# Architecting in the Automotive Domain: Descriptive vs Prescriptive Architecture

Ulf Eliasson*, Rogardt Heldal†, Patrizio Pelliccione†, Jonn Lantz*
*Volvo Car Group, Sweden
Email: {ulf.eliasson,jonn.lantz}@volvocars.com
†Chalmers University of Technology | University of Gothenburg
Department of Computer Science and Engineering.
Email: heldal@chalmers.se, patrizio.pelliccione@gu.se

***Abstract*—To investigate the new requirements and challenges of architecting often safety critical software in the automotive domain, we have performed two case studies on Volvo Car Group and Volvo Group Truck Technology. Our findings suggest that automotive software architects produce two different architectures (or views) of the same system. The first one is a high-level descriptive architecture, mainly documenting system design decisions and describing principles and guidelines that should govern the overall system. The second architecture is the working architecture, defining the actual blueprint for the implementation teams and being used in their daily work. The working architecture is characterized by high complexity and considerably lower readability than the high-level architecture.**

**Unfortunately, the team responsible for the high-level architecture tends to get isolated from the rest of the development organization, with few communications except regarding the working architecture. This creates tensions within the organizations, suboptimal design of the communication matrix and limited usage of the high-level architecture in the development teams. To adapt to the current pace of software development and rapidly growing software systems new ways of working are required, both on technical and on an organizational level.**

## I. Introduction

During the past twenty years, vehicles have become more and more robot-like, interpreting and exploiting the input from various sensors to make decisions and finally, carry out actions that were previously performed solely by humans. Modern vehicles can have over 100 Electronic Control Units (ECUs), small to PC-sized computers, together executing gigabytes of software in an optimized network. This evolution of the automotive industry creates new challenges regarding software architecture development and maintenance.

The goal of this work is to investigate the challenges regarding the split of responsibility while architecting software in the automotive industry. We have performed two case studies, on Volvo Car Group (VCG) and Volvo Group Truck Technology (VGTT). Our findings are that for large and complex systems two different types of architectures, with different abstraction levels, are used. A *high-level architecture* with the purpose of guiding and dividing the work between the construction or development groups is designed. Construction groups create the *working architecture* for their components, containing software design and detailed interfaces, and specify the implementation. Often, the different groups have very different views on the meaning of architecture and how it shall be done. One must also note that, in general, these groups are formed by domain experts, who are responsible for developing the mechatronic functions rather than producing the system architecture or optimizing the communication matrix. In general we can identify the following topics of conflict: (i) selecting the optimal solution for the architecture; (ii) selecting the platform to be defined to facilitate the development in the long run; (iii) identifying the priority of the construction groups in order to deliver in time.

There is seldom an obvious connection between the high-level architecture and the working architectures, especially during later phases of the development projects. Artifacts exported from the working architecture are directly used by the construction groups. Because of this, the working architecture is forced to be consistent with the implemented system, and it will inevitably diverge from the original design as the product evolves. Contrariwise, the high-level architecture is only communicated as large documents, supposed to be read by stakeholders. Maintenance of the high-level architecture, while the working architectures evolve, is demanding and not always performed completely; consequently, it is difficult to determine the quality of the high-level architecture description at any time. The high-level architecture has a *descriptive* nature, i.e., it aims at describing network design decisions and the main structures in the system architecture. The working architecture is *prescriptive*, i.e., it defines the detailed design and interfaces, to be used during the implementation of the specific functionalities.

## II. Automotive domain

Historically, software was introduced in cars to optimize engine control. Today not a single function is performed without the involvement of software, and the growth of software is exponential over time. According to industry experts, 80% to 90% of the innovation within the automotive industry is based on electronics. Customers buying cars today do not want to pick a mass-produced product; they want a customized modern transportation system that caters their wishes and needs. Car manufacturers are releasing electric and hybrid electric vehicles, introducing new kinds of software-intense systems in the car for controlling electrical engines, batteries, and charging. Hybrid vehicles also require electrical and conventional combustion engines to work in cooperation.

All of this introduces challenges, both technical and organizational, that previously were not part of automotive manufacturers life. To better understand the organizational challenges, let us consider the case of VCG. As the complexity of the system grew, the development organization followed. This forced the group, responsible for producing the complete electrical system, to be split into several different groups, each of them responsible for architecting the implementation of a specific set of functions. In addition, several new feature-oriented groups have been formed, e.g. for automated vehicle control, user assistance or user interface systems, all of them interacting intensively with the cars mechatronic system.

Consequently, to guide the implementation of the overall system, a new high-level system architecture team was formed, with the responsibility for (i) dividing the system into subsystems, (ii) setting the boundaries for the different groups, (iii) maintaining the consistency of the communication platform, (iv) making sure that the work of the different groups conforms to what is decided. This might seem like the natural things to do. However, there are indications that this construction has caused problems and frustration in the organization, as the work splits between function developers and the central architecture group were difficult to define. As a consequence, the architecture group was losing control over the working architecture and focused more on generic functionality, i.e. requirement engineering of functions shared by several ECUs.

The communication between ECUs is achieved through protocols optimized for limited bandwidth and determinism, driven by price and safety. This communication model, which is common in automotive and other embedded applications, provides a safe network which, however, is monolithic and reluctant to changes, as any change in any data frame may require a complete rework of the communication matrix. Consequently, the system design process used supports parallel development of components only while freezing the interfaces between ECUs. Hence, components can be developed either in-house or by sub-contractors, assuming no or minor updates of the inter ECU communication. Agile development (meaning fast feedback loops and iterative development) is possible only within the boundaries of individual ECUs [3], [2]. Given the importance of the interfaces to the overall development process, formal approval for interface changes is given only a few times per year. Thus, frustration over a working architecture resistant to change, where developers must wait for weeks even for small interface updates, is common. Nevertheless, the rate of change is still too high to allow a up to date high-level architecture.

## III. Research method

In these case studies we used semi-structured interviews as data generation methods: 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. The study was initiated on September 1, 2014. The interviews have been carefully prepared by gathering knowledge from the companies and the domain. To better prepare the interviews we organized two workshops with representatives of the companies that have been involved in the study. The first workshop was organized on September 22, 2014 with all the authors of the paper and one experienced ($>$10 years of experience) software architect for each company. The purpose of the meeting was to gather background information on the organizations and on their context. This knowledge has been exploited to improve the design of the study. The second workshop was organized on November 4, 2014 with the same participants as at the first workshop. The purpose of the workshop was to organize and schedule the interviews. The representatives of the companies helped us to identify software architects to interview within the two companies and to have their commitment. After the second workshop, we performed 11 interviews ending on December 19, 2014. The research questions we have defined are:

**RQ1**: What is the role of architecture in the automotive domain?

**RQ2**: How does the organization affect the architecture?

In order to mitigate the threat to validity in this study, we followed the guidelines for conducting and reporting case study research in software engineering [7], [8].

As for what concerns *construct validity*, we performed semi-structured interviews by following a questionnaire that has been defined by exploiting knowledge collected through two workshops, as described above. The workshops enabled also to identify and set a proper language to be used during the interviews. In addition to that, interviews have been performed by the first author, who is an industrial Ph.D. working at VCG and therefore expert of the domain. This permits us to be confident that interviewed architects correctly interpreted the questions, and, in turn, we correctly interpreted their answers.

As for what concerns *external validity*, having two authors from VCG could be seen as a threat to validity due to the risk of the study becoming heavily focused and influenced by the situation at one company. However, having two authors from academia and being aware of the risk of such a bias, permitted to actively counter it. Moreover, the first author is a VCG employee, recruited as a Ph.D. student funded by VCG. This puts him in the position of being a perfect observer, without having strong influences and bias from the company. Moreover, we performed the study in two different companies in order to reduce this bias.

## IV. Results

In this section, we present the results from the data collected through our interviews. They provide answers to our two research questions.

**RQ1**: What is the role of architecture in the automotive domain?

The companies in our study have two types of architectures: a high-level architecture and a working architecture.

***High-level architecture***: The high-level architecture differed between the two companies. However, both companies agree that the high-level architecture should contain the principles, rules, and patterns that should govern the rest of the design work, both for what concerns the high-level architecture and the rest of the work, down to implementation.

To better explain the concept we report two citations from high-level architects working at VCG:

*"The role of the [high-level] architecture at Volvo is rather to guide than to be a direct correlation with the design."*

*"What is the purpose of the high-level architecture description? It is above all to guide."*

and one citation from a high-level architect at VGTT:

*"[About high-level architecture description] ... so it is more focused on what patterns and principles that have been applied."*

Both companies also agree that the high-level architecture should provide a logical design of the system to support the construction of the working architecture as well as dividing the work between the different groups. However, currently only VCG has a logical design in their high-level architecture description. High-level architects at VGTT currently do not provide a logical design as part of the architecture description, but this is something they want to do in the future to get better control over where things go in the design. The high-level architecture is described in documents. At VCG, this is a big word document with text and UML-models as visualization. This document takes considerable effort for the architecture team to release, consequently it is updated only two-four times per year. The focus on this level is on the platform that runs across projects. VGTT high-level architects currently produce a document with architectural principles and patterns, so it is a considerable smaller document than the one produced at VCG. They produce also supporting documents such as Powerpoints with the topology but also parts of the principles so they can be used for presentations when discussing architecture concerns.

Having the high-level architecture clearly separated from the working architecture is seen as a good thing by architects of both companies since it gives them freedom to reason about the future and try changes out.

***Working architecture***: Both at VCG and VGTT the working architecture is stored in a custom made proprietary tool. At VCG, the logical architecture components identified by the high-level design are broken down into smaller logical components and deployed on software components. Logical components are also connected to requirements, as well as to signals they receive and/or send. The tool also contains deployment information, i.e., information about the mapping between software components and ECUs. Requirement specifications to be sent to suppliers can be generated from the tool. The output is used by the in-house teams to generate the scaffolding for the developers to implement their solutions within. Finally, it generates Autosar ECU-extracts used for the integration. For this reason, developers are forced to keep the working architecture updated and consistent with the real implementation, as the software will not build or even integrate otherwise. Finally, as the working architecture is updated within the tool the new scaffolding can be generated and used without overwriting what the developers have previously done.

VGTT currently does not have logical components in the high-level architecture description. Therefore, logical components are created at the level of the working architecture. Consequently, at this level requirements as well as signals are attached to these components. The tool also contains the physical topology with the ECUs and the mapping between the ECUs and logical components. VGTT does not have software components in its tool but the concept of logical components maps, in the majority of the cases, directly to software components. This means that it is possible to export ECU-extracts from the tool to be used by the software teams for the integration. However, because there are still some small differences between the logical components in the tool and the software components in the code, manual work to check for changes and keep the ECU-extracts up to date with the current design is required.

Different teams of architects are responsible for the high-level and working architectures. As described above, at VCG the high-level architects are working on the logical architecture, but they are also adding design details that should be included in the working architecture. This leads to a high-level architecture description that is rather large and complex, thus a bit of the purpose of the high-level architecture vanishes. The working architecture has matured over several years. In many ways, the working architecture contains all the design information of the high-level architecture, but with a lot more details. It makes it larger and more complex than the high-level architecture, but it is always kept up to date based on the products. So when the project has started, the high-level architecture is often only updated to keep it up to date with what have happened in the working architecture.

**RQ2**: How does the organization affect the architecture?

The interviewees said that having a separate group for high-level architecture created tension between high-level architects and the construction groups, which include people involved in the construction of the working architecture. Members of the construction groups saw a need for a high-level architecture, but they identified two main problems:

- high-level architects lack an understanding of the current situation and the system under implementation;
- high-level architects focused too much on what might be good for the future, while neglecting a concrete vision of what is the best solution for the current situation.

To have a flavor of the emerging tension, we report a quote from a low-level architect at VGTT:

*"But sometimes you feel that the [high-level] architecture-group thinks that we should change*

*everything already done. While we are more focused on that we have to solve something to the project, and yes, what we have is maybe not the optimal solution but it is what we have."*

Similarly, the high-level architects themselves thought that the low-level architects have these two main limitations:
• they are very focused and then they miss an overall picture;
• they are too focused on short-term solutions: the best solution in the short run might cause problems in the future.

At VCG, low-level architects feel that the high-level architecture group often make design decisions that should not be of their concern. In fact, they do not have enough information so early in the projects, as needed for making some of these design decisions. In general, there is a need to change what was first thought to be the best solution; these extra details added early, contribute to making the work more tedious instead of the opposite. Summarizing, it emerged that the high-level architects have a more long time perspective, while the architects working with the working architecture have more short-time perspective and pragmatic view; they want the product to be ready in time for strict deadlines. Moreover, both groups criticize the other group of being not so open to discussion. Each group feels that in the case of conflicts and discussions, the primary goal was to defend their position, instead of trying to solve the conflict and looking for the best solution for the company.

## V. Conclusion and Discussion

The role of the high-level architecture, if asking the architects themselves, is to serve as a set of guidelines and to identify the boundaries for the detailed design. In reality, this architecture is often, if not ignored, at least not in the mind of the engineers doing the low-level architecture on a daily basis. It emerged that low-level architects seldom, if ever, read the documentation produced by the high-level architects. Now, this does not mean that the work of the high-level architects is without value. However, having the high-level architects in a group of their own, separated from the other groups creates tensions between the different groups and a power struggle.

The different organizations have different competencies, attitude, characteristics. However, new problems emerge. In fact, we can confirm that we found many architecture antipatterns. We found the *Goldplating* antipattern [4] since high-level 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* [4]: the high-level 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.

Another interesting finding is that a high-level architect did not feel that there was a difference between using in-house developers and subcontractors. This might be due to the fact that the architects are more distant from the product. In the case of the people working with working architecture, this is not the case as we also found in our work [1]. They find it very frustrating to wait for part of the system to be integrated.

For the future, we want to explore both organizational and technical possibilities for tighter cooperation between architecture levels, and to measure effects such improvements would have. On the technical side, one idea is to define a framework able to automatically generate high-level views from the low-level architecture. The challenge here is to support multiple views, each devoted to showing only what is relevant for respective stakeholders. Moreover, both high-level and low-level architects need ways to perform early validation of their solution and to sketch and try different visions of how the future systems should look like, to understand the effect of design decisions affect the architecture.

On the organizational side, we found a need to improve the communication between different groups, for instance by making teams more cross-functional. Today there are several levels between architects and implementers and at some of these steps the connection is not very tight. This leads implementers to think that the high-level architects are sitting in their cloud above, without having any connection to the reality. On the other hand the high-level architects 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 [5], [6], high-level architects should be first of all "Extrovert" architects (conceptually related to the external focus of [4]), i.e. devoted to communicating the architectural decisions and knowledge to the other stakeholders.

## Acknowledgements

This work is partially supported by the ASSUME and NGEA Vinnova projects, the Software Center initiative (software-center.se - project n.4, "Developing Complex Products - Architecture perspective"), VCG and VGTT.

## References

[1] H. Burden, R. Heldal, and J. Whittle. Comparing and contrasting model-driven engineering at three large companies. In *Proceedings of ESEM '14*, pages 14:1–14:10. ACM, 2014.

[2] U. Eklund, H. H. Olsson, and N. J. Strm. Industrial challenges of scaling agile in mass-produced embedded systems. In *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*, number 199 in Lecture Notes in Business Information Processing, pages 30–42. Springer International Publishing, Jan. 2014.

[3] U. Eliasson, R. Heldal, J. Lantz, and C. Berger. Agile model-driven engineering in mechatronic systems - an industrial case study. In *MODELS 2014*, volume 8767 of *LNCS*, pages 433–449, Valencia, Spain, Oct. 2014. Springer International Publishing Switzerland.

[4] P. Kruchten. What do software architects really do? *Journal of Systems and Software*, 81(12):2413 – 2416, 2008.

[5] P. Lago, I. Malavolta, H. Muccini, P. Pelliccione, and A. Tang. The road ahead for architectural languages. *IEEE Software*, 32(1):1, 2015.

[6] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang. What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering*, 39(6):869–891, 2013.

[7] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164, Apr. 2009.

[8] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.