, modifiability, support for evolution, etc. are not really main aspects to be considered. The hypothesis seems to be true when the product is well-defined. Another aspect to be considered is that agile calls for refactoring, however refactoring often is not performed since the priority is given to what should be realized.

4.4. Towards a software ecosystem perspective

Another interesting finding is that the architecture is not clearly considering the highly complex supplier-network that characterizes automotive engineering.

In a previous study [32,33] we found that a holistic strategy for aligning work across the value-chain is currently missing. Specifically, mixing commodity and differentiating components lead to sub-optimal situations, resulting in duplicated work (an observation in line with [34]). We argue that automotive architecture needs to assume a holistic perspective with respect to the whole value-chain and optimize the architecture for facilitating beneficial subcontractor interaction.

- *Commodity Components* require clearly defined technical and organizational interfaces. The goal is to develop them as efficiently as possible, thus reducing coordination overhead. Ideally, of-the-shelf commodity components can be integrated with minimal adjustment.
- *Differentiating Components* should be developed as independent from the commodity components as possible, probably in-house.
- *Innovative Components* naturally require coordination and iterative work between a number of partners. Proper communication means should be established in order to effectively stimulate and develop innovative behaviours.

In traditional software engineering, a software product is often the result of an effort of a single independent software vendor, investing into creating a monolithic product [35,36]. Modern software engineering strongly relies on components and infrastructure from third-party vendors or open source suppliers [35,36]. The emergence of Software Ecosystems (SECOs) is a recent development within the software industry [36,37]. It implies a shift from closed-organizations to open structures where external actors become involved in the development to create competitive value [35,37].

Based on the ecosystem classification model [35], we understand the automotive value chain as an ecosystem of cross-organizational collaborations among automotive suppliers [38]. In this ecosystem, the Original Equipment Manufacturer (OEM) is in the role of the ecosystem coordinator. The ecosystem is privately owned, and participation to it is based on a list of screened actors.

The automotive ecosystem is characterized by relying heavily on complex supplier networks, and strong dependence on hardware and software development [38]. Due to the increasing number of networked components, a level of complexity has been reached which is difficult to handle using traditional development processes [39]. The automotive industry addresses this problem through a paradigm shift from a hardware-, component-driven to a requirement- and function-driven development process, and a stringent standardization of infrastructure elements.

The principal aim of the AUTomotive Open System ARchitecture (AUTOSAR) standard is to master the growing complexity of automotive electronic architectures [40]. We refer to the *AUTOSAR ecosystem* as a subset of the *automotive ecosystem*, where different actors participate in value creation (i.e. development of software components) by exchanging products and services based on a technical platform defined by the AUTOSAR standard.

Several challenging areas, including requirements engineering, are reported in the automotive domain [41]. OEMs and suppliers need to communicate requirements based on the requirements documents, which are imprecise and incomplete nowadays [41]. In this regard, the work in [42] reports that the requirements value chain is little understood beyond software projects. It is unclear which requirements communication, collaboration, and decision-making principles lead to efficient, value-creating and sustainable alignment of interests between interdependent stakeholders across software projects and products [42]. This is an important point since the way the interests and expectations of stakeholders of SECOs are communicated is critical for successfully influencing stakeholders in conceiving future solutions that meet their needs [43]. It is important to highlight that in the automotive domain, communication, collaboration, and decision-making are cross-organizational challenges related to the requirements viewpoint and requirements engineering.

5. Volvo Cars architecture framework

The starting point for defining an architecture framework is the identification of established stakeholders within the domain of the framework. Stakeholders may be individuals, teams, organizations or classes (of individuals, teams or organizations), while concerns may be fine-grained or very broad in scope [11].

5.1. Stakeholders

Table 1 describes the main stakeholders we have identified through dedicated meetings with software architects of Volvo Cars; they fall into five major groups:

- *End-users* of the electrical system, like drivers and passengers.
- *Customer* stakeholders, such as paying customers of products and services that depend on the electrical system (i.e. the car)

Table 1
Overview of stakeholders.

Stakeholder	Group	Comment
Passengers	end-user	
Drivers	end-user	
Customers	customer	Purchaser of a car or related service
Product planner	customer	Acquirer of electrical system
Purchaser	customer	Purchasers of electrical system
Line managers	management	Has scheduling responsibility, long term quality responsibility, includes group, department
Project managers	management	Owns budget for development
System architects	developers of electrical system	
Functional developers	developers of electrical system	Owns functional and non-functional aspects (Synonyms: function owner; function realizer; function developer, function realizer, system developer)
Component developers	developers of electrical system	
SW supplier (internal/external)	developers of electrical system	Can be internal or external from the perspective of the OEM.
HW supplier (internal/external)	developers of electrical system	Can be internal or external from the perspective of the OEM.
Testers	developers of electrical system	
Attribute Owners	developers of electrical system	Owns non-functional attributes like performance
Tool Engineers	developers of electrical system	Specifically testing tools, including design tools (e.g. for requirements)
Calibrators	developers of electrical system	
Diagnostic method engineers	maintainers of electrical system	
Workshop Personnel	maintainers of electrical system	
Fault Tracing Specialists	maintainers of electrical system	
Technical Specialist	specialists	Support developers and maintainers on specific topics

and product planners, who acquire the electrical system as part of the overall product.
- *Management* with responsibility for scheduling, long term quality, groups, departments, and budget.
- *Developers of the electrical system* include engineers throughout the value chain that create the electrical system, its architecture, and the necessary tools that test and integrate the various components.
- *Maintainers of the electrical system* who interact with the electrical system throughout its lifetime.

We then exploit the identified stakeholders to identify the set of concerns on which the architecture framework will focus. This will help the consumer of the architecture framework and of the views and connected modeling tools to understand why they are modeling and when they are done.

5.2. Challenging scenarios

In order to define the stakeholders concerns, we identified a set of challenging scenarios through a number of focused groups with experts in automotive engineering and architecture from different companies and universities. These scenarios have been then refined for defining the architecture framework and its viewpoints. Fig. 2 gives an overview of the scenarios that are strongly connected to the viewpoints we will detail in this paper. These scenarios are described in the following items.

- Scenario 1 **Decision Management** aims at exploring how to make, communicate, and manage decisions.

  User Story: *"As a member of the development ecosystem I would like to have a clear understanding of how to take decisions and how to communicate them."*

  Interesting sub-scenarios include decisions about:
  - *Requirements (1.1)*: When decisions on requirements are made too early, it will lead to unnecessary changes.

    User Story: *"As a component responsible, I need to write a requirement. Currently, I am forced to write the requirement in a certain document or certain structure. This determines whether the requirement refers to a HW or SW component, whether it will be implemented in-house or by an external supplier. Often, it is too early to make such decisions and changes are necessary later, when more information becomes available. I do not feel comfortable to make this decision so early."*

  - *Architecture (1.2)*: Architectural decision making involves making the right decision, communicating it, ensuring that it is followed, and changing it when needed.

    User Story: *"As an architect* (one of {system architect; functional developer; low level architect}), *I need to make the right decision at the right moment (i.e. when it is useful to make the decision and when the necessary information is available). I need to make this decision on the right level. I need to be introvert to make the best possible decision on the available data and extrovert to communicate it."*

  - *Customer functions (1.3)*: Electrical architecture is guiding realization of customer functions, but it is not obvious how the architecture supports customer functions (with respect to tracing and methodology).

    User Story: *"As a product manager, I want to be supported by the electrical architecture in making decisions about customer functions. Based on a wishlist from the Market Analysis Department, I engage in a dialog with the departments that design the system. For this task, I wish for support froman electrical architecture that not only takes into account non-functional aspects, but also the nature of customer functions (which changes over time)."*

  - *Change management (1.4)*: Good flexibility of the architecture allows us to continuously remove assumptions and do changes even late in the process. However, in a weak electrical architecture, such changes will impact the stability of the electrical system because of technical dependencies.

    User Story: *"As a product planner I want to have a flexible Change Management process, allowing me to change or*

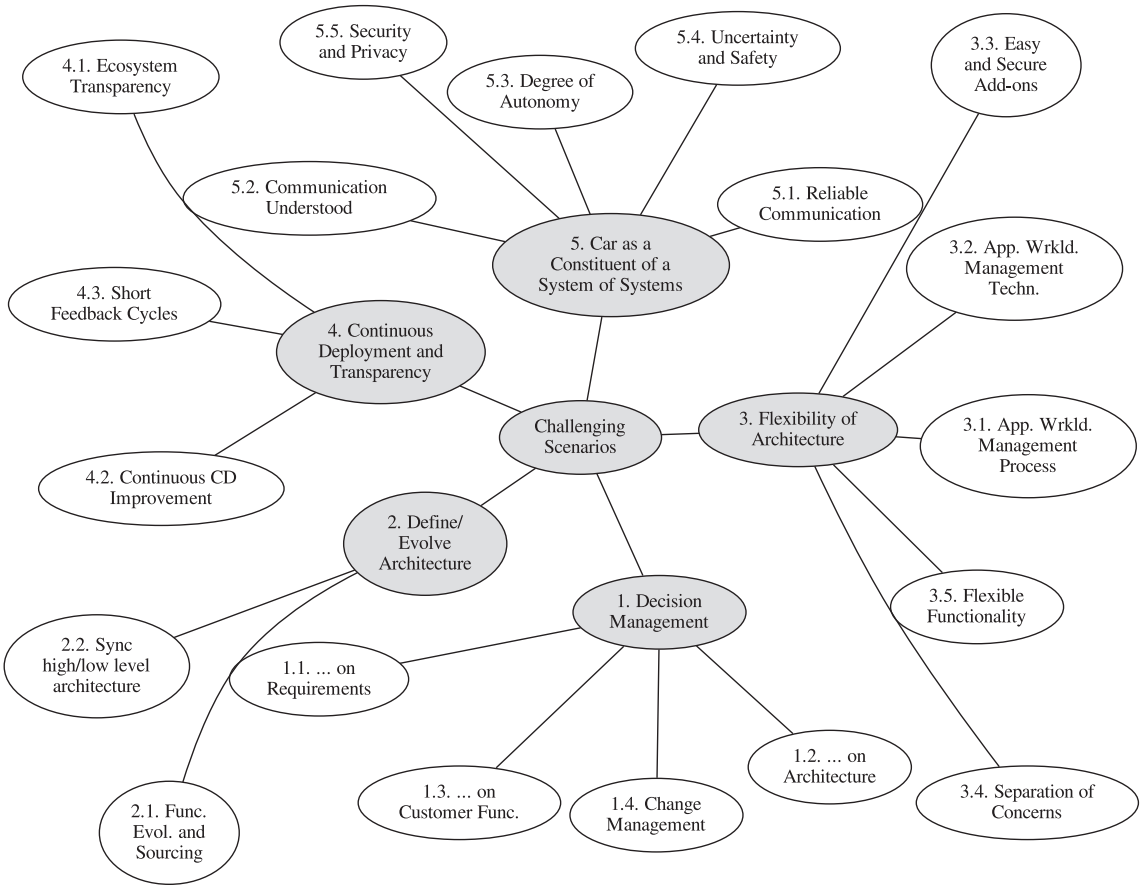

Fig. 2. Overview of challenging scenarios.

add functions late in the process, often with the goal of removing assumptions. This includes for example defining and changing the allocation of Functions to ECU late in the process."

- Scenario 2 **Define/Evolve Architecture** explores aspects of long-time evolution of the electrical architecture. This includes:
  - *The impact of long-term sourcing decisions on the logical architecture (2.1)*: Long term sourcing contracts allow optimizing the production cost, but constrain evolution of the architecture.

    User Story: *"As a Function Developer I want to optimize the evolution of functions without considering sourcing agreements. Today I have to discuss with the product planner about the plan for a function two years or more in advance to reflect on existing contracts, e.g. when related nodes are sourced for different time intervals. Problem: Sourcing needs make physical layout dominate logical decisions."*

  - *The danger of architecture and design evolving in different directions (2.2)*: If not actively managed, architecture and design diverge over time. The architecture is then perceived as outdated and not useful, thus it looses its ability to guide design decisions and implementation.

    User Story: *"As a system architect or function developer I want a stringent correlation between architecture and design. Otherwise, one or the other is wrong."*

- Scenario 3 **Flexibility of Architecture** addresses different scenarios that emphasize flexibility of the electrical architecture, both on a technical and on a process level. On a technical level, more capacity in the ECUs allows the late addition of functionalities. On a process level, the available resources need to be managed across several contributing partners. This includes:
  - *Application workload management from a process perspective (3.1)*:

    User Story: *"As a functional developer, I want to be flexible when it comes to application workload. Suppliers should be able to acquire and return resources dynamically throughout the lifecycle. This would allow moving faster from assumptions to decisions, and allow decisions to be changed as needed."*

  - *Application workload management from a technical perspective (3.2)*: Higher capacity of ECUs and Buses will facilitate late or even very late updates (i.e. when the vehicle is on the street), because new functionality could use more resources. This flexibility comes at a cost and it is not trivial to understand where the break-even point is.

    User Story: *"As a system architect I want to balance capacity of ECUs and Buses against cost, to ensure future-proofing is done in an economically feasible way."*

  - *Easy and secure add-ons (3.3)*: Being able to add new functions in a secure and easy way would allow us to detach software development to some extent from the development cycle. During the development of a car, an OEM could focus on the critical basic functionality. Convenience features as well as more advanced connected features could be added independent from the start of production. This would allow us to develop more continuously and should

decrease the peak of trouble reports before critical deadlines observed earlier [44].

User Story: *"As a functional developer I want to be able to add new functions (very) late. Such add-ons could originate from third parties and should be added even after the vehicle has been put into use. In that way I could avoid big-bang integration just before the start of production."*

– *Separation of concerns (3.4)*:

User Story: *"As a system architect, I want to balance separation of concerns on two levels: between domains (e.g. safety vs. infotainment) and levels of abstractions (e.g. architecture vision vs. architecture implementation). By this, I empower developers to make decisions within the constraints of the architecture and to apply appropriate development methods within their context (e.g. safety vs. non-safety)."*

– *Flexible functionality (3.5)*:

User Story: *"As a functional developer, I want to be flexible about the functions that are running in the car and allow for very late deployment. Thus, I gain flexibility and short time-to-market for implementing specific customer features."*

• Scenario 4 **Continuous Deployment and Transparency** includes:

– *The need for openness and transparent information throughout the different value chains in the automotive ecosystem (4.1)*: Good transparency in the value chain supports flexibility, since all partners can take appropriate action to reach a (changing) goal quickly. The *cone of uncertainty* is a good example for this [45,46]: In the beginning of a project, not much data is available and decisions are very uncertain. This is also true in the automotive domain where the architecture work starts with an already existing system - usually the architecture of the previous series - that the developers try to enhance without changing too much in order to reduce risks. As the project proceeds, assumptions are removed and uncertainty is reduced, but as a consequence, it becomes harder to make decisions that change a lot. Good transparency can help to reduce the uncertainty quicker and can allow us to decide and learn fast.

User Story: *"As any developer of the electrical system (internal or external), I want to have access to the relevant decisions, to the status of relevant assumptions, and knowledge generated during the development. This allows me to participate in the fast learning throughout the value chain and enables me to be effective and flexible in my work."*

– *The need for using this information to continuously improve a continuous integration, delivery, and deployment flow (4.2)*: Cross-organizational continuous integration, delivery, and deployment facilitate fast feedback and rich learning. This learning should also help to improve the interaction of organizations and the integration flow between them. It is also necessary to create and maintain a culture of continuous improvement between the OEM and partners, e.g. by applying continuous deployment to the (shared) CD tool chain.

User Story: *"As a Project Manager, I want to have a culture of continuous improvement within the OEM and with partners in the value-chain. As any developer of the electrical system (internal and external) I want to benefit from and take responsibility in improving the continuous delivery flow."*

– *The need for establishing short feedback cycles (4.3)*: While developing new functionality, basic software, and hardware, one should plan on how to receive feedback, which data to collect, and how to use it in the development.

User Story: *"As any developer of the electrical system (internal or external) I want to have quick feedback on how my contribution will work on the various levels of integration. As a Functional Developer, I want to have fast and defined feedback cycles. As a tester, I want quick updates on all levels of tests and continuous improvement of functionality. This will allow to gain benefits from continuous integration (e.g. improved quality and faster overall development) at scale in automotive development."*

• Scenario 5 **Car as a constituent of a System of Systems** includes:

– *The need of having reliable communication (5.1)*: To have the car as a constituent of a System of Systems (SoS), the car needs to be connected to other cars, infrastructure, cloud, and any other kind of constituent system of the SoS. Most probably the car will be connected through heterogeneous communication means and with various degrees of quality of service.

User Story: *"As an architect I need to engineer the car so that it will support heterogeneous communication means with various quality of service (QoS).The QoS of the different communication means should be clearly identified so that it will be possible to develop end-to-end functionalities that go beyond the boundaries of the car."*

– *Guaranteeing that communication from and to the car is properly understood (5.2)*: It is not enough to have proper communication means; the exchanged information should be properly understood by all involved parts.

User Story: *"As an architect I need to ensure a high degree of interoperability between the car and other constituent systems of the SoS. End-to-end functionalities that go beyond the boundaries of the car might need to be built under the assumption that exchanged information is properly understood and required actions are taken by the receiving system."*

– *Degree of autonomy and readiness to be at the service of the SoS (5.3)*: Often SoS scenarios require a degree of autonomy of the vehicle. Moreover, often a constituent system has to temporarily sacrifice its own goal in order to take actions needed to fulfil the goal of the SoS.

User Story: *"As an architect I need to manage the transition towards autonomous behaviours of the car that are needed to achieve the goal of the SoS. The car should be engineered so to, often immediately, temporarily stop what is currently doing and perform the actions required by the SoS. The right tradeoff between independence of the vehicle (from the SoS) and service offered to the SoS needs to be found."*

– *Dealing with uncertainty and functional safety (5.4)*: Vehicles are starting receiving a huge amount of information from the environment as communication coming from other vehicles, pedestrians, road signals, and the city in general. This information might be precious for supporting new safety behaviours [47]. However, this information is often unreliable and subject to different dimensions of uncertainty, like presence of other constituent systems, communication means, etc.

User Story: *"As an architect I would like to support innovative and very promising safety scenarios that can involve cyclists, pedestrians, etc. However, this would imply that the way functional safety is ensured should change since the SoS is open (constituent systems might join or leave the SoS at any moment) and we cannot rely 100% on information sent by uncontrollable and independent constituent systems."*

– *Cyber security and privacy (5.5)*: Once the car is connected it is exposed to attacks as any other computer or device that is connected to Internet. The effects might be tragic.

User Story: *"As an architect I need to protect the car from attacks to avoid dangerous or catastrophic scenarios and to guarantee the privacy of user data."*

5.3. Consequences for architectural framework

The identified architecture-related concerns determine the choice of viewpoints and view to be included. It is important to note that the concepts involved in almost each viewpoint are already handled within the current architecture of Volvo Cars. However, we are investigating the definition of proper viewpoints that will create the architecture framework according to the ISO 42010 international standard [3]. Most of the viewpoints summarized in Section 2 are indeed interesting also for Volvo Cars. In addition to these viewpoints we foresee a set of viewpoints that are emerging from the new challenges of the automotive domain, as explained in Section 4 and illustrated by the scenarios described above. The new viewpoints are listed in the following; then for space constraints in the reminder of the paper we will focus on three of the most challenging ones:

- *Continuous integration and deployment* (detailed in Section 6) - OEMs are increasingly interested to reduce the development time, to increase flexibility, to have early feedback on decisions made, and to add new functionalities incrementally even after production.
- *Ecosystem and transparency* (detailed in Section 7) - somehow related to the value net viewpoint only briefly sketched in AAF [4,21]. The ecosystem around the OEM can be seen as a virtual organization consisting of the OEM, its suppliers and other partners involved in the process of creating customer value.
- *System of Systems viewpoint: vehicle point of view* (detailed in Section 8) - Future scenarios of collaborating and connected vehicles are posing new challenges on the vehicle architecture. This viewpoint aims at understanding how the architecture should change in order to engineer a car that is a constituent of a System of Systems.
- *Autonomous cars* - autonomous cars require special architecture solutions, e.g. inspired by autonomous and self-adaptive systems [48].
- *Modes management* - a mode viewpoint is needed to design the different modes of a vehicle as well as the transitions from one mode to another.
- Special viewpoints and views might be conceived to enable dissemination and communication of the architecture to developers and other stakeholders, as discussed in Section 4.2.

The natural consequence of the use of multiple viewpoints and views in architecture descriptions is the need to express correspondences and consistency rules between those views. The mechanism introduced in [3] is called model correspondences and it allows the definition of relations between two or more architecture models. Since architecture views are composed of architecture models [3], model correspondences can be used to relate views to express consistency, traceability, refinement or other dependencies [11]. These mechanisms allow an architect to impose constraints between types of models and check whether these constraints are satisfied by the architecture.

6. Continuous integration and deployment viewpoint

6.1. Continuous integration and deployment

The viewpoint "Continuous Integration and Deployment" adds a development perspective to the system architecture and aims to give an answer to the following questions:

1. How do continuous integration and deployment practices in automotive engineering impact the system architecture?
2. How do architectural decisions in the system architecture impact continuous integration and deployment practices?

6.2. Overview

Agile approaches and practices such as continuous integration and deployment promise to help reducing development time, to increase flexibility, and to generally shorten the feedback cycle time, which in turn can lead to a better management of complex system knowledge. However, the complex supplier network (see the Ecosystem and Transparency viewpoint in Section 7), and typical setup with a large number of ECUs, pose specific challenges to these practices, including safety concerns and complex dependencies.

Support for continuous deployment has to face safety concerns. An example for this is the question on whether it should be possible, at runtime, to modify or update the software running on an ECU responsible for a safety critical function. This might be desirable, e.g. when better algorithms for autonomous driving become available.

Dependencies between ECUs are a property of the architecture. As mentioned in Section 4.1, the as-realized architecture may differ from the as-intended architecture, and continuous integration and deployment of software may entail architectural changes. This highlights both the need for collaboration between parts of the organization working on different architectural levels, and the need of a proper support for agile and flexible architecting, which raises two main concerns: (i) one concerning architecture as an enabler of continuous integration and deployment, facilitating variant handling and coordination of updates, and (ii) another concerning continuous integration and deployment on the architecture level, facilitating reasoning about modifications to the architecture itself.

6.3. Concerns and stakeholders

6.3.1. Concerns

In this section we focus on the concerns that are essential to enable continuous integration and deployment when engineering software for cars. We express the concerns in the form of questions as suggested by the ISO/IEC/IEEE 42010 standard.

- *How can we avoid building the wrong architecture?* Even a technically sound architecture is worthless if it does not fit the desired purpose. Naturally, a system architecture needs to exist long before the product under consideration is developed and can reach the market, since it is required to guide organization of development activities. Even though a lot of information is needed early in the product lifecycle, a crucial amount of information will only become available during its development and early market phase.
- *How can we reduce the number of architectural assumptions?* The more the architecture is defined up-front, the more it is based

on assumptions instead of facts. This can lead to late changes, duplicate work, or in the worst case to problems that only surface once the product is on the market.
- *How can a system respond quicker to changes in the market?* Many of the assumptions will concern the market situation once the product has been released. Markets are however prone to constant change and such change might invalidate architectural decisions during the development of software.
- *How can we deal with changing interfaces?* Reacting to change in the market will lead to late changes of the architecture, which in turn will affect continuous integration and deployment: it is not possible to integrate a component into the system after its interface changes, unless the platform and other dependent components have already been adjusted to the new interface. Deployment after an interface change might therefore involve a large number of updated components, which will not be possible in a continuous fashion. In other words, after an interface change it will be hard to guarantee that the main branch of all components is in a deployable state.
- *How can we deal with dependencies?* Dependencies might require additional coordination effort between teams, which effectively slows down or even hinders continuous integration and deployment. There are two types of dependencies, explicit and implicit dependencies. Explicit dependencies are covered by the as-intended architecture and can be properly managed, monitored, and updated during the continuous system evolution. Implicit dependencies should be properly identified and reduced as much as possible as follows. Those implicit dependencies that are undesirable (e.g., those that violate important architecture constraints) should be removed. The remaining ones should be made explicit.

6.3.2. Typical stakeholders

For the continuous integration and deployment viewpoint, we need to consider all stakeholders described in Section 5 and summarized in Table 1. In addition, we need to consider suppliers, as continuous integration and deployment will depend on their deliveries (see also the ecosystem and transparency viewpoint in the next section).

6.4. Model kinds for continuous integration and deployment viewpoint

Various notations have been proposed to model continuous integration and deployment pipelines [49]. Among those, several could be applicable to this viewpoint. In addition, one would need to relate those to architectural approaches as well as to the concerns defined above. We are currently working on a qualitative assessment method that will allow us to identify disruptors and bottlenecks for continuous integration and deployment. A CI view would then elaborate on how disruption or unnecessary slowdowns is avoided. Moreover, it would be desirable a modeling notation that is able to capture and visualize dependencies. Further investigation is needed to understand the most suitable model kinds for this viewpoint.

Grubb and Chechik have modeled system evolution with extended goal models [50]. Their approach allows to reason to what extend stakeholder goals are fulfilled during the evolution of a system. We plan to investigate whether this technique can provide a suitable model kind, that for example allows reasoning at what point of time continuous integration becomes possible and which implications the architecture has on continuous architecture over the development time. By this we expect that the dependencies between system architecture, processes, and organizations become more explicit and can be better optimized with respect to supporting business goals.

6.5. Operations on views

Operations on views should address two main concerns of this viewpoint: (i) enabling continuous integration and deployment through architecture, and (ii) supporting continuous integration and deployment on the architecture level. As examples for the first group of operation we mention methods to identify and remove undesired implicit dependencies. Moreover, suitable operations are needed to maintain, monitor, and update explicit dependencies according to the system evolution.

As examples for the second group we mention operations to identify and show the exact evolutions of the system that have an impact on the system architecture. Specifically these operations should support developers by highlighting the effects of changes on the overall architecture, as well as potential violations of architecture constraints before performing a deployment. This can be particularly complex when the changes impact on non-functional properties, such as performance [51].

Finally, operations should be also provided to evaluate architecture descriptions and decisions. For instance, analysis methods might be defined to evaluate the efficiency of the feedback cycle in continuous integration and deployment as well as its ability to help resolving assumptions.

6.6. Correspondence rules

This viewpoint has correspondences with various other viewpoints, including functional safety, security, privacy, and the new viewpoint Ecosystem and Transparency proposed in this paper. The latter needs to be considered when continuous integration and deployment involves contributions from other actors in the ecosystem.

7. Ecosystem and transparency viewpoint

7.1. Ecosystem and transparency

The viewpoint “Ecosystem and Transparency” focuses on the value-chain of hardware and software, as well as logical components of the architecture and aims at answering the following questions:

1. How can an electrical architecture enable a network of organizations (including OEM, Tier-1 supplier, Tier-2 supplier) to work together to (continuously) provide value?
2. How are architectural decisions affected by the need for transparent and continuous collaboration within the value-chain?

7.2. Overview

The Ecosystem and Transparency viewpoint takes into account architectural assets in automotive engineering that are provided based on a complex network of cooperating organizations. A logical component in the architecture will be implemented by a variety of physical components, including hardware and software on different levels of abstraction.

In a concrete scenario, a logical component could be deployed on several AUTOSAR ECUs. The hardware in this scenario would be provided by different Tier-1 suppliers, while the AUTOSAR layer would be provided by one or more certified AUTOSAR suppliers. These AUTOSAR suppliers could be characterised as Tier-2 suppliers, since they are contracted by the Tier-1 suppliers who are supposed to deliver an ECU with basic software, but different setups are possible. On top of these ECUs, application software would be developed either by the OEM or by separate software suppliers in order to provide the services defined by a logical component.

The example scenario above shows the importance of the Ecosystem and Transparency viewpoint: if an electrical architecture should take into account strategic business goals such as improved flexibility, reduced time-to-market, or development efficiency, the Ecosystem and Transparency viewpoint will offer important information that must be considered. This viewpoint ties into business decisions and often long-running sourcing contracts, but also into new business models with respect to after market software updates. Thus, we expect that lifecycles of hardware and software components are important architectural aspects as well as managing different development cycles of suppliers and components.

7.3. Concerns and stakeholders

7.3.1. Concerns

In this section we focus on the concerns that are essential to enable efficient work in the ecosystem of automotive electrical systems.

- *Which types of value-chains are implied by a given system architecture and what is their purpose?* A given system architecture will define the interplay of different types of logical components with the goal to support user visible features. While some of these features will be well-understood and stable, others will be highly innovative and their development will be subject to volatile requirements. Value-chains for implementing those logical components will differ each other based on the type of feature that they support.
- *How to map supplier development capabilities to demands created by a specific system architecture?* Different capabilities are demanded for development across the value-chain, depending of the type of feature. A system architecture might be able to minimize or limit interaction on critical innovative features in the ecosystem to only actors that can support a highly iterative development.
- *How can we establish the required level of transparency in a value-chain?* The more volatile the requirements of a feature are, the more information exchange is needed between ecosystem actors. To enable continuous delivery throughout the value-chain, all involved partners should have access to critical information so that they can take responsibility. In a less volatile environment the (legal and organizational) effort of creating high-transparency will not pay off.
- *How can we manage transparency (e.g. of architectural decisions) in the face of changing suppliers?* In order to maintain a powerful automotive engineering ecosystem we need to be able to change actor relationships. This will also affect the level of transparency (including what information to share and at what frequency).

7.3.2. Typical stakeholders

For the ecosystem and transparency viewpoint, we need to consider stakeholders within the different ecosystem actors. With respect to the OEM (the keystone actor in the automotive engineering ecosystem), potentially every stakeholder described in Section 5 and summarized in Table 1 will be part of the audience of this viewpoint.

Other actors (such as Tier-1 and Tier-2 suppliers) will have very similar stakeholders that need to be considered.

7.4. Model kinds for ecosystem and transparency

Since the we are proposing a new viewpoint for Ecosystem and Transparency concerns in this paper, there is no established way of modelling it so far. Models would need to provide the following information:

Analysis of Value-Chain: In order to document the different value-chains, one could either create a map of a (part of an) ecosystem as proposed by Janssen et al. [52] or single out a specific value chain as we for example did in our previous work on the AUTOSAR ecosystem [33] based on Boucharas et al.'s notation [53].

Analysis of Actors and Goals: In addition to the critical value chains, one should maintain information about the actors involved in those chains and their goals. Yu and Franch have proposed to use goal models for this purpose, which is a promising approach to align goals of different actors based on architectural decisions and to achieve win-win situations [54]. These are important when targeting strong collaborations and partnerships.

Reasoning about anonymous actors: To some extent, the concrete partners and suppliers are not known when creating the electrical architecture. In such situations, we propose a persona based approach, i.e. defining archetypical actors with important characteristics as well as their expectation towards the ecosystem [55,56].

Defining Transparency Requirements: Hosseini et al. have proposed a framework for defining transparency requirements which could be a good starting point for these aspects [57,58]. For practical use in this viewpoint, this framework would need to be extended with dynamic aspects, e.g. for withdrawing information when a partnership ends. This includes also ownership and digital rights management issues, for which we hope to borrow concepts from other works (e.g. from [59]).

7.5. Operations on views

Please refer to the previous section, i.e. Section 7.4, for a list of operations on views.

Moreover, operations should be also provided to evaluate architecture descriptions and decisions. For instance, analysis methods could be defined to (i) check win-win situations in the ecosystem, (ii) identify suitable partners for critical value-chains, and (iii) analyse and manage required transparency levels with development partners.

7.6. Correspondence rules

This viewpoint has correspondences with various other viewpoints, including security, privacy, and value-net, but also to Continuous Integration and Deployment in the sense that development cycles of different ecosystem actors need to be synchronized in order to guarantee for quick feedback cycles. In addition, it corresponds with the System of Systems viewpoint, e.g. if several constituents of the SoS need to be aligned to provide a certain functionality.

8. System of Systems viewpoint: vehicle point of view

8.1. System of Systems: vehicle point of view

The viewpoint "System of Systems: vehicle point of view" focuses on the car as a constituent of the SoS. Therefore, this viewpoint aims at giving an answer to this question: How to engineer a car so to be part of a System of Systems?

8.2. Overview

Connected cars will benefit from Intelligent Transport Systems (ITS), Smart Cities and IoT to provide new application scenarios like

smart traffic control, smart platooning coordination, collective collision avoidance, etc. Vehicles will combine data collected through its sensors with external data coming from the environment, e.g. other vehicles, road, cloud, etc.

Connected vehicles will face new challenges and opportunities related to safety issues. Current regulations for safety aspects, like the ISO 26262 standard, do not account for scenarios in which the vehicle is part of a more complex system; the challenge is on how to manage new hazards that, coming from the environment, could jeopardize safety. At the same time, connected cars open new opportunities for safety, called "connected safety" within Volvo Cars:[6] e.g., this new technology will allow a connected car to be aware of a slippery road, of cyclists on the road, etc., so to initiate all the actions needed to avoid accidents and collisions.

These scenarios are posing new requirements to the architecture. For instance, one of the main issue with safety guarantees in connected cars is that a full analysis of the system is not possible at design time. When moving from a single vehicle to a cooperative system, a new safety analysis is required to handle uncertainties at runtime. Some approaches have been proposed to deal with certification at runtime, e.g. [60,61], but a clear framework that can be used to define the connected safety requirements is still missing.

8.3. Concerns and stakeholders

8.3.1. Concerns

In this section we focus on the concerns that are essential to enable functions in the systems of systems for cars.

- Once the car is part of a SoS, how to guarantee functional safety requirements?
- Once the car is part of a SoS, what are the implication on system design and functional distribution for functional safety?
- Once functional safety requirements involve devices that are outside of the vehicle (other constituent systems of the SoS), how to ensure that these requirements will be guaranteed?
- How the methods and processes for end-to-end function development and continuous delivery of software need to evolve to be suitable in a systems of systems setting?
- How to enable a reliable and efficient communication between the vehicle and heterogeneous entities, like other vehicles, road signals, pedestrians, etc.?
- How to be sure that the vehicle and other constituent systems of the SoS will be able to exchange information and to use the information that has been ex-changed?
- How to guarantee that the security of the vehicle is preserved once the vehicle becomes connected?
- How to identify the right tradeoff between shared data and users' privacy?
- How to keep the data shared within the SoS (and possible replication of data) sufficiently updated or synchronized?
- How to manage the age of available information?
- Which functions in the car are allowed to make use of data coming from other constituents?

8.3.2. Typical stakeholders

Since in this viewpoint we take the vehicle point of view, potentially every stakeholder described in Section 5 and summarized in Table 1 will be part of the audience of this viewpoint. Moreover, additional stakeholders external to the company should be considered as stakeholders. Examples of them are standard authorities for safety and communication means used by the SoS, stakeholders of other OEMs, suppliers, road authorities, etc.

8.4. Model kinds for car as constituent of a SoS

There are not obvious model kinds that can be used for this viewpoint. There are some attempts in the literature to define Architectural Languages (ALs) for SoS, such as [62] or the effort made within the Atesst project,[7] however, a deep analysis of these languages need to be performed in order to precisely understand whether the ADL can exactly satisfy the needs of this viewpoint and of Volvo cars. Results of an initial analysis suggest that this language might be too formal and heavy. As it is stated in [23,28] often ALs are not adopted by practitioners since they are (i) too complex and require specialized competencies, (ii) the return on investment is insufficiently perceived, (iii) they require over-specification and at the same time practitioners are not able to model design decisions explicitly in the AL, and (iv) there is a lack of integration in the software and system life cycle, lack of mature tools, and usability issues.

This suggests that the language to be used should be defined and/or customized within the company, so that it can precisely implement the needs of the company and users and at the same time can be integrated with the company life-cycle. An interesting approach to customize existing ALs according to specific needs might be found in [63].

The model kind for this viewpoint should enable the specification of the following concepts that are essential building blocks for engineering a car that will be part of a System of Systems. These building blocks have been identified within the NGEA project and are documented in the deliverable D3.2 Systems of Systems for Cars Concepts.

- **Heterogeneity of communication channels**: Today cars already use several communication channels for communicating with other systems. Communication means include also GPS receivers, sound signals, visual break lights, and turn indicators, which can be used to communicate to other systems the intention of the driver, as well as cameras able to detect other vehicles intentions or to read road signs, which are then highlighted to the driver. In the near future we can expect several new communication channels. Examples are IMT2020 (5G) the successor of the LTE (4G) mobile networks, Vehicle to vehicle (V2V) and Vehicle to infrastructure (V2I) communication over IEEE 802.11p, BLE (Bluetooth low energy), Proprietary communication channels used by single or a few vendors, standard WiFi versions, etc.
- **Connectivity**: Connectivity is essential to enable the expected service level in different places of the SoS. Connectivity may be possible through many different channels as discusses in the previous item. However there has to be on-board functions that handle graceful degradation of services when connectivity is limited or not available at all. The user shall experience a robust behaviour of the available functions. The quality of service (QoS) of both data and functions must be made sufficiently clear to the users. Acceptable QoS typically requires very good connectivity and availability of real-time information. However, since connectivity cannot always be guaranteed, other forms of fall-back solutions or redundancy are usually needed. The QoS available in a specific location and time need to be communicated to the user in a user-friendly and easy to grasp way.
- **Large and unreliable information**: low quality information could lead to disastrous events, especially when functions are implemented to rely heavily on external information. Therefore, it is important to be able to determine whether the received information's quality is high in term of correctness and timeli-

[6] https://goo.gl/mIWWS3.

[7] http://www.atesst.org/home/liblocal/docs/ows/I5_ATESST2_OWS_CooperativeSystems.pdf.

ness. Multi-sensor data fusion provides an answer by combining perceived data from sensors to make the resulting information more reliable.

- **Interoperability**: Interoperability is the ability of diverse systems to work together. This general definition has been conjugated in many different ways based on the reference application area and on the many different factors and aspects characterizing them. Interoperability involves standards, protocols, and integration and adaptation of interfaces to enable the effective and efficient communication between constituent systems. Interoperability is the ability of two or more constituent systems that are part of SoS to exchange information and to use the information that has been ex-changed. Unambiguous interpretation of shared data between systems is necessary for interoperation, but it is not sufficient. Despite standards for shared data provides specification with the objective to enhance the functionality and interoperability, the data encoded using these standards are not necessarily interoperable. For instance, concepts that have the same labels, and somehow even the same meaning, can be used completely differently in different applications. This is for instance the case of speed within a car that can have different meanings.
- **Cyber security and privacy**: Connectivity of cars poses challenges both on security and privacy since cars become exposed as any other device that is connected to Internet. A representative example related to security risks is the hacking of a Jeep car in 2015 [64]. For what concerns privacy, cars will need to share several information to enable promising SoS scenarios, however sensitive information should be properly protected.
- **Distributed end-to-end functionality**: End-to-end functionalities should be provided not only between nodes in the vehicles but also outside the vehicle, and then involving other constituent systems of the SoS, e.g., clouds, other cars, pedestrians, road infrastructure and signals, etc. Connected vehicles can benefit a lot from access to cloud computing based data and services. Services implemented in distributed end-to-end functions will benefit from the possibility to dynamically load software to the on-board electrical architecture. Dynamically loaded software may be executed in one or several physical nodes, and this will open challenges in terms of cyber-security, functional safety and compatibility.
- **Functional safety**: Functional safety requirements are expected to apply for functions outside the car. To be able to handle safety once the car becomes a constituent system of a SoS, one can expect that operational and key functions will be protected. However, connected safety [47] promises important and interesting improvements for what concerns the overall safety of the car. A typical example is a car that receives information about some ice on the street from the cloud as reported from another trusted vehicle. These types of scenarios might implicitly create expectations to the driver of a car that will start relying on a service that is however dependent on potentially uncontrollable connections, devices, sensors, etc.

8.5. Operations on views

Model-driven engineering (MDE) techniques provide interesting capabilities to define new model kinds [28] or to tune and customize existing ones [63]. MDE techniques typically rely on the definition of a metamodel representing the concepts of the model kind. Then views can be seen as models that have to be compliant to the viewpoint's rules and constraints, i.e. the models (views) should conform to metamodels (viewpoints). The interested reader can find further information in [65,66].

Other operations should help and guide the user in interpreting the views. The starting point should be a study focused on understanding exactly both the users and the purpose of the views. This study would then define the requirements of the syntax (e.g., graphical, textual, tree-like) to be used for the language. Operations should be also provided to evaluate architecture descriptions and decisions. For instance, analysis methods might be defined to evaluate the interoperability level, the ability of the system to cope with heterogeneity of communication channels, security and privacy when the car become part of the SoS, etc. Finally, the view should be connected to the software and system life-cycle through methods helping the design and build of the system.

8.6. Correspondence rules

This viewpoint has correspondences with various other viewpoints, including functional safety, security, privacy, and the other SoS viewpoint that focuses on the engineering of the overall SoS. Moreover, this viewpoint has also correspondences with the topology, cost, variability, continuous integration and deployment and ecosystem and transparency.

9. Discussion and concluding remarks

In this paper we describe the current investigation of Volvo Cars towards the definition of an architecture framework. At this stage we are identifying potential viewpoints of the vehicles of the near future based on our shared experience in the automotive domain as well as a series of workshops for identification and validation for important concepts with a wide range of domain experts. Specifically, we present challenging scenarios about future demands related to architecture and proposed three new viewpoints (Continuous Integration and Deployment, Ecosystem and Transparency, System of System). For each of these viewpoints, we start collecting actual needs and consumers of their respective views. Based on these, future work needs to identify the most suitable modeling languages to be used to describe the architecture. For these modeling languages, it is important to find the right tradeoff between a language's (i) ability to support the level of formality and precision required by disciplined development processes, or (ii) simplicity and it being intuitive enough to communicate the right message to stakeholders and to promote collaboration [23,28]. We plan to evaluate the architecture framework by putting it in practice and through a qualitative evaluation with architects within Volvo Cars and by using ATAM (Architecture Tradeoff Analysis Method) [67] as a method for evaluating our architecture with respect to identified quality attribute goals.

In a continuous architecting context we expect that viewpoints, stakeholders, concerns, etc. will change over time. Then, there should be a continuous evaluation to understand whether changes on stakeholders and concerns should, in turn, trigger changes on the architecture. When the architecture change, the new version of the architecture should be properly documented; this would mean another architecture description since the rules of the 42010 standard [3] are written for expressing one architecture with one architecture description. It would be profitable also to link, or provide some traceability between old and new, as-intended and as-realised, and other kinds of transitions as an architecture evolves. The same thinking would apply to evolving viewpoints: as the stakeholders and concerns change, the viewpoints used may evolve and correspondence rules should be used to document what changes, what is constant, etc.

For what concerns the realization of the architecture framework we will investigate MEGAF [65,66], which is an infrastructure that enables software architects to realize their own architecture frameworks specialized for a particular application domain or community of stakeholders and compliant to the ISO/IEC/IEEE 42010 standard [3].

Acknowledgement

We would like to thank Rich Hilliard for his precious comments and support. We would like to thanks also the consortium of the NGEA project for the valuable input that allowed the definition of the architecture framework. This work is also partially supported by the Wallenberg Autonomous Systems Program (WASP).

References

[1] Peter van Staa, How KETs can contribute to the re-industrialisation of Europe, European Technology Congress, Wrocław, June 12–13, 2014:, 2014. http://docplayer.net/21724658-Date-2012-how-kets-can-contribute-to-the-re-industrialisation-of-europe.html.
[2] P. Pelliccione, E. Knauss, R. Heldal, M. Ågren, P. Mallozzi, A. Alminger, D. Borgentun, A proposal for an automotive architecture framework for Volvo Cars, in: 2016 Workshop on Automotive Systems/Software Architectures (WASA), 2016, pp. 18–21, doi:10.1109/WASA.2016.9.
[3] ISO/IEC/IEEE 42010, systems and software engineering — architecture description, ISO/IEC/IEEE, 2011.
[4] M. Broy, M. Gleirscher, S. Merenda, D. Wild, P. Kluge, W. Krenzer, Toward a holistic and standardized automotive architecture description, Computer 42 (12) (2009) 98–101. http://doi.ieeecomputersociety.org/10.1109/MC.2009.413.
[5] Y. Dajsuren, On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems, PhD thesis, Technische Universiteit Eindhoven, 2015.
[6] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, M. Goedicke, Viewpoints: a framework for integrating multiple perspectives in system development, Int. J. Softw. Eng. Knowl. Eng. 2 (1) (1992).
[7] P.B. Kruchten, The 4+1 view model of architecture, IEEE Softw. 12 (6) (1995).
[8] N. Rozanski, E. Woods, Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, Addison-Wesley Professional, 2005.
[9] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond, Addison-Wesley, 2003.
[10] P. Eeles, P. Cripps, The Process of Software Architecting, Addison Wesley, 2010.
[11] D. Emery, R. Hilliard, Every architecture description needs a framework: expressing architecture frameworks using ISO/IEC 42010, WICSA/ECSA 2009, 2009.
[12] ISO, ISO 15704 industrial automation systems ? Requirements for enterprise-reference architectures and methodologies, 2000,
[13] U.D.D.C.I. Officer, US department of defense architecture framework, version 2.02, August 2010:http://dodcio.defense.gov/Portals/0/Documents/DODAF/DoDAF_v2-02_web.pdf, 2010,
[14] O. Group, The open group architectural framework (TOGAF), version 9.1, December 2011:http://www.opengroup.org/subjectareas/enterprise/togaf, 2011.
[15] M. of Defence UK, MOD architecture framework (MODAF), version 9.1, December 2012:https://www.gov.uk/guidance/mod-architecture-framework, 2012.
[16] ISO/IEC-JTC1/SC7-WG42, Survey of architecture frameworks,
[17] D.A. Marca, C.L. McGowan, SADT: Structured Analysis and Design Technique, McGraw-Hill, Inc., New York, NY, USA, 1987.
[18] G. Booch, J. Rumbaugh, I. Jacobson, Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series), Addison-Wesley Professional, 2005.
[19] P. Cuenot, P. Frey, R. Johansson, H. Lönn, Y. Papadopoulos, M.-O. Reiser, A. Sandberg, D. Servat, R.T. Kolagari, M. Törngren, M. Weber, The east-adl architecture description language for automotive embedded software, in: Proceedings of MBEERTS'07, Springer-Verlag, 2010, pp. 297–307.
[20] S. Friedenthal, A. Moore, R. Steiner, A Practical Guide to SysML: Systems Modeling Language, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
[21] M. Broy, M. Gleirscher, P. Kluge, W. Krenzer, S. Merenda, D. Wild, Automotive Architecture Framework: Towards a Holistic and Standardised System Architecture Description, Technical Report, 2009.
[22] H.G.C. Góngora, T. Gaudré, S. Tucci-Piergiovanni, Towards an architectural design framework for automotive systems development, in: Complex Systems Design & Management, Springer Berlin Heidelberg, 2013, pp. 241–258, doi:10.1007/978-3-642-34404-6_16.
[23] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, A. Tang, What industry needs from architectural languages: a survey, IEEE Trans. Softw. Eng. 39 (6) (2013) 25. http://doi.ieeecomputersociety.org/10.1109/TSE.2012.74.
[24] N. Medvidovic, R.N. Taylor, A classification and comparison framework for software architecture description languages, IEEE Trans. Softw. Eng. 26 (1) (2000).
[25] M.V. M. Rappl P. Braun, C. Schröder, Automotive software development: a model based approach, World Congress of Automotive Engineers, SAE Technical Paper Series 2002-01-0875, 2002.
[26] U. Eliasson, R. Heldal, P. Pelliccione, J. Lantz, Architecting in the automotive domain: descriptive vs. prescriptive architecture, in: Proceedings of WICSA2015, 2015, pp. 115–118, doi:10.1109/WICSA.2015.18.
[27] R. Heldal, P. Pelliccione, U. Eliasson, J. Lantz, J. Derehag, J. Whittle, Descriptive vs. prescriptive models in industry, in: ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), 2016.
[28] P. Lago, I. Malavolta, H. Muccini, P. Pelliccione, A. Tang, The road ahead for architectural languages, Softw. IEEE 32 (1) (2015) 98–105, doi:10.1109/MS.2014.28.
[29] P. Kruchten, What do software architects really do? J. Syst. Softw. 81 (12) (2008) 2413–2416, doi:10.1016/j.jss.2008.08.025.
[30] A. Shahrokni, P. Gergely, J. Söderberg, P. Pelliccione, Organic evolution of development organizations-an experience report, 2016. SAE Technical Paper, 2016-01-0028, doi:10.4271/2016-01-0028.
[31] M. Waterman, J. Noble, G. Allan, How much up-front? A grounded theory of agile architecture, in: Proceedings of 37th IEEE International Conference on Software Engineering (ICSE '15), Florence, Italy, 2015, pp. 347–357.
[32] M. Soltani, E. Knauss, Challenges of requirements engineering in autosar ecosystems, in: Proceedings of 23rd IEEE International Requirements Engineering Conference, Ottawa, Canada, 2015a.
[33] M. Soltani, E. Knauss, Cross-organizational challenges of requirements engineering in the autosar ecosystem: a case study, in: Proceedings of 5th IEEE International Workshop on Empirical Requirements Engineering, Ottawa, Canada, 2015b. At RE'15
[34] H.H. Olsson, J. Bosch, Strategic ecosystem management: a multi-case study on challenges and strategies for different ecosystem types, in: Proceedings of 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA '15), Funchal, Portugal, 2015.
[35] S. Jansen, M. Cusumano, Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance, Edward Elgar Pub, 2013.
[36] K. Wnuk, K. Manikas, P. Runeson, M. Lantz, O. Weijden, H. Munir, Evaluating the governance model of hardware-dependent software ecosystems–a case study of the axis ecosystem, in: Software Business. Towards Continuous Value Delivery, Springer, 2014, pp. 212–226.
[37] G.K. Hanssen, A longitudinal case study of an emerging software ecosystem: implications for practice and theory, J. Syst. Softw. 85 (7) (2012) 1455–1466.
[38] E. Knauss, D. Damian, Towards enabling cross-organizational modeling in automotive ecosystems, in: Proceedings of MD2P2, 2014, pp. 38–47.
[39] H. Fennel, S. Bunzel, H. Heinecke, J. Bielefeld, S. Fürst, K.-P. Schnelle, W. Grote, N. Maldener, T. Weber, F. Wohlgemuth, et al., Achievements and exploitation of the autosar development partnership, Convergence 2006 (2006) 10.
[40] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, K. Lange, Autosar–a worldwide standard is on the road, 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden, 62, 2009.
[41] M. Broy, Challenges in automotive software engineering, in: Proceedings of the 28th International Conference on Software Engineering, ACM, 2006, pp. 33–42.
[42] S. Fricker, Requirements value chains: stakeholder management and requirements engineering in software ecosystems, in: Requirements Engineering: Foundation for Software Quality, Springer, 2010, pp. 60–66.
[43] S. Fricker, Specification and analysis of requirements negotiation strategy in software ecosystems., IWSECO@ ICSR, 2009.
[44] N. Mellegård, M. Staron, F. Törner, Why do we not learn from defects? - Towards defect-driven software process improvement, in: Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development, 2013, pp. 297–303, doi:10.5220/0004345602970303.
[45] S. McConnell, Rapid Development: Taming Wild Software Schedules, 1st, Microsoft Press, Redmond, WA, USA, 1996.
[46] T. Little, Schedule estimation and uncertainty surrounding the cone of uncertainty, IEEE Softw. 23 (3) (2006) 48–54, doi:10.1109/MS.2006.82.
[47] Volvo Cars, Volvo Cars' connected car program delivers pioneering vision of safety and convenience, https://www.media.volvocars.com/global/en-gb/media/pressreleases/159478/volvo-cars-connected-car-program-delivers-pioneering-vision-of-safety-and-convenience. 2015.
[48] M. Salehie, L. Tahvildari, Self-adaptive software: landscape and research challenges, ACM Trans. Auton. Adapt. Syst. 4 (2) (2009) 14:1–14:42, doi:10.1145/1516533.1516538.
[49] D. Ståhl, J. Bosch, Modeling continuous integration practice differences in industry software development, J. Syst. Softw. 87 (2014) 48–59.
[50] A.M. Grubb, M. Chechik, Looking into the crystal ball: requirements evolution over time, in: Proceedings of 24th IEEE International Requirements Engineering Conference (RE '16), Bejing, China, 2016, pp. 347–357.
[51] M. Fagerström, E.E. Ismail, G. Liebel, R. Guliani, F. Larsson, K. Nordling, E. Knauss, P. Pelliccione, Verdict machinery: on the need to automatically make sense of test results, in: Proceedings of the 25th International Symposium on Software Testing and Analysis, in: ISSTA 2016, ACM, New York, NY, USA, 2016, pp. 225–234, doi:10.1145/2931037.2931064.
[52] S. Jansen, S. Brinkkemper, A. Finkelstein, Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry, Edward Elgar, Cheltenham, UK, pp. 29–42.
[53] V. Boucharas, S. Jansen, S. Brinkkemper, Formalizing software ecosystem modeling, in: Proceedings of the 1st International Workshop on Open Component Ecosystems, ACM, 2009, pp. 41–50.
[54] X. Franch, A. Susi, E.S. Yu, Business and software ecosystems: how to model, analyze, and survive!, in: Proceedings of 23rd International Requirements Engineering Conference (RE '15), Ottawa, Canada, 2015. Tutorial based on RISCOSS EU FP7 project
[55] E. Knauss, I. Hammouda, EAM: ecosystemability assessment method, in: Proceedings of 22nd International Requirements Engineering Conference (RE '14), Karlskrona, Sweden, 2014, pp. 319–320.

[56] I. Hammouda, E. Knauss, L. Costantini, Continuous api-design for software ecosystems, in: Proceedings of 2nd International Workshop on Rapid and Continuous Software Engeering (RCoSE '15 @ ICSE), Florenz, Italy, 2015.
[57] M. Hosseini, A. Shahri, K. Phalp, R. Ali, A modelling language for transparency requirements in business information systems, in: International Conference on Advanced Information Systems Engineering, Springer International Publishing, 2016a, pp. 239–254.
[58] M. Hosseini, A. Shahri, K. Phalp, R. Ali, Foundations for transparency requirements engineering, in: Proceedings of 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '16), Gothenburg, Sweden, 2016b.
[59] A. Averbakh, E. Knauss, S. Kiesling, K. Schneider, Dedicated support for experience sharing in distributed software projects, in: Proceedings of 26th International Conference on Software Engineering and Knowledge Engineering (SEKE '14), Vancouver BC, Canada, 2014, pp. 355–360.
[60] D. Schneider, M. Trapp, Conditional safety certification of open adaptive systems, ACM Trans. Auton. Adapt. Syst. (TAAS) 8 (2) (2013) 8.
[61] K. Östberg, M. Bengtsson, Run time safety analysis for automotive systems in an open and adaptive environment, in: Proceedings of SAFECOMP 2013-Workshop ASCoMS, 2013.
[62] F. Oquendo, Formally describing the software architecture of systems-of-systems with sosadl, in: 2016 11th System of Systems Engineering Conference (SoSE), 2016, pp. 1–6, doi:10.1109/SYSOSE.2016.7542926.
[63] D. Di Ruscio, I. Malavolta, H. Muccini, P. Pelliccione, A. Pierantonio, Developing next generation ADLs through MDE techniques, ICSE 2010, 2010.
[64] Andy Greenberg, Hackers remotely kill a jeep on the highway - with me in it, https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. 2015,
[65] R. Hilliard, I. Malavolta, H. Muccini, P. Pelliccione, Realizing architecture frameworks through megamodelling techniques, in: Proceedings of ASE '10, ACM, 2010, pp. 305–308, doi:10.1145/1858996.1859057.
[66] R. Hilliard, I. Malavolta, H. Muccini, P. Pelliccione, On the composition and reuse of viewpoints across architecture frameworks, in: Proceedings of WICSA-ECSA '12, IEEE Computer Society, 2012, pp. 131–140, doi:10.1109/WICSA-ECSA.212.21.
[67] R. Kazman, M. Klein, P. Clements, N.L. Compton, L. Col, ATAM: method for architecture evaluation, 2000,

Patrizio Pelliccione is Associate Professor at the Chalmers University of Technology and University of Gothenburg, Sweden, Department of Computer Science and Engineering. He got his Ph.D. in 2005 at the University of L'Aquila (Italy) and from February 1, 2014 he is Docent in Software Engineering, title given by the University of Gothenburg. His research topics are mainly in software engineering, software architectures modelling and verification, autonomous systems, and formal methods. He has co-authored more than 100 publications in journals and international conferences and workshops in these topics. He has been on the program committees for several top conferences, and is a reviewer for top journals in the software engineering domain. He is very active in European and National projects. In his research activity he has collaborated with several industries such as Volvo Cars, Volvo AB, Ericsson, Jeppesen, Axis communication, Thales Italia, Selex Marconi telecommunications, Siemens, Saab, TERMA, etc. More information is available at http://www.patriziopelliccione.com.

Eric Knauss is an Associate Professor (Docent) at the Department of Computer Science and Engineering, Chalmers | University of Gothenburg. His research interests focuses on managing requirements and related knowledge in large-scale and distributed software projects. Research topics include Requirements Engineering, Software Ecosystems, Global and Cross-Organizational Software Development, and Agile Methods. More information is available at https://oerich.wordpress.com.

Rogardt Heldal is Professor at the Bergen University College, Bergen, Faculty of Engineering and Business Administration, and he is also Associate Professor (on leave) at the Chalmers University of Technology and University of Gothenburg, Sweden, Department of Computer Science and Engineering. He got his Ph.D. in 2000 at the Chalmers University of Technology. His research topics are mainly in software engineering, software architectures modelling, software processes and empirical studies. He has co-authored more than 60 publications in journals and international conferences and workshops in this theme. He has been on the program committees for several conferences, and is a reviewer for top journals in the software engineering domain. He is active in International and National projects. He is an ISERN member. In his research activity he has collaborated with several industries such as Volvo Cars, Volvo AB, Ericsson, Grundfos, Axis communication and Saab.

Magnus Ågren is a Ph.D. student at Chalmers University of Technology and the University of Gothenburg, Sweden, Department of Computer Science and Engineering. He received his M.Sc. in Computer Science and Engineering from Chalmers University of Technology in 2010. His main research interests are in software engineering; the intersection between software integration practices and software architecture, and systems of systems. Prior to taking up his Ph.D.position, he worked as a software engineer in the telecom industry, at Ascom Wireless Solutions, with embedded systems, IP telephony, and systems interoperability.

Piergiuseppe Mallozzi is a Ph.D. student in Software Engineering at Chalmers University of Technology. He received his M.Sc. in 2015 in Computer Engineering from University of Pisa. He is part of the Wallenberg Autonomous Systems Program (WASP), addressing research on autonomous systems acting in collaboration with humans and selfadapting to the environment. His topic is on collective run-time adaptation of System of Systems, studying ways to assure safety in such complex systems by exploring innovative approaches.

Anders Alminger is a Senior System Architect at Volvo Car Group, Göteborg, Sweden, department of System Architecture and Logic Design. He graduated as M.Sc. in Electrical Engineering at Chalmers University of Technology in 1989. He has more than 20 years experience in automotive electrical system engineering, and his main fields of interest are system architecture and system safety. As a senior System Architect he has led several advance engineering projects in areas of System Safety and System Architecture. Currently he is leading a team responsible for defining the next generation of complete electrical architecture for Volvo Cars.

Daniel Borgentun is an System Architect, Change Agent and a trainer in the agile community. He received his M.Sc.E.E. in 1996 and has since then worked in IT industry, telecom and automotive area with both software, architectures and way-of-working. Daniel is founder and Managing Director for a consultant company and have now an assignment for Volvo Cars. As a senior System Architect he is responsible for defining next generation of complete electrical architecture for the Volvo Cars vehicles.