IDENTIFYING AND MANAGING KEY CHALLENGES IN ARCHITECTING SOFTWARE-INTENSIVE SYSTEMS

Peter Wallin

2011
IDENTIFYING AND MANAGING KEY CHALLENGES
IN ARCHITECTING SOFTWARE-INTENSIVE SYSTEMS

Peter Wallin

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid
Akademin för innovation, design och teknik kommer att offentligen försvaras
fredagen den 18 mars 2011, 09.00 i Beta, Mälardalens högskola, Västerås.

Fakultetsopponent: professor Antony Tang, VU University
Amsterdam and Swinburne University of Technology
Abstract

In many traditional industry applications, such as automotive, process automation and manufacturing automation, software plays a crucial role as an enabler for the introduction of new functionality and retaining competitiveness. The system and software architecture plays an important part in ensuring the systems’ qualities. However, the design of the architecture may be neglected during system development, whilst development efforts are centered on implementing new functionality. The architecture is supposed to support and enable key quality attributes such as safety, reliability, maintainability and flexibility, and so on. This thesis identifies some of the key issues in architecting these software intensive systems. In total, 21 issues have been identified; examples of these issues are (1) there is a lack of process for architecture development, (2) there is a lack of method or model to evaluate business value when choosing architecture, (3) there is a lack of clear long-term architectural strategy, and (4) processes and methods are less valued than individuals’ knowledge and competence. Through a series of workshops, root causes were identified for a selection of these issues. Based on these root causes, five success factors were identified. The success factors are (1) define an architectural strategy (2) implement a process for architectural work (3) ensure authority for architects (4) clarify the business impact of the architecture and (5) optimize on the project portfolio level instead of optimizing each project. In an attempt to provide a possible solution to some of the issues, a method has been created to evaluate how new functionality is successfully integrated into an existing architecture. The method is a combination of the Architecture Tradeoff Analysis Method, ATAM, and the Analytical Hierarchy Process, AHP. The method firstly supports a structured way of listing system goals, and secondly, it also supports design decision-making. Since several issues relate to the organization and are affected by management, a comparison was made between the view of management and architects. This study revealed that one cause for the lack of focus on architecture could be that the existing performance measurement systems used by management all focus on the later phases of development when the architecture is already set.
Abstract

In many traditional industry applications, such as automotive, process automation and manufacturing automation, software plays a crucial role as an enabler for the introduction of new functionality and retaining competitiveness. The system and software architecture plays an important part in ensuring the systems’ qualities. However, the design of the architecture may be neglected during system development, whilst development efforts are centered on implementing new functionality. The architecture is supposed to support and enable key quality attributes such as safety, reliability, maintainability and flexibility, and so on. This thesis identifies some of the key issues in architecting these software intensive systems. In total, 21 issues have been identified; examples of these issues are (1) there is a lack of process for architecture development, (2) there is a lack of method or model to evaluate business value when choosing architecture, (3) there is a lack of clear long-term architectural strategy, and (4) processes and methods are less valued than individuals’ knowledge and competence. Through a series of workshops, root causes were identified for a selection of these issues. Based on these root causes, five success factors were identified. The success factors are (1) define an architectural strategy (2) implement a process for architectural work (3) ensure authority for architects (4) clarify the business impact of the architecture and (5) optimize on the project portfolio level instead of optimizing each project. In an attempt to provide a possible solution to some of the issues, a method has been created to evaluate how new functionality is successfully integrated into an existing architecture. The method is a combination of the Architecture Tradeoff Analysis Method, ATAM, and the Analytical Hierarchy Process, AHP. The method firstly supports a structured way of listing system goals, and secondly, it also supports design decision-making. Since several issues relate to the organization and are affected by management, a comparison was made between the view of management and architects. This study revealed that one cause for the lack of focus on architecture could be that the existing performance measurement systems used by management all focus on the later phases of development when the architecture is already set.
En allt större del av utvecklingskostnaden för bilar, anläggningsmaskiner och automationssystem består idag av mjukvara. I en modern bil idag finns det miljontals rader kod och flera kilometer med kablage som kopplar samman de upp till 70 datorerna som kan finnas tillgängliga. Allt detta gör att de mjukvaruintensiva systemen blir mer och mer komplexa och svåra att utveckla.

Elektroniksystemet i ett fordon delas ofta mellan flera modeller och varianter. Det gör att systemet måste kunna anpassas för att användas både i en billigare bil med endast enklare basfunktioner, och i en bil som är full med avancerade funktioner. Samtidigt är det svårt att värdera hur mycket som ska delas då det kan betyda att den billigare bilen får onödiga kostnader i form av mer avancerade komponenter än vad som krävs för basfunktionaliteten. Arbetet med struktur och gränssnitt är centrale när man utvecklar elektronik och mjukvara för fordonsindustrin. Vi har dock observerat att området är eftersatt och behöver vidareutvecklas.


Genom rotorsaksanalyser av nyckelfaktorerna har vi tagit fram fem stycken framgångsfaktorer. Exempel på framgångsfaktorer är att det behövs en tydlig och långsiktig strategi för utvecklingen av dessa system och att man ska optimera produktparteföljen istället för enskilda projekt.
Som lösning på ett av de identifierade problemen har vi tagit fram en metod för att utvärdera hur nya funktioner kan integreras i ett redan existerande elektroniksystem. Metoden tillhandahåller ett strukturerat och effektivt sätt att resonera kring betydelsen av olika systemegenskaper, såsom flexibilitet, säkerhet, pålitlighet och servicebarhet.
Acknowledgements

Many people deserve to be acknowledged in this journey. First of all I would like to thank my supervisor Professor Jakob Axelsson, for always finding time to guide me, both in the world of research and in the automotive industry. I would like to thank my assistant supervisors Dr Stig Larsson and Professor Christer Norström for all the invaluable input to my research. The studies made in this thesis would have been impossible without the great support from the participating companies and their representatives, Björn Villing at Volvo 3P, Nils-Erik Bånkestad and Patrik Lindblom at Volvo Construction Equipment, and Jakob Axelsson at Volvo Car Corporation, now at SICS. A special thanks to all people that has been participating in the studies and through these gave me invaluable input. Without your input this thesis would be without any results.

I would also like to thank the people involved in the research schools AR-TESS and SAVE-IT. They have provided great collaboration and many interesting trips. All the great people in the research group BESS deserves to be acknowledged for great collaboration and discussions during the years.

I would like to thank all my current and former colleagues at the School of Innovation, Design and Engineering, Anton Jansen, Sara Dersten, Magnus Larsson, Anders Wall, Thomas Nolte, Monica Wasell, Carola Rytersson, Else-Maj Silén, Susanne Fromnå, Åsa Lundkvist, Hans Hansson, Ivica Crnkovic, Jörgen Lidholm, Fredrik Ekstrand, Ylva Wretås, Johan Kraft, Daniel Sundmark, Hongyu Pei-Breivold, and all the ones I have forgotten.

For the six month I spent at the Software Engineering Institute during 2009 I would like to thank Ipek Ozkaya for great collaboration. A special thank to Jörgen Hansson for showing all the great spots in Pittsburgh and for all the interesting discussions about what things to buy. My wife will always blame you for the purchases of mountain bike, power tools, cookware, camera lenses, headphones, and much more.

A special thank to Stefan Cedergren, Joakim Fröberg, Markus Lindgren, Ana Magazinius, and Håkan Gustavsson for great collaboration and for all the fruitful discussions we have had, mostly about other things than just work. Håkan and I started our Ph.D. studies the same date in March 2006 and now
it seems that we will finish the same day, although I am about four hours ahead. I would also like to thank Andreas Hjertström for making sure that I do get to the early morning swim now and then, and of course for all the fun we had during the years.

A special thank you to my cousin Fredrik Wallin, for all the great dinners, discussions and more that we had during the years. Actually, Fredrik was the one that talked me into starting my Ph.D. studies in the first place. What convinced me was the reasoning “Can Fredrik can I”. And with the same reasoning but with my name instead, I am sure I have brought more people into the world of Ph.D. studies.

Many thanks to my mother, Christina, my father, Bernt, my sister, Josefin and my parents in law, Eva and Bengt, for all the great support during the years.

Finally and most importantly I would like to thank my wife Katarina and our newly born daughter Annie for always supporting me despite my sometimes over optimistic plans. I love you both.

Peter Wallin

Västerås, February 2011
List of Papers

Publications Included in the Thesis

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.


I was the main author of this paper and had help from Ana Magazinovic from Chalmers in collecting data, and from my supervisor Jakob Axelsson in the analysis part. Jakob also wrote most of the validity section of the article. This paper received the award for best paper at the conference.


I was the main author of this paper and had help from Stefan Johnsson during data collection and analysis. Jakob Axelsson helped to analyze the data and also wrote major parts of the validity section.


I was the main author and was helped with data collection and data analysis by Stig Larsson and Joakim Fröberg. I did most of
the writing with some help from Stig, Joakim and Jakob. Jakob also did the statistical analysis.


This paper was a collaborative effort with Ipek Ozkaya. I did most of the planning and the interviews were done together. In writing the paper I wrote most of the methods section together with parts of the results and contribution.


I was the main author and did all writing except for the related work part of Performance Measurements in Product Development. The analysis was done together with Stefan Cedergren and Stig Larsson.


I was the main author and the paper was written in close collaboration with Joakim Fröberg, who provided most of the ideas and thoughts on ATAM. I provided the thoughts around AHP and how they could be used together. The writing was divided equally between the two of us. This paper was also presented at Real-Time in Sweden (RTiS), Västerås, Sweden, August 2007.

Reprints were made with permission from the relevant publisher.
Other Related Publications by the Author


Contents

Abstract.............................................................................................................................i

Swedish Summary – Svensk Sammanfattning.............................................................. iii

Acknowledgements........................................................................................................... v

List of Papers ................................................................................................................... ix
  Publications Included in the Thesis................................................................................ ix
  Other Related Publications by the Author....................................................................... xi

Part I

Chapter 1  Introduction.................................................................................................. 3
  1.1  Software-Intensive Systems .................................................................................. 4
  1.2  Automotive E/E System Architecture .................................................................. 5
  1.3  Automotive Development Context ....................................................................... 7
  1.4  Other Software-Intensive Industries .................................................................... 9
  1.5  Thesis Overview ................................................................................................. 10

Chapter 2  Research Scope .......................................................................................... 11
  2.1  Motivation and Positioning of the Work ............................................................. 11
  2.2  Research Questions ............................................................................................ 12
  2.3  Methodology ....................................................................................................... 13
    2.3.1  Exploratory Case Study (RQ 1) .................................................................... 14
    2.3.2  Survey with Structured Interviews (RQ 2) .................................................. 16
    2.3.3  Combining Data from Exploratory Studies (RQ 3) .................................... 17
    2.3.4  Combining Existing Evaluation Methods (RQ 4) .................................. 17

Chapter 3  Related Work ............................................................................................ 19
  3.1  Software Architecture ......................................................................................... 20
  3.2  Software Architecture Evolution ......................................................................... 22
  3.3  Software Architecture Principles ....................................................................... 23
    3.3.1  Software Product Lines ............................................................................... 23
    3.3.2  Reference Architectures .............................................................................. 25
    3.3.3  Documenting Architectures ......................................................................... 25
    3.3.4  Evaluating an Architecture .......................................................................... 26
    3.3.5  Architecture Process .................................................................................... 28
  3.4  Challenges in Automotive Software Engineering ................................................ 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>Organization</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Management</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Results</td>
</tr>
<tr>
<td>4.1</td>
<td>Key Issues with Success Factors</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Key Issues</td>
</tr>
<tr>
<td>4.1.2</td>
<td>General Software-Intensive System Issues</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Success Factors</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Discussion Related to RQ 1</td>
</tr>
<tr>
<td>4.2</td>
<td>Architecture-Centric Practices to Guide Software Evolution</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Discussion Related to RQ2</td>
</tr>
<tr>
<td>4.3</td>
<td>Limiting Practices in Developing Software-Intensive Systems</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Discussion Related to RQ3</td>
</tr>
<tr>
<td>4.4</td>
<td>Effective Decisions</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Discussion Related to RQ4</td>
</tr>
<tr>
<td>4.5</td>
<td>Relationships between Results</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Relationships between Papers</td>
</tr>
<tr>
<td>4.6</td>
<td>Validity</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Conclusions and Future Work</td>
</tr>
<tr>
<td>5.1</td>
<td>Industrial Footprint</td>
</tr>
<tr>
<td>5.2</td>
<td>Limitations and Self-criticism</td>
</tr>
<tr>
<td>5.3</td>
<td>Future Research Directions</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Bibliography</td>
</tr>
</tbody>
</table>

Part II Included Papers

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 7</td>
<td>Paper I</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>Paper II</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Paper III</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Paper IV</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Paper V</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Paper VI</td>
</tr>
</tbody>
</table>
Part I
Chapter 1  Introduction

Software is now playing a greater part in what used to be purely mechanical products. The products are still largely mechanical, but without the software it would be impossible to get the same functionality. For instance, in the automotive industry, more than 80% of the innovations in a car come from computer systems (Broy et al., 2007; Grimm, 2003; Leen and Heffernan, 2002). One reason for the significant increase in software and electronics in cars is customer demand for new safety and convenience functions, such as adaptive cruise control, blind spot detection, forward collision avoidance, lane departure warning and many more. Furthermore, the use of software and electronics is necessary to cope with new regulations on emissions. For the Original Equipment Manufacturer (OEM), software and electronics aid in test procedures since many tests can be automated. It further provides the OEM with flexibility, managing variants by changing the software parameters, instead of using different mechanical components. One example of this is software controlling the engine, which can be parameterized differently for different engine models.

In other fields, such as the aircraft industry, software controls most flight capabilities. In a modern airplane, there is no mechanical wiring from the cockpit to control each wing; instead electrical signals control the rudder maneuvers. For telecom systems, several components that used to require manual reconfiguration have been replaced by smaller and more flexible components implemented in software.

In the field of power transmission and distribution, mechanical protection relays have now been implemented in software. The major advantage is better control and monitoring. Another advantage is that the same hardware can be used for many different systems. This increases purchasing volumes and reduces the number of physical variants that have to be maintained and supported.

An increasing amount of software and electronics in what used to be traditional mechanical products is now an enabler to remaining competitive and developing products efficiently and effectively. An important factor in dealing with this inherent complexity is the use of a system and software architecture. The architecture describes the characteristics of the system, includ-
ing both internal and external properties. The architecture is typically an enabler for both efficiency and effectiveness in the development of software-intensive systems, but is not directly linked to customer needs. For example, the architecture can increase the agility of upcoming product releases, in order to cost-effectively satisfy future customer needs.

Initial discussion with practitioners indicated that regardless of the benefits described above, those gained by putting more effort into the system and software architecture development, it is still not done. However, the reason for not focusing on the software and system architecture development remained unclear.

1.1 Software-Intensive Systems

A number of companies and organizations have been involved in this research, either by participating in interviews, answering surveys or attending workshops. A common theme for all companies is that they develop products that involve a great deal of software, although they are not considered typical software companies.

The products these companies develop are, in many ways, traditional mechanical products. However, in recent decades more and more software has been included to maintain competitiveness. The motivation for introducing more software in these types of systems is first that some of the new functionality requires to be implemented in software. A second motivation is that mechanical components can in some cases be removed and replaced by software, hopefully at a lower cost.

Further characteristics of these products are that they are often safety critical, which means that a software failure could have severe consequences. Furthermore, these products are all long-lived systems that have an expected life-time of over ten years.

This type of system is often referred to as software-intensive. IEEE (2007) defines a software-intensive system as:

Any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole.

To us, a software-intensive system is a system that is highly dependent on software to function. However, it also has a close relationship with the electronic and mechanical parts of the same system. A software-intensive system thus contains software, electronics and mechanics, and combining these
three entities makes the product include more features at a lower cost than would be possible with only mechanical components. Typical software-intensive systems are cars, medical devices, trains, airplanes, and mobile phones.

The automotive industry typically makes products that are software-intensive. Since several of the studies in this thesis have been done at automotive companies, an overview of the characteristics of these products with a special focus on the architecture is presented below.

1.2 Automotive E/E System Architecture

The terms architecture and system are frequently used when developing automotive electrical and electronic (E/E) systems. However, there is not always a common understanding of what is included in an architecture. Practitioners’ in the automotive industry usually refers to the IEEE definition of architecture which is:

"The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" (IEEE, 2000).

The E/E architecture in vehicles includes sensors, actuators and control units, as well as other electrical and electronic components. The architecture does not specify the details of each component, rather that there should also be, for instance, a sensor that measures the distance to objects in front of the vehicle, i.e. whether this is a radar, high speed camera or a laser range finder is not necessarily part of the architecture. One of the most important characteristics of an architecture is to define the interfaces between these different components and modules.

Furthermore, the physical network, software and wiring is part of the E/E architecture. One reason for this is the tight coupling between electrical hardware and software. For instance, a braking application is very tightly bound to the electrical hardware for which it is tested and developed. A change of actuators or other mechanical or electrical components in such an application would likely generate a change of software functionality.

According to the IEEE definition, the architecture also includes guiding principles and rules about the design of the system. A typical rule could be the type of communication protocols that should be used. An example of a design principle could include what architectural pattern should be used. Architectural patterns are further described in the related work section. It
should also include guidelines about how the system should evolve. An automotive electronic system architecture can be described in many ways, using different views as stipulated by the IEEE 1471 standard (IEEE, 2000).

Different views show different aspects of the architecture. By using views, stakeholders can get more tailored information about their specific area of concern. For example, a view showing the placement of physical components will be of extra interest to people working with the overall design and optimizing the available space in the vehicle. Below are five views that are fairly common in the automotive industry. The views have some similarities with the 4+1 view proposed by Kruchten (1995), that is further discussed in 0.

Functional view
The functional view describes all the functionality available to the user. Naturally, one user of the vehicle is the driver, but maintenance staff and production might also be part of the user scheme.

Physical/electrical view
A commonly used view is the physical and electrical view that shows where the different Electronic Control Units, ECUs, are physically placed and also how they are connected to each other via different networks such as CAN, FLEXRAY and MOST. An example of this view is shown in Figure 1. This view could also show the electrical distribution in form of cabling, fuses and power generation, and storage.

Logical view
Another view that is important is the logical view. It describes how the functionality is decomposed into a set of interacting components.

Figure 1. Physical view of the communication architecture of the Volvo XC90.
Software architecture view

The software architecture view describes how the internal software is structured. On a high level, the software architecture view can seem similar to the logical view. However, the software architecture view also describes the operating system, how the software is partitioned into application layer, runtime environment and so on. In the automotive industry it is quite common that the OEM specifies the software architecture view but leaves the actual implementation to suppliers. This is further described in Section 1.3.

Deployment view

The deployment view describes how the logical view is mapped to the different physical nodes (ECUs). The deployment view therefore connects the software with the hardware, describing the allocation of software in the different ECUs.

In the automotive industry, the physical and electrical views are usually the ones that will get the most attention. This is mainly due to the fact that it is easier to understand the placement of real physical components instead of the sometimes more abstract logical view.

Another reason for the focus on the physical and electrical views is that they are determined and limited by the design of the vehicle. In a vehicle the available space to mount physical components is fairly limited as well as the space for electrical cables and connectors. However, the logical architecture might increase dependencies between different ECUs. An increased number of dependencies will most likely cause the system to be more complex, and make it harder to remove or change components, or to add new functionality. On the other hand, the physical and electrical views need to conform to manufacturing and service.

The above described views are further explained by Broy et al. (2007). They describe the interaction between these views. First, the logical view is set by the functional view. Then, the logical view decides the software architecture view and the physical/electrical view. The interaction between the logical view and the physical/electrical view is described in the deployment view. In our experience, however, key elements of the electrical/physical view are often decided early and the other views have to comply with them.

1.3 Automotive Development Context

In the automotive industry, many vehicle models share the same platform and architecture. The architecture has to comply with requirements, not only
from different models within one brand, but also requirements from different brands. For example, when Ford Motor Company owned Volvo Cars, a high-end Volvo car could share architecture with a low-end Ford car. This meant that the architecture had to be scalable to support both; not making the architecture for the Ford model more expensive, while at the same time still supporting the greater functionality required by Volvo.

As described above, several models share the same platform and architecture. This causes the evolvability of the architecture to be a key concern. Figure 2 provides an example of how the sharing is actually done between three different car models. The first introduced model on platform A is the S80. The typical life-cycle for each model is around seven years. For each new model year there is a minor update (a major update is usually done three to four years into each life-cycle but that is not illustrated in the figure). As the platform in this example is shared between three different models and each model is introduced separated by a year, this causes the platform to evolve for a period of at least ten years. (It should be noted that this is an example and that these characteristics might differ in other cases.)

There are many parameters to consider when deciding whether it is beneficial to share an architecture between models or not. One reason for sharing architecture could be that quality is increased when reducing the number of architectures, as more development time can be used for each architecture. On the other hand, quality may be reduced if the architecture becomes more complex when enabling support for different models. If the architecture is customized for a particular model, the development cost for that particular architecture will most likely be lower but, considering that sharing the architecture means that development costs will be shared between many models, it will probably generate a better business case for the company as a whole. Furthermore, production costs may be lower when sharing the architecture since larger quantities of the same components can be purchased.

Common to all automotive developers is that they purchase subsystems from different suppliers and integrate these systems. Much of the software is not made in-house and is usually included in the specific electrical hardware. In a braking system, for example, both software and electrical hardware are
bought from the same supplier. However, the AUTOSAR¹ (www.autosar.org, 2010) initiative might enable software and electrical hardware to be purchased from different suppliers or the software to be developed in-house. Automotive development is also characterized by relatively long lead times, where the start of production is typically around four years after the start of development.

Another factor that puts more constraints on development is that suppliers in the automotive domain are usually very large. This makes some of the OEMs relatively small compared to some suppliers. The supplier tends to strive to use a design developed for some other OEM when offering to sell a component. The OEM, on the other hand, usually prefers to have the function developed exactly according to its own defined requirements. By choosing something already developed for another OEM, costs might be reduced and the quality increased. This might, however, limit the design space for the OEM.

1.4 Other Software-Intensive Industries

In our opinion, the characteristics described in Section 1.2 and Section 1.3 are, to a great extent, general to software-intensive systems. One characteristic that has not been identified to the same extent at the non-automotive participating companies is the complex supplier structure. This could be explained by the relatively large volumes in which cars are produced, compared to, for example, process automation systems or air control systems. The reason that volumes affect the supplier complexity is that each Euro saved per vehicle by using another supplier will have a great impact on the company’s overall profit. Furthermore, the automotive industry is extremely competitive, forcing OEMs to constantly cut costs. Naturally, it is not only the automotive industry that is under constant pressure due to competitiveness, but the margins are generally lower, at least compared to the other domains discussed in this thesis.

Another difference is that the automotive industry develops little software itself, compared to the other studied companies. However, there is a tendency for automotive companies to consider software development as a core competence and as being necessary to have in-house. Again it seems as if production volumes could play a major part in this; among the three automo-

¹ AUTOSAR (AUTomotive Open System Architecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers.
tive manufacturers studied, the one with the largest volumes develops the least software itself. At the same time, the one with the smallest volumes develops most software itself. This could also be related to the amount of specialization required; the OEM with the lowest volumes develops heavy machinery and could require more specialized components. This could limit incentives for a supplier to develop this component since it is harder to gain any synergy effects from specialized components. The suppliers usually strive to sell the same component with minor modifications to the different OEMs.

Overall, despite the above mentioned differences, according to our observations it seems that company culture and geographical location have a greater impact than the differences in domain.

1.5 Thesis Overview

The aim of this thesis is to highlight issues related to the architectural development of software-intensive systems, finding the underlying causes of issues that are considered the most significant for the development of software-intensive systems and proposing actions to deal with these issues.

The rest of the thesis is organized as follows: Chapter 2 presents the motivational aspects and the research question. This chapter also includes the research methods used to address each research question. In Chapter 3, work related to this thesis is presented. Chapter 4 presents the results for each research question and a discussion of the relationship between the individual results for each research question. It concludes with a discussion of the validity of the results. Chapter 5 concludes the first part of the thesis, presenting the conclusions and possible future research directions.
Chapter 2 Research Scope

This chapter explains the motivation behind the research, which is used to form concrete research questions. Furthermore, the overall research methodology used will be explained in this chapter.

2.1 Motivation and Positioning of the Work

Due to the increasing amount of software and electronics, decisions made during E/E system development have become more and more important. An incorrect decision in the architecture may now increase costs significantly, both during development and also in later phases, such as maintenance. However, little research has been done into how such decisions are made today.

The architecture itself does not provide any customer functionality. Instead the design of the architecture affects qualities such as flexibility, dependability, serviceability, security and maintainability. These quality attributes are hard to evaluate, both compared to each other and with their cost. The benefits of enhancing serviceability might not be seen until late in the life cycle of the vehicle and it is extremely hard to quantify the value of such attributes. Even if a certain quality attribute is valued, it is difficult to know what particular design decisions will enhance the desired quality.

Griss (1995) points out the importance of non-technical factors in software reuse and, since the architecture is often reused, the same principles should apply to architecture development:

"In almost all cases, a simple architecture, a separate component group, a stable application domain, standards and organizational support are the keys to success. Correct handling of these (largely non-technical) issues is almost always more critical to successful reuse than the choice of specific language or design method, yet too many experts choose to ignore these factors."

As the statement above suggests, non-technical factors seem to influence architecture development. The studies are thus not limited to technical parameters.
A more concrete motivation, and also the motivation for the whole research project, is that practitioners within the automotive industry in particular have stated that there are problems with the current development of system and software architecture.

2.2 Research Questions

This section presents the different research questions that this thesis investigates. The first research question is posed to investigate the key issues when developing architectures for software-intensive systems. The aim of starting with an exploratory study was to gain insights that would set the direction of future contributions.

RQ1: What are the key issues that affect real-world decisions regarding a vehicle’s electrical and electronic system architecture?

This question is quite broad so three sub-questions were posed to get deeper understanding. The first two relate to root causes of the issues and how to deal with these issues.

What are the root causes of the identified issues?

Is it possible to elicit success factors based on the identified root causes?

The question is limited to the automotive industry and the last sub-question was posed in an effort to generalize the results:

Are these issues general to the development of software-intensive systems?

The findings from the first research question served as input to the other three research questions.

RQ2: Which architecture-centric practices are used in industry to systematically guide evolution?

The reason for posing the above question was based on the observation from the initial study that long-term architectural planning was fairly limited. The literature describes several architectural practices, such as evaluation, documentation, and explicit design. The aim was to understand to what extent these practices were used at these companies, as well as investigate whether
some practices were not used at all. Using some of these architectural prac-
tices can aid or form a prerequisite for successful long-term architectural
planning.

A common theme during the data collection for both RQ1 and RQ2 was the
lack of management support and understanding. This was especially true in
the companies whose products used to be more mechanical by nature and
had more recently transformed into including more software. To get a more
balanced picture, instead of mostly the view of the architects, senior manag-
ers were included as well.

**RQ3:** What are the causes of the mismatch between architects and man-
agement in the development of software-intensive systems?

The first three research questions primarily focus on identifying issues and
problems, apart from the identified success factors. In the last research ques-
tion, the aim was to investigate whether it is possible to construct an evaluation
method that can be used to evaluate different architectural design alter-
natives.

**RQ4:** How can effective decisions be made when adding new functionality
to an existing electrical/electronic system architecture?

### 2.3 Methodology

This section describes the different methodological approaches used in the
thesis. A number of different research methods were applied to investigate
the research questions presented in the previous section. An approach with
exploratory interviews and semi-formal questions was used to obtain a broad
understanding and find the key issues. Secondly, to find the most significant
issues and to be able to generalize the data, a survey was distributed to a
broader audience. Thirdly, root cause analysis workshops were used to find
causes to issues and to identify solution to issues. To investigate the architec-
ture-centric practices used to guide evolution, structured interviews were
used. The mismatch between architects and management was investigated
using the results of two different studies using exploratory interviews. Last-
ly, to develop the proposed evaluation method the approach of constructive
methods was used.

For system engineering quite few guidelines for doing quantitative research
exists. However, in the area of software engineering much more has been
done. There are for example concrete guidelines for how to do case studies
in software engineering (Runeson and Höst, 2009). Kitchenham et al. (2002)
propose guidelines for empirical research in software engineering. However, Kitchenham et al. focus on quantitative studies although some recommendation is applicable to qualitative research as well. For example, in exploratory studies it is important to have clearly defined what questions the study wishes to address. This is a point that should be valid also for qualitative studies.

Runeson and Höst (2009) describe a research process of five steps that are used in case study research:

1. Case study design: objectives are defined and the case study is planned.
2. Preparation for data collection: procedures and protocols for data collection are defined.
3. Collecting evidence: execution with data collection on the studied case.
4. Analysis of collected data.
5. Reporting.

These five steps are however similar for all empirical studies when comparing with Wohlin (2000) and Kitchenham (2002).

Lethbridge et al. (2005) categorize data collection methods into the categories, first degree, second degree and third degree. First degree means that the researcher has a direct contact with the respondent and data is collected in real-time. Typical examples of first degree data collection is interviews and brainstorming. In second degree data collection the researcher does not have to communicate directly with the participants. Second degree data analysis can be used when evaluating a tool. Different instrumentation points are recording how the participant uses the tool. Third degree data collection is typically used when the researcher wishes to investigate the artifact, such as the source code or documentation. This requires basically no involvement at all from the participant.

In this thesis the main approach to collect data is thorough different types of case studies. The data has mainly been collected with the first degree category as described by Lethbridge (2005). Below a more detailed discussion about the specific method used to investigate each research question is presented.

2.3.1 Exploratory Case Study (RQ 1)

The case study methodology was chosen because the aim was to explore what people working in the organizations believe to be the most challenging issues within the development of system and software architectures. Semi-structured interviews were used as the tool to collect data, it provided us with the flexibility to change direction based on the answers. Semi-
structured interviews have predetermined questions, but the order can vary based on the interviewer's perception of what seems most appropriate (Robson, 2002). Additional questions can also be constructed during the interview, and it is also possible to remove questions.

Another important advantage of using interviews instead of using a non-guided survey, for example, is the ability to explain a question further if the respondent is unsure about how to interpret the question. In a guided survey the respondent can ask questions, but it poses similar limitations to interviews. It is also possible for the interviewer to ask the respondent to further elaborate the answer when necessary. However, interviews are time consuming compared to surveys.

Another approach that could have been used to answer the first research question is participant observation. Participant observation is part of the ethnographic studies that are popular in social sciences. However, participant observations are more time-consuming than interviews. Ethnographical studies are discussed in more detail by Agar (1996).

An overview of the three different steps used for investigating the first research question is shown in Figure 3.

![Figure 3. Overview of the three-step method.](image)

**Survey with questionnaire**

The survey served several different purposes. The first one was to validate that the issues extracted were actual issues and that they were in line with the opinions of the different respondents. The second motivation for the survey was to investigate to what extent these issues occur at different companies, outside of the ones participating in the initial study. Another purpose served by the survey was to investigate whether a respondent thinks an issue is important but did not state that clearly during the interview. Lastly, the survey was used as an elicitation of issues that seemed more significant and therefore were candidates for further analysis in the next step. Apart from the
three organizations participating in the initial interview study, the survey was performed at three more companies in other industries.

**Causal analysis**

In the next step, causal analysis was used to find the root causes of the identified issues. This method is used to perform retrospective analysis when following up completed projects or milestones (Bjørnson et al., 2009). The method originates from the KJ method (Kawakita, 1975), in which data is collected from participants and structured in a fishbone diagram. With causal analysis, participants can construct causal maps with little guidance from the facilitators. These maps are used to describe dependencies between different issues and investigate what the root causes of these issues are. The issues elicited from the exploratory case study and the survey was used as a starting point for our causal analysis.

**Specific details about the case study design**

Several measures were carried out to increase the validity of the interview study. The guaranteed anonymity and the inability to trace answers back to the person’s role or responsibilities ensured that the respondent spoke freely, without risking that the respondent’s answers would be used against him or her at a later point in time. In cases where it was suspected that our terminology might be unclear, the respondent was provided with a definition. To avoid any misinterpretation the respondent was able to ask questions at any time during the interview.

The causal analysis sessions were done as workshops, one at each of the participating companies. Each workshop lasted for approximately two hours. The workshop participants were senior architects within software and system architecture development. The method consisted of five steps and is fully described in *Paper III*.

**2.3.2 Survey with Structured Interviews (RQ 2)**

To explore the architecture-centric practices that are used to guide evolution, and which ones that are generally omitted, structured interviews were used with open-ended questions to collect data (Robson, 2002).

Structured interviews can be seen as a guided survey that is conducted face to face, or over telephone. The major drawback of this approach is that it is relatively time consuming to perform structured interviews and this usually reduces the sample size due to the effort it takes to conduct such interviews.
However, there are several advantages as well, many of them similar to the case study using semi-formal interviews (Wohlin et al., 2000).

- One-on-one interviews ensure a high response rate.
- They ensure collection of data for all the questions and reduce ambiguities.
- The respondents have the opportunity to ask clarification questions when needed, so the risk of misinterpreting the questions is reduced.
- The interviewers can encourage the respondents to elaborate on their answers.

### Specific details about the design of the structured interviews

The structured interviews were planned and two pilot interviews were conducted. Since no major changes were made after the pilot interviews, they were also included in the dataset. In our case, most of the interviews were done over telephone, primarily due to practical reasons such as geographical distance.

#### 2.3.3 Combining Data from Exploratory Studies (RQ 3)

To answer this question, we combined the data collected and analyzed to answer RQ1. The issues elicited were primarily used. These were compared to the challenges found in a separate study (Cedergren et al., 2010). This approach, combining the data of two different studies is efficient because the data has already been collected.

### Specific details about the design of the comparative study

As presented above, the issues were focused on system and software architecture development. In the study about challenges, managers were interviewed about evaluating performance in product development. Many of the companies in the two studies overlap and the same methodology was used, including semi-structured interviews. The comparison between challenges and issues was structured in the sense that all issues and challenges were put in a table and compared one by one; similar issues and challenges were grouped together. Each grouping was agreed upon by two researchers and later reviewed by a third researcher.

#### 2.3.4 Combining Existing Evaluation Methods (RQ 4)

To answer the fourth research question, an evaluation method that can aid the choice between different integration strategies is proposed. The research method used to develop this evaluation method is similar to the constructive methods approach (Crnkovic, 2010; Kasanen et al., 1993). In constructive
methods, the first step is to identify the problem and make sure it is relevant, as well as to obtain a comprehensive understanding of the topic. In our case this was done with literature reviews and discussions with practitioners. The next step is to innovate, i.e., construct a solution idea. This is the part where the two different evaluation methods were combined and modified. As a last step one should demonstrate that the solution works. This was done at one of the case companies.

Specific information about how the above described research methods are implemented is presented in each of the included articles.
Chapter 3  Related Work

This chapter gives a brief overview of different concepts and results that are related to this thesis. As several of the studies include automotive companies, the related work focuses on automotive systems and software engineering. A majority of the related work is related to software architecture. Software architecture is a key component of this thesis. However, system architecture is equally important, but related work in this area is at best sparse.

Practitioners working with software-intensive systems often use the term system architecture to describe the overall architecture, including both software and electrical components. However, in academic literature the term system architecture is commonly used to describe the structure of an FPGA or the structure of different processors (Xinming et al., 2008), for example. No concrete definitions were found in a literature search.

Systems engineering is popular in the defense industry and a few of the case companies use systems engineering principles in their development. One result of the work in systems engineering is the ISO/IEC 15288 standard for life-cycle processes (ISO/IEC/IEEE, 2008) that we will use to categorize our issues in Section 4.1. On its website, the International Council on Systems Engineering describes systems engineering as:

An engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle (www.incose.org, 2010).

This thesis’ relationship to system engineering is limited to the use of ISO/IEC 15288 to categorize issues and it is thus discussed no further in the related work.

The related work starts with an overview of software architecture (Section 3.1). In Section 3.2, the evolution of software architecture is discussed. Key principles related to software architecture development are described in Section 3.3. Section 3.4 focuses on challenges related to software and system development specific for the automotive industry.
3.1 Software Architecture

The foundation of software architecture builds upon the theories of Dijkstra (1968), although he refers to the architecture as system structure. Parnas (1972) suggests a system design where flexibility is highlighted as a key attribute. This is the first reference to claim that the structure of software does matter. Software architecture later emerged as an important discipline (Perry and Wolf, 1992).

The term software architecture has evolved over the years and despite several initiatives there is no generally accepted definition. The Software Engineering Institute has collected over 100 definitions (www.sei.cmu.edu, 2010) for software architecture from different professionals all over the world. The reason for the existence of many different definitions could be that the type of product or system being built puts explicit demands on the software architecture; hence it is hard to find a general definition that is suitable for all software systems. If no general definition exists there is a great risk that the discussion will be more about what an architecture is and what it is not, instead of focusing on the real issues.

The software architecture describes the structure of the software system, how it is organized and how it relates to other systems. Furthermore, the software architecture includes rules and guidelines for how a system should evolve (IEEE, 2007). The software architecture is an enabler for quality attributes or non-functional requirements. These are further explained below.

Bass et al. (2003) motivate the importance of explicitly designing and maintaining a software architecture with the following steps:

*Communication among stakeholders.* The software architecture represents a common abstraction of the system that, in most cases, is understandable by all stakeholders. This common understanding is key to negotiating and communicating the architecture.

*Early design decisions.* The software architecture constitutes the earliest design decisions of a system. This forms the keystone upon which other requirements will be built.

*Transferable abstraction of a system.* A software architecture can form a small general component that can be used by other similar systems. If designed correctly, the software architecture can be inherited by other systems, reducing both the time to market and increasing the quality of the output.
The architecture determines the quality attributes of any non-trivial system (Clements et al., 2002). Quality attributes, non-functional requirements or architectural significant requirements ensure that the system has the envisioned qualities. Typical quality attributes are reliability, performance, reusability, modifiability, security and usability.

To aid in designing the architecture for specific quality attributes, architectural styles or patterns can be used. Architectural styles were introduced by Shaw and Garlan (1996). For example, a layered pattern can be used to incorporate flexibility and portability in the architecture. An example of layered architecture is shown in Figure 4. Several of the companies included in this research, as well as the AUTOSAR standard described above, use a layered pattern to be able to hide specific hardware characteristics.

A number of different architectural design methods have been proposed. Attribute Driven Design (ADD) (Bass et al., 2003) is an architecture design method developed at the SEI. ADD bases the design process on the quality attribute requirements.

Another architectural design method is the Siemens Four Views (S4V) (Hofmeister et al., 2000; Soni et al., 1995). This method was developed at Siemens Corporate Research and is based on best practices. The goal of the method is to reduce the complexity of the architecture by separating concerns. The four views are conceptual, execution, module, and code.
Architecture Separation of Concerns (ASC) or the ARES System of Concept (Ran, 2000) is a method developed mainly at Nokia and also uses a separation of concerns approach to manage the complexity of the architecture.

None of the abovementioned methods are used at the companies studied. However, using these methods puts an explicit focus on the architecture. One reason these methods are not used could be that they are more focused on software architecture.

In this thesis, software architecture is used as a framework when discussing software-intensive systems. The software architecture is an important part of all studied systems and several of the research questions described in Section 2.2 are focused on the system and software architecture.

3.2 Software Architecture Evolution

The systems involved in this research are all long-lived systems. Evolvability is a key quality attribute to aid in achieving this without degrading the system too early. It describes the ability to evolve software over time to meet the changing needs of its stakeholders. According to Nehaniv and Wernick (2007), evolvability is one of the principal challenges currently facing software engineering. Software evolvability is, according to Rowe et al., defined as:

“an attribute that bears on the ability of a system to accommodate changes in its requirements throughout the system’s lifespan with the least possible cost while maintaining architectural integrity”.

Lehman formulated the laws of software evolution between 1974 and 1996 (Lehman et al., 1997). These laws are based on his observations of the operating system IBM OS/360. Two of the eight laws seem to be of extra importance for the safety critical embedded systems that we are dealing with in this thesis. The second law is about increasing complexity and the need to spend time in maintaining and reducing it. A related law is number seven, which is concerned with degrading quality, and how all evolving systems will suffer declining quality if they are not maintained and adapted. The complexity of the system could cause the quality of the system to decline. Since the studied systems are all long-lived systems with evolvability as an important quality attribute, it is important to manage complexity. Furthermore, the safety critical aspects of these systems have to be characterized by high quality.
Breivold et al. (2008) propose an evolvability model as a framework for the analysis of software evolvability. The aim is to be able to understand and systematically analyze the impact of a change stimulus in advance.

Parnas (1994) claims that designing for change is designing for success. However, he also claims that it is hard to predict the type of change that should be designed for. For long-lived systems, such as the ones discussed in this thesis, it is important to predict what types of change will occur later. It could turn out that the system has been designed for a change that did not happen and the extra effort put in is thus wasted.

The relationship between the use of architecture-centric practices and evolution is investigated as part of the second research question. In general, evolvability is a key concern for all the organizations studied due to the relatively long life-cycles.

3.3 Software Architecture Principles

In this section, some of the key concepts related to software architecture are described. There is a particular focus on principles that are in some way important to the types of systems discussed in this thesis.

3.3.1 Software Product Lines

The software product line concept has been used for many years in the automotive industry, but more recent efforts have tried to formalize the concept, such as Clements and Northrop (2001). They define software product lines as:

“a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”

A product line is typically used by a company when building a low end product and a high end product from the same set of core assets. A company that develops a range of industrial robots, both high-end and low-end, can utilize a product line to share assets. The high-end robot probably has specific functionality that the low-end robot does not. They have a common set of core assets, but each model has some product specific assets to differentiate between the products.
Several of the case companies use product lines. In particular, the automotive industry has used the product line concept for a long time, but on more of a system level.

Software product families is a term that is used interchangeably with software product lines. Van der Linden et al. (2004) suggest a framework for evaluating software product lines. The framework is four dimensional and the different dimensions are general software development concerns. The four dimensions are:

- Business, how to make a profit from your products
- Architecture, the technical means to build the software
- Process, roles, responsibilities and relationships within software development
- Organization, actual mapping of roles and responsibilities to organizational structures.

Each development concern or dimension is evaluated and may use individual evaluation scales. In the model the different evaluation scales affect each other, i.e. an action that is taken to increase a parameter in the architecture dimension might decrease a parameter in the organization dimension.

A similar model based on the same dimensions is proposed by Larsson et al. (2007). This model explains the relationship between business objectives, architecture, organization and process. It highlights how all changes should start with a change in the business objectives. This five-step method is only relevant when a change in the business objectives causes the architecture to change. For example, a new business objective could be to support distributed development. Distributed development might enforce some changes to the architecture, such as moving from a monolithic core to a more component-based architecture. However, this component-based development might entail changes to the development process. An illustrative figure of what changes are possible, and what dimensions affect each other, are shown in Figure 5. The arrows indicate how the direction of the change, i.e. the business objectives, cannot be altered due to a change in the architecture, for example.
3.3.2 Reference Architectures

The concept of reference architectures is related to product lines. A reference architecture is an architecture that is used as a reference to several products. In the automotive industry, one reference architecture can be used for several models through the addition of model-specific characteristics. The major difference from the product line concept is that the product line has a broader view, including not only the architecture. Eklund et al. (2005) present their experience of introducing reference architectures in the automotive domain. Their main conclusion is that more resources are needed to adopt and maintain the reference architecture than the initial design. Furthermore, they emphasize the support from the organization to be able to use a more architecture-centric development approach.

3.3.3 Documenting Architectures

As with all artifacts of a system, documentation is an important part of distributing information. As described by Clements et al. (2008), the architecture documentation communicates the achievement of the quality attributes decided for the system. The 4+1 view suggested by Kruchten (1995) can be used to document a software architecture. This model provides four different views to tailor the information needed for particular stakeholders:

- *Logical view* primarily supports the functional requirements, i.e. what services or functionality the system should provide to its users.

- *Process view* captures the quality attributes or non-functional requirements.
Physical view describes the mapping of the software onto the electrical hardware and how it is distributed between different nodes.

Development view describes how the actual software is organized and packaged.

Scenarios or use cases are used to connect the different views and becomes the +1.

Clements et al. (2008) describe seven rules for sound documentation. These rules mostly constitute what could be considered common sense, but still provide some valuable thoughts. The seven rules are presented in the list below.

- Write documentation from the readers point of view
- Avoid unnecessary repetition
- Avoid ambiguity
- Use a standard organization
- Record Rationale
- Keep documentation current but not too current
- Review documentation for fitness of purpose

Tang et al. (2005) have investigated the design rationale for architectures through a survey. The survey focused on how design rationale is documented and whether practitioners believe it is important to document the design rationale. The main conclusion is that it is important to document design rationale, but this is not done sufficiently as yet. A possible reason could be the lack of tool support for such activities.

3.3.4 Evaluating an Architecture

In the automotive industry, with many products sharing the same architecture, scalability is an important quality. One of the issues found in the exploratory case study confirms this. A possible solution to this issue could be to use the approach suggested by Bahsoon and Emmerich (2008), where real options are used to value scalability. Gustavsson and Axelsson (2009) also
use real options, but to value flexibility in the architecture. One problem that can arise when an architecture and functions are shared between brands is discussed by Sudjianto and Otto (2001).

Another issue found in the exploratory case study relates to the lack of methods for evaluating the business value of different architectural design alternatives. This issue is supported by Bosch (1999). As a possible solution to this issue an iterative, scenario-based evaluation of a software architecture is proposed by Bosch and Molin (1999). Another scenario-based approach is the Architecture Tradeoff Analysis Method (ATAM) (Kazman et al., 2000) where quality attributes are derived from the business objectives and then used to evaluate a software architecture. The Global Analysis method by Hofmeister et al. (1999) also uses scenarios to drive the design of the software architecture forward through different perspectives. The purpose of the global analysis is to identify important factors that affect the architecture and develop strategies to guide the architecture’s design.

Several evaluation techniques have tried to incorporate the Analytical Hierarchy Process, AHP, originally developed by Thomas L. Saaty in 1980 (1980). Zhu et al. (2005) suggest the use of AHP for software architecture evaluation, but the suggested method requires a combination of a more structured evaluation method such as ATAM.

Svahnberg et al. (2003) present a method for increasing the understanding of limitations of different software architectures with respect to quality attributes, using AHP to create initial values. The input is a set of quality attribute requirements in combination with architectural candidates.

Popp et al. (2007) present a method for evaluating automotive E/E systems. It suggests a system level architecture design methodology supported by tools and methods for the quantitative evaluation of key metrics of interest, related to timing, dependability and cost. To generate an optimized E/E system-level design, a tool chain supporting AUTOSAR (www.autosar.org, 2010) is suggested by Rajnak and Kumar (2007). It is possible that these methods can be used to successfully evaluate automotive E/E architectures, but to our knowledge they have not had any significant impact in the automotive industry.

Related to automotive E/E architectures, Kanajan et al. (2006) suggest an approach to how to balance a centralized architecture and a fully distributed architecture, using the concept of platform-based design (Sangiovanni-Vincentelli, 2002) and the Metropolis framework described by Balarin et al. (2002). Different architectures are valued based on four different qualities:
control latency, geometric metrics (number of connectors, wire length), serial data metrics and flexibility.

Many of these methods seem promising and can probably provide partial solutions to the issues found for RQ1, but have to our knowledge not had any success in industrial cases. ATAM is probably one of the methods described above that has received the most attention from practitioners. However, performing an ATAM requires full commitment from the company and is quite time consuming. At the same time, the evaluation requires the architecture to be well documented, which might not always be the case.

3.3.5 Architecture Process

The process of designing an architecture for an automotive system is complex. The architecture should comply with many different stakeholder needs. Not a single process description for developing automotive architectures has been found in the literature. A general process for architecture development is described by Rozanski and Woods (2005), including seven key activities. One of the more important steps in the process is to identify and engage stakeholders. One problem is that many stakeholders do not see the architecture as their core business and easily prioritize other activities.

Another problem is that the requirements for the functionality that the architecture should support are finalized at a later stage. This means that the team responsible for the architecture has to make qualified guesses about what the future requirements might be and what functions should be supported.

When developing software-intensive system, the development is usually constrained by an overall development process for the product. The lack of a clear process for the architecture might therefore limit the opportunities to influence the design of the architecture.

The above described principles are key concerns in finding issues related to the development of software-intensive systems. The section on architecture evaluation is particularly related to the fourth research question. The reason for choosing these particular practices is mainly based on what is used at the case companies.

3.4 Challenges in Automotive Software Engineering

The challenges of automotive software engineering are discussed by Broy et al. (2007). According to them, one of the issues in the automotive industry today is the lack of competence in software engineering. A further challenge
is that current development processes for software are insufficient. Therefore there is a need for new processes that can aid in reducing complexity, and at the same time enable innovation and reduce costs.

Grimm (2003) claims that one prerequisite for an OEM to be successful is to have competences in software development processes and software quality management.

Pretschner et al. (2007) list five areas that are salient to automotive software development. One of them is a focus on the unit cost of electronic components. As vehicle components are mass-produced over approximately seven years, a Euro 1 cost reduction in one component for an automotive manufacturer that produces 500,000 units per year will lead to a substantial overall cost reduction over seven years. This makes engineers focus on reducing the necessary computational power and memory by optimizing the code for that particular processor and not including more memory than the necessary minimum. The major drawbacks with this approach are that it will be hard to add new functionality or change processor without rewriting the software.

Similar challenges to the ones described above have been identified in the case studies. However, these authors discuss the challenges from a software perspective, while in our studies we have taken a broader approach, focusing on the system and software architecture.

3.4.1 Organization

Some of the issues from the case studies relate to the organization. The reason why the organizational aspects are important can be described using Conway’s law (1968), which says: "Any organization which designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure".

The lack of cooperation between different departments will negatively affect productivity, as discussed by Tjosvold (1988). Based on interviews with managers from an engineering consultancy firm, they conclude that inefficient interaction between departments is costly, both for the organization and for their people.

Beyerlein et al. (2003) list ten principles of collaborative organizations. One of the principles is to align authority, information and decision-making. If there is a failure to do so, the decisions that are made will easily be overturned, with few attempts to explain the reason why to those who made the original decision. This is closely related to one of the identified issues stating
that decisions are usually poorly motivated and it is hard to reach consensus and acceptance in a decision.

3.4.2 Management

Management influences the architecture, and often the lack of understanding for software and electronics negatively affects the architecture. This is an issue that is observed by Graaf et al. (2003), who state that systems engineering is mostly driven from a mechanical and electronic point of view and seldom from a software perspective.

The increased complexity could be related to why management seems to have a lack of understanding for software and electronics, causing the systems to be too complex to overview effectively. The extra layer of complexity is the added software and electronics, compared to traditional mechanical products. Sengupta et al. (2008) show that the learning cycle in managers breaks down in complex environments. One reason for this is the time lag between cause and effect. This time lag is sometimes referred to as dynamic complexity and is further elaborated by Roth and Senge (1996).

One of the management-related issues is utilizing enough resources in advanced engineering projects. One reason that a company fails to do so might be that old development projects cannot keep their deadlines and are therefore utilizing resources that were allocated for advanced engineering projects. This issue and a possible solution is discussed by Thomke and Fujimoto (2000).

Even if the above section is written mostly from an automotive perspective, based on the reasoning in Section 1.4, the described challenges are to some extent general, at least for software-intensive systems.

Further related work specific to each finding is presented in each paper.
Chapter 4  Results

This chapter describes the results for each research question. The chapter concludes with how the results of the individual questions relate to each other and a discussion about the validity of the result.

4.1 Key Issues with Success Factors

This section will discuss the results related to the first research question. Papers I, II, III provide answers to RQ1.

RQ1: What are the key issues affecting real-world decisions regarding a vehicle’s electrical and electronic system architecture?

The results related to the first research question are divided into four parts. First, the use of the research method is described. Although, this is in itself a minor contribution, we discuss it first since the other results are extracted using that method. Section 0 presents the identified issues. Section 4.1.2 discusses the extent to which the issues are general to a larger group of software-intensive systems. This is followed by Section 4.1.3, where the elicited success factors are presented. This section concludes with a discussion of the results related to RQ1 (Section 4.1.4).

The three-step method described in Section 2.3.1, with its exploratory approach, gives an overview of the chosen area to investigate. The prioritization of issues through the survey provides a way of recognizing the most significant issues. The causal analysis workshop provides in-depth knowledge about the root causes of the most significant issues.

Based on the experiences of using this three-step method and positive comments from practitioners, there are indications that it is a suitable method for investigating these types of questions.
4.1.1 Key Issues

The elicitation of issues was done as two separate studies, the first one at a car manufacturer (Paper I) and the second one at company that develops heavy trucks and heavy machinery (Paper II). In total, 21 issues were identified. Not all the issues are directly related to the technical aspects of architecture development, but some are related to managerial process aspects.

In Paper III, the issues are grouped based on the process framework for the life cycle of man-made systems ISO/IEC 15288 (ISO/IEC/IEEE, 2008). An overview of the process framework is shown in Figure 6.

Table 1 presents a full list of all identified issues. The table also includes mapping to different processes according to ISO/IEC 15288. The reasoning behind the mapping is that an issue belongs to a certain process area based on where it should be dealt with. For example, an architecture design process is described within the technical processes and Issue 2 expresses the lack of such a process. It should be noted that ISO/IEC 15288 describes the processes at a very high level and it is expected that each organization that intends to use these processes tailor them accordingly.

As for the architecture process, three major activities are suggested:

- Define the architecture
- Analyze and evaluate the architecture
- Document and maintain the architecture

![System Life Cycle Processes](image)

*Figure 6. Overview of ISO 15008 (2008).*
The organizational project-enabling processes are closely related to the architecture of software-intensive systems as they include portfolio management and life-cycle management.

Project processes describe how individual projects should be planned and executed. They include decision management, which concerns several of the identified issues.

The agreement processes provide brief guidelines for how to acquire or supply products. Compared to the architecture process, the acquisition part alone has more steps than the architecture process. This could indicate a lack of focus on architectural activities.

Looking at the four main areas in ISO/IEC 15288, seven out of 21 issues are categorized under technical processes. Twelve issues are mapped to the organizational project enabling processes.

4.1.2 General Software-Intensive System Issues

To investigate to what extent it is possible to say that these issues are valid outside the automotive industry, a survey was conducted with participants from other companies, all of which develop software-intensive systems. As shown in Paper III a Kruskal-Wallis test indicated that for six issues the companies’ answers were significantly different. For each of the issues that did have a significant difference, in all cases there was only one company that differed. However, the company that differed was different for each of the issues and no real conclusions can be drawn from this.

Discussions about the results with practitioners indicate that company culture and maturity have more impact than domain i.e. whether the company is an automotive company or not. Hence, this indicates that the issues could be general for software-intensive systems. It should be noted that the population used (on average 10 respondents per company) can be considered too low to draw any real statistical conclusions. The results should be considered to be initial indications and additional larger studies are necessary to draw any statistical conclusions.
<table>
<thead>
<tr>
<th>#</th>
<th>Issue</th>
<th>Technical</th>
<th>Organizational</th>
<th>Project</th>
<th>Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Architectural issues should be handled more energetically and it should be made clearer who in the organization is responsible for such issues</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>There is a lack of process for architecture development</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>There is no clear process for handling requirements</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>Cooperation between product development and product planning needs to be improved</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>There is a lack of a method or model for evaluating business value when choosing the architecture</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Architecture decisions are often made based on experience and gut feeling</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>History has a major influence on architectural decisions, and is reflected both in the choice of technology and the organization</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Several brands and products share the same architecture but have different orders of priority between, for example, quality and cost</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>There is a lack of a clear long-term architectural strategy</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>There is a lack of understanding of the electrical system and software at the management level</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>11</td>
<td>There is no method or model for measuring and following up quality problems during development</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>12</td>
<td>Complexity in the organization, as well as the product, has increased, leading to a situation where the existing processes are insufficient</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td>The modeling tools used today demand resources and provide little value</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>Decisions are easily made that suit one's own project, team or component even though it leads to a poorer overall solution</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Pre-development projects have a low priority and to increase their priority they are merged into development projects too early</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>16</td>
<td>Processes and methods are less valued than individuals’ knowledge and competence</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>17</td>
<td>Prestige and rivalry complicate cooperation between different departments and business units</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>18</td>
<td>It is unclear how to prioritize between time, cost and quality</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>19</td>
<td>Decisions are usually poorly motivated and it is hard to reach consensus and acceptance about a decision</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>20</td>
<td>There is a lack of a clear strategy for what development should be done in-house and what should be done at external suppliers</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>21</td>
<td>Technical parameters are regarded as less important than cost when selecting components or suppliers</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
4.1.3 Success Factors

To elicit success factors, the first step was to investigate the root causes of the most significant issues.

By further examining these issues in three workshops with practitioners, we found five success factors. These are considered crucial to successful system and software architecture development. The five success factors are the result of RQ1 and are described below.

A more detailed description of the exact causes that could be partially solved by these success factors is included in Paper III. As described in the methods section, four issues were selected for deeper analysis, based primarily on the results of the survey. The issues that were further disseminated were:

- **Issue 2**: There is a lack of process for architecture development;
- **Issue 5**: There is a lack of a method or model for evaluating business value when choosing the architecture;
- **Issue 9**: There is a lack of a clear long-term strategy;
- **Issue 16**: Processes and methods are less valued than individuals’ knowledge and competence.

The success factors are therefore based on the causes elicited from these particular issues. Each success factor and how it relates to other issues that were not part of the elicitation of success factors are presented below.

**Define an architectural strategy**

A few things need to be in place in order to define a strategy for the architecture. One of them is to have the current architecture documented, communicated and disseminated. Furthermore, a set of clear goals for working with the architecture is needed, both a long-term vision for the architecture and short-term goals. These need to be balanced so that the focus is not only on the short-term work.

This success factor is clearly related to Issue 9, lack of strategy; but also to Issue 16, processes are less valued. However, the relationship to Issue 16 is that underlying causes identified through the workshops are related, such as the lack of insight from management regarding architectural strategy.
Define and implement a process for architectural work
To be able to implement a process for architecture development, the need for such a process must be communicated to management. The process needs to encompass both minor changes related to the introduction of new functionality, and major refactoring of the architectural basis.

Issue 2, lack of process; Issue 5, lack of method or model to evaluate; and Issue 16, processes are less valued, are related to this success factor. Issue 2 is merely a negation of the success factor, but for Issue 5 the reasoning is that the requested method or model and its usage should be described in the process for architecture development. As for Issue 16, the reason for not using the processes today is that they are outdated and not adapted to architecture development.

Define and ensure authority for architects
An important activity is to assign responsibilities related to the architecture. The authority of the architect is important to success when sharing an architecture between different models and products. The architect needs to be given the authority to decide what is important in a project-portfolio context. This includes assigning sufficient time to architectural work, as well as ensuring that the resources working with architecture have the proper training and architectural knowledge. Furthermore, the authority should not only be empowered by management, it has to be gained based on trust in the architect.

Issue 2, lack of process, is connected to this success factor because it is expected that a process defines roles and responsibilities. The contributing causes to this success factor from Issue 5, lack of method to evaluate, Issue 9, lack of strategy, and Issue 16, processes are less valued, are the lack of formal training among architects, insufficient stimuli for improving the architecture, insufficient architecture competence and too much local adaptation. When revisiting the complete list of issues, it is clear that Issue 1 is affected if this success factor is achieved. With the same reasoning it is likely that a part of Issue 18, decisions are usually poorly motivated and it is hard to reach consensus and acceptance in a decision, is related to this success factor.

Clarify the business impact of the architecture
The benefit of the architecture should be highlighted. This could include defining a set of methods or models that visualize the business impact of the architecture. To our knowledge a model that visualizes this is non-existent so this is a possible topic for future research. A prerequisite is to know the business model, both short and long term.
Having a model to evaluate the business value of architectural decisions, as stated in Issue 2, is a clear concern for this success factor. Indirectly, three causes from Issue 9 are also related, namely: insufficient management reliance of the importance of the architecture; the architectural strategy and the priorities from the organization are in conflict; and architecture requirements are less important than feature-oriented requirements.

Optimize the project portfolio, based on the architectural strategy

By reducing the power of each individual project, sub-optimizations can be avoided. Furthermore, there is a need to balance short-term and long-term needs with a project portfolio and ensure that internal needs are balanced towards external customer needs. Also, project and strategy goals should be separate, but without being in conflict. Resources should be reallocated from application development to platform and architecture development.

None of the four issues disseminated at workshops have a clear relationship with this success factor. However, many of the causes do mention the lack of optimizing the product portfolio as a clear delimiter in architecture development. Examples of causes that form this success factor are that product development is focused on customer functionality, that there is a lack of understanding in the need to see beyond individual development projects and optimize globally. Other causes were the lack of time to do long-term work instead of always working in customer delivery projects and that the focus is to solve short-term problems.

4.1.4 Discussion Related to RQ 1

With the first research question, the aim was to find key issues that affect the architecture development of software-intensive systems. By starting broadly, instead of making many initial assumptions, the hope was to find the real key issues. There are most certainly other issues that do affect the architecture development of these systems, and some might even be more important than the ones identified. However, the issues identified clearly do negatively affect architecture development.

The outcome of the initial study was 21 issues that all, to different degrees, effect the architecture development of these software-intensive systems. The deeper analysis, with surveys and causal workshops, guided us to form the success factors presented above. These success factors should not be seen as independent of each other. There are clear dependencies between several of them. For example, define an architectural strategy is one success factor and another one is to optimize the project portfolio, based on the architectural strategy. However, if there is no architectural strategy available it is hard to optimize the project portfolio based on it.
Cost-benefit discussion for proposed success factors

As mentioned above several of the success factors have dependencies between each others. It is therefore hard to do a cost benefit in isolation. A problem with implementing the success factors might not be related to a large cost, but could instead be close to impossible due to internal politics in the organization.

To create a strategy for the architecture could be a relatively easy task if it were to be done in isolation. However, the architectural strategy should be based on the long term vision for the company. If such a vision is clear and communicated the actual cost for the architectural strategy is fairly low. The value that the strategy could provide is at least in theory big. By committing to a strategy for the architecture, it is more likely that all projects work in the same direction. This would probably reduce the cost implied to the development of the architecture and at the same time increase the quality of the architecture.

The cost for implementing a process for architecture development could also be fairly low. However, the need for such a process currently is not obvious to management. By getting a more structured workflow the architecture work can be more streamlined and reduce the risk costs related to late deliveries. With a structured workflow the quality of the architecture will probably increase.

To define and ensure authority implies basically no cost at all. However, this would require other areas such as functional development to give up some of their mandate. The risk is that enforcing this from management will create tensions between the architecture team and other parts of development that might lead to reduced productivity.

To clarify the business impact of the architecture could influence the organization to make better and more well motivated decisions. Since we have not seen any methods that work well for industry the cost for developing new working ones could be large. However, to some extent this could be done by a systematic reasoning with for example explaining the effects of certain architectural decisions.

If each development project would optimize their development based on the project portfolio major cost savings could be made. This would most likely mean that more components could be shared between different projects, reducing the overall cost. To be able to do this in practice there might be a need for some reorganization that could imply a cost.
Table 2. Usage of architectural-centric practices.

<table>
<thead>
<tr>
<th>Architectural practices</th>
<th>Standard practice</th>
<th>Ad-hoc practices</th>
<th>Did not use practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>6</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Evaluation</td>
<td>3</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Reconstruction</td>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Explicit design</td>
<td>7</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Architectural requirements</td>
<td>2</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Elicitation of business goals</td>
<td>5</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

4.2 Architecture-Centric Practices to Guide Software Evolution

The second research question addresses the use of architecture-centric practices, and the way in which they are used to guide evolution. This was done as an exploratory study with structured interviews as the means of collecting data. The concrete research question posed was:

*RQ2: Which architecture-centric practices are used in industry to guide evolution systematically?*

The particular architecture-centric practices included in the study were; documenting the architecture; architecture evaluation; reconstruction of the architecture; explicit architecture design; architectural requirements; and elicitation of business goals related to the architecture. The respondents indicated the extent to which each practice was used. A summary of the different architectural practices considered and the extent to which they are used by the participating organizations are presented in Table 2. Respondents from nine different organizations participated. A wider range of companies participated in this study, both with the geographical distribution as well as the type of systems.

A noteworthy point is that although most of the participating organizations explicitly design the architecture, quite a few collect specific architectural requirements as a standard practice. This could indicate that mostly the functional requirements are considered when designing the architecture. The complete results for this research question are described in Paper IV.

4.2.1 Discussion Related to RQ2

In general, architecture-centric practices are not systematically followed. Many practices are used, but mostly the uses of these practices are people-
Table 3. Overview of the six limiting practices.

<table>
<thead>
<tr>
<th>Nr</th>
<th>Limiting practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Short term cost savings triumph long term success</td>
</tr>
<tr>
<td>2</td>
<td>Quality problems are not recognized until after product launch</td>
</tr>
<tr>
<td>3</td>
<td>Estimating business value is challenging and therefore often neglected</td>
</tr>
<tr>
<td>4</td>
<td>Project performance is more important than product portfolio performance</td>
</tr>
<tr>
<td>5</td>
<td>Technology is not part of the performance evaluation and therefore not prioritized</td>
</tr>
<tr>
<td>6</td>
<td>A cancelled project is seen as a failure and is not acceptable</td>
</tr>
</tbody>
</table>

based instead of process-based. The results indicate that the architectural practices that are most commonly used are architectural design and documentation. However, in most cases the architectural design is high-level and constitutes simple box and line sketches. When it comes to documentation, all companies acknowledge that they do document the architecture, however this documentation is not always up to date with the system. A pattern seems to be that an initial documentation is made, but is not updated as the system evolves.

4.3 Limiting Practices in Developing Software-Intensive Systems

The third research question concerns how the views of the management differ from the architects’ views.

**RQ3:** *What are the causes for the mismatch between architects and management in the development of software-intensive systems?*

In this study, data from two different studies were compared. The first study is the same as was used as input for Paper I and Paper II. This data was compared with a study that focused on challenges in performance measurement (Cedergren et al., 2010). This second study has a more managerial focus, instead of the view of the architect.

The comparison resulted in six limiting practices. Each practice has a negative impact on the architecture development and the overall performance of the product development. In Table 3, the six limiting practices are presented.

A recurring theme is that the early phases are not as prioritized as they should be. The reason for this could be that the company’s existing mea-
surement systems focus on the later phases of development. However, in the later phases of development, the solution space is limited.

In several cases, a reason for not prioritizing the early phases is that most senior people are involved in fire-fighting activities, as described by Repenning (2001). This means that key persons are involved in projects that have already left the development phase, but due to quality issues they still become highest priority of the development organization. The complete results related to RQ3 are described in Paper V.

4.3.1 Discussion Related to RQ3

Our observations indicate that there seem to be consistency between the perspectives of the architects and the managers; both believe that the architecture is important and that it should be prioritized. Still, when resources are limited the architecture is among the first things to go. A possible explanation for this is that if you remove all architectural work, the products will still come out, with possible consequences such as; lack of quality, increased product portfolio cost, decreased possibilities to add new functionality, and increased development time for each product. However, if all software developers are removed, no software will be produced, hence these software-intensive products will not work at all, despite the number of resources put into designing an architecture. The architecture is more of an enabler, but is not usually a necessity for adding new functionality and does therefore not provide direct customer value. However, if the architecture is constantly neglected and the focus is solely on new functionality, the technical debt of the system will increase. This will create a technical debt i.e. doing quick fixes that increase the complexity and cause earlier reconstruction of the system (Cunningham, 1992).

The same comparison can be made with performance measurement in product development. Even if you decide to not measure the performance, products will still be delivered.

The objective of the architecture, when developing software-intensive systems, is to support several products over a long time span. When several products utilize the same architecture it is extremely hard to value the benefit of the architecture for each product. However, from a product portfolio perspective, it should be possible to see the benefit of the architecture. Thus, it is typical of the architecture that it includes both the dynamic and behavioral complexity discussed by Roth and Senge (1996). A large dynamic complexity indicates that there is a long time between cause and effect. This is typical
for architectural decisions where the effect of a certain decision it is most likely not revealed until years after the decision was made.

Significant behavioral complexity indicates that there is a high diversity of aspirations, mental models, and values among decision makers. In the process of architecting, this behavioral complexity is characterized by the different requirements from different stakeholders, such as product owner, marketing department, customer, developing organization and maintenance organization.

Architects say they need more resources, but these are usually cut short. According to this result, the managerial perspective and the architectural perspective coincide. However, existing performance measurement systems focus on parameters that neither the architects nor management see as important. This could partly be solved by adapting the existing performance measurement system to focus on the value perspective for the product portfolio and not on time and cost for specific projects.

4.4 Effective Decisions

The fourth and last research question investigates the possibility of creating a model that can aid decision-making when choosing an integration strategy. The formulation of the fourth research question was:

\textit{RQ4: How to make effective decisions when adding new functionality to an existing electrical/electronic system architecture?}

Based on this question, the result is a method that can aid in the decision-making process when integrating new functionality. The proposed method is fully described in \textit{Paper VI}, together with a guiding example. Issues that may be reduced through using this method are firstly the issue stating that architecture decisions are often made based on experience and gut feeling (Issue 6). The method could also provide a positive influence on Issue 19, which states that decisions are usually poorly motivated and it is hard to reach consensus and acceptance in a decision.

The method provides a structured reasoning for how to choose between different integration strategies when adding new functionality to an existing electronic system architecture. An integration strategy is chosen based on what quality attributes are most important to that particular system. To extract these qualities, a lightweight version of the Architecture Tradeoff Analysis Method (ATAM) (Kazman et al., 2000) is proposed. To prioritize these qualities, both against each other and how well they are suited for a particu-
lar integration strategy, a variant of the Analytical Hierarchy Process (AHP) (Saaty, 1980) was used. The method is flexible and scalable, meaning it is possible to choose the number of people involved, as well as the effort put in by each individual. It also provides some support in answering why a certain design alternative is chosen. If the "why" is clearly understood, there is a low risk that the decision is overrun by a new decision.

4.4.1 Discussion Related to RQ4

The proposed method has been tested on a small scale at one of the participating automotive companies. Although the method was specifically developed for the purpose of adding new functionality, the method itself is general and can probably be applied in other settings when making architectural decisions for software-intensive systems. Since the first part of the method is a variant of ATAM, and that method specifically focuses on eliciting the key qualities for the architecture, it is most likely that the method is limited to use within the area of system and software architecture.

The method helps to select a suitable integration strategy. Choosing a particular integration strategy will affect how the system evolves. Some integration strategies might cause an increase in internal software complexity, whereas other strategies might increase dependencies between different ECU's. However, as discussed in Section 4.2, architecture-centric practices are not systematically used to guide evolution. The proposed method can therefore be used to systematically guide system evolution.

The cost of using the method would be about a full working day for the key stakeholders with an about two extra days for the facilitators. The cost might not be the main problem, but instead the problem of getting all stakeholders to commit to a full day and at the same time. The actual economic value of using this method is not easily measured since it depends if the results pose the decision to be different. However, even with the same decision, the value in using the method is more in the creation of a common understanding.

4.5 Relationships between Results

There are two types of results in this thesis. The first type defines the problem itself, i.e. the identified issues and limitations in the development of software-intensive systems. The second type is the proposed success factors and suggested evaluation method.

Looking from a chronological perspective, the proposed method using ATAM and AHP was the first to be developed (Paper VI). Even if the me-
The method was developed before the elicitation of issues, it is expected that the usage of the method can have positive effects on some issues. The method was developed in close collaboration with the same companies that participated in interviews to elicit issues, which suggests that although not explicitly considered, some specific issues were still targeted with the method.

In the results presented in Paper I and Paper II, the 21 issues were used as input for the study about what architecture-centric practices are actually used. The reason for a specific study (Paper IV) about architecture-centric practices was that several issues were concerned with, for example, a lack of long-term strategy, no architectural strategy and a lack of evaluation methods for the architecture. The results indicate that few architectural practices are followed. The usage of the practices could or should be described in the process of architecture development, however this is undocumented in many organizations.

The lack of understanding from management for software and system development was the issue that motivated the comparative study described in Paper V. This study compared the view of architects with the view of managers. The results indicate that a possible cause for the lack of management understanding could be more related to a lack of ability to communicate with each other. A problem that seems to affect the architecture negatively is the constant focus on project performance, as opposed to project portfolio performance.

To gain deeper knowledge of a few selected issues, workshops were held in which root causes were elicited. These root causes were used as input to extract the success factors described in Paper III.

4.5.1 Relationships between Papers

In Paper I-II, 12 and 14 issues out of a total 21 are presented respectively. This means that 5 issues overlap between the papers. The results from Paper I are from a car manufacturer and the results from Paper II include a truck and a heavy machinery manufacturer.

In Paper III, all 21 issues are summarized, and issues that were ranked by practitioners to be the most significant were further analyzed. The main result of Paper III is the five success factors. Even if all issues are presented in Paper III, Paper I-II are still included in the thesis since they provide more information about the particular organization where the study was done. A discussion about the specific actions that can be implemented for each company is also presented.
*Paper IV* investigates how architecture-centric practices are used to guide system evolution. The paper concludes that architecture-centric practices are not systematically followed. This confirms some of our earlier findings, for example that there is a need for a process for architecture development. In such a process it is likely that the usage of architecture-centric practices would be described.

*Paper V* compares the view of managers with the view of architects. The results indicate that a reason for not emphasizing the architecture more is that most measurement systems used by managers focus on the later phases of development, when the architecture is already set.

In *Paper VI* a method that can aid the organization in deciding which architectural approaches are most suitable for a certain integration strategy. The evaluation is made based on quality attributes that are prioritized with AHP.

### 4.6 Validity

In qualitative research, as this thesis is mainly built upon, an important aspect is its validity. The reason is to ensure that both the research methods and the conclusions drawn from the results are valid. In the literature on research methodology, several different categories of validity are discussed. In this thesis, primarily four types of validity have been considered.

- **Construct validity** is about ensuring that the construction of the study actually relates to the problem stated in the research question, and that the chosen sources of information are relevant.
- **Internal and conclusion validity** concern the possibility of ensuring that the actual conclusions drawn are true.
- **External validity** primarily concerns the possibility of generalizing the conclusions to different projects or domains.
- **Reliability** relates to the ability of others to replicate the study and arrive at the same results.

In each paper, where applicable, an in-depth discussion about these four validity aspects is presented. The validity discussion is mainly based on Yin (2002), but is complemented with more detailed guidelines from Wohlin et al. (2000) and Robson (2002). Below is a more general discussion about the validity of the results, with a focus on the possibility to generalize and replicate them.

Even if the 21 issues (*Paper I-II*) originally were extracted from interviews with automotive companies, there are still strong indications that these issues
are applicable to other software-intensive systems. One reason for this is the observed similarities in the development of these systems as described in Section 1.4. Another reason is that part of the issues overlaps the results from the study about architecture-centric practices (Section 4.2 and Paper IV). In this study, several companies are distributed geographically and therefore limit the potential for geographical differences in the results. However, since the different studies had different focuses, it is not possible to claim this for certain, but we still have indications that this is the case.

Some of the issues could be caused by the culture in the country where the study took place. The initial interview study with the three automotive companies took place in Sweden. A well-known phenomenon in Sweden is decision by consensus. In for example, Issue 19, it is an issue not to reach consensus in a decision. This issue could be less important in other cultures.

Another point that relates to the external validity is described in Section 4.1.2, where the outcome of the described survey indicates that the results are common to the companies participating in the survey.

Another aspect that does not formally verify the results, but provides indications that the results are valid, is the discussions with practitioners. During seminars and discussions, practitioners have expressed that the results are indeed representative of their organizations.

The success factors described in Section 4.1.3 (Paper III), were elicited with data from three different organizations, one of them an automotive manufacturer and the other two in process automation and industrial robotics. Based on the indications that the issues are general for the group of participating companies, there are no reasons to believe that the success factors are not general as well, even if their relative importance could differ. However, the actual implementation of each success factor might be different between the companies.

A concern with the comparative study describing limiting practices in the development of software-intensive systems is the use of two different studies with partially different focuses. One risk could be that the results are interpreted incorrectly. However, this risk was limited since the same two researchers participated in both studies, so that both researchers had extensive knowledge about the individual results and underlying data of each study.

The proposed evaluation method that combines ATAM and AHP (Paper VI) has been evaluated on a small scale at one of the participating automotive companies. Even if the feedback from using the method is positive, there is a need to evaluate the method in full scale and at other companies. A risk re-
lated to this is that the method was done before the initial interview studies. This could affect the company participating in the development of the method. However, a limited number of people at the company were part of the development of the method.

A point that should be discussed is to what extent it would be possible to replicate the results of these different studies. It is hard to predict whether the results would be exactly the same. Regarding the different issues, many will probably overlap, but most likely a few might not be found again and at the same time new ones will probably emerge. Since no one claims that all issues related to the development of software-intensive systems have been found, this is not really a problem. Although one reason for potential differences in the results could be that at least two of the participating organizations have reorganized their development efforts after these studies were made. Another point that could affect the ability to replicate the studies is that the organizations might learn from our results. Also related to the ability to replicate, is the question if another researcher using our notes from the interviews still ends up with the same conclusions. Due to the promised confidentiality to the companies, this would not be possible to achieve. However, since at least two researchers participated in the analysis part, it can be assumed that another pair of researchers would reach similar conclusions.
Chapter 5  Conclusions and Future Work

This chapter concludes the thesis. It starts with an overview of the more academic contributions. Since the work has been done in close collaboration with practitioners there is a separate discussion about specific contributions for industry. The chapter is concluded with my own reflections and possible future research directions.

This thesis has investigated key issues related to the development of software-intensive systems. To elicit issues, an exploratory approach with interviews was used.

A characteristic of the studied systems is that they are all long-lived systems with safety criticality as a major concern. This is applicable to products like cars, trucks, airplanes, industrial control systems, and so on.

It can be concluded that many of the identified issues are not technical, i.e. they relate to the organization, processes and management. When this research started it was expected that the focus would be on technical parameters for designing architectures for software-intensive systems. However, on analyzing the results from the different studies, it became clear that more than just technical challenges had been found.

The issues are relevant to the studied companies and are real issues at these companies. It is hard to claim that all the issues related to the development of software-intensive systems have been found, but the results indicate that these are important issues to focus on to remain successful within each company’s domain.

The initial studies only included automotive companies, but were expanded to other domains such as process automation, and industrial robots. The survey including all participating companies indicates that the issues identified at automotive companies seem to be general for other software-intensive companies as well.
Another contribution is the three-step research method that started with an exploratory case study. The data were collected using semi-structured interviews to get a broad understanding of the challenges. The most significant issues were elicited with a survey and, in the last step, causal analysis was used to find root causes for the identified issues. This approach could be used when the problem might not be well defined, and instead there is more of an instinct that certain aspects are problematic. The method both identifies possible problems and provides potential solutions.

As an approach to limit some of the issues, the proposed method can be used for valuing different design alternatives against each other. The method is based on two well-known methods, the Architecture Tradeoff Analysis Method (ATAM), and the Analytical Hierarchy Process (AHP).

5.1 Industrial Footprint

When doing research in an industrial setting it is always hard to determine what causes the organization to change in one direction or another. Therefore the industrial footprint of this research is hard to measure or give an estimate of its impact.

Industry can use the identified issues and analyze in what way they are important within their organizations. Furthermore, the issues can be used as a way to increase awareness and therefore start actively thinking about these issues and what to do about them.

The three-step method is possible to apply in an industrial setting for organizational development. In general, the thesis might increase the understanding of what the challenges are when developing software-intensive systems. Furthermore, it might provide an enhanced understanding from management about what it means to develop a system and software architecture. Most likely, this understanding will be mutual, giving the architects a perspective of the business drivers for the company.

The two most important direct contributions to industry are firstly the method combining ATAM and AHP and secondly the five elicited success factors.

The concrete recommendations that are proposed will in some cases imply an initial cost to the organization. However, the cost factor is relatively small and none of the proposed success factors would require any new tools. Since the cost is relatively small, does this mean that it is easy to the proposed changes? Even if the cost is low the internal politics related to the proposed
solutions are much harder to deal with. The reason that many of the identi-

fied issues even exist in the first place is probably due to internal politics and

lack of understanding. Few of the issues can be solved by buying a tool or

attend a two day course. The mindset in the studied organizations needs to

change and acknowledge that system and software architecture development

is crucial for these companies in order to continue to be successful.

5.2 Limitations and Self-criticism

A Ph.D. project is basically about learning to do research, and there are

many limitations. With hindsight I see things that could have been done bet-

ter, or at least differently. The broad approach in this thesis is both a strength

and a limitation. The strength is that there are issues which most likely

would not have been found if only the technical implications of system and

software architecture development had been considered. However, this could

also be a limitation since it gives a broad understanding that could be consi-

dered too broad for a Ph.D. project.

The thesis has focused on identifying issues or challenges, but relatively few

concrete solutions for how to deal with these issues are proposed. Looking

back, it could be that more effort should have been put into trying to develop

new methods and models that could solve these issues. However, methods

and models by themselves do not solve anything without the right people

there to use them.

The companies mainly involved in this thesis are mostly based in Sweden.

This is a clear limitation and maybe greater effort should have been made to

get companies from a wider range of countries. However, this could create

other problems such as language barriers, different cultures and so on. It

would of course have been interesting to compare the results from this thesis

with a wider range of companies, based on both products and geographical

location. To some extent, this was done in the study comprising Paper IV.

Overall, I believe that the same research methods would have been used if I

were to pursue the same research questions again.

5.3 Future Research Directions

This thesis has mainly focused on identifying and analyzing issues related to

the architecture development of software-intensive systems. Even if some

efforts have been made to solve these issues, there is still plenty of work that

can be done in this area.
Each of the five elicited success factors can serve as a starting point for future research contributions. One of the most important points is to be able to evaluate the performance of the architecture. There is a great need for an architecture performance measurement system in order to motivate increased resources for architecture development. In particular, there are indications that it is important to be able to evaluate the architecture based on the business objectives. If this can be done in a convincing way, more focus will most likely be put on the architecture.

One other thing that becomes clear in this thesis is that many of the issues related to architecture development are non-technical. This is an area where little has been done and where future contributions are important.

Based on discussions with practitioners, it seems that some issues might already be in focus at some of the companies. In a few years it would therefore be interesting to elicit a new set of issues to determine whether the original issues presented in this thesis still exist.

For industry, and in the area of the development of software-intensive systems, one focus should be to understand in what way the architecture affects the overall profitability of the product. There should be extra focus on the product portfolio management aspects.

This thesis is mainly within the area of software engineering, but is also related to the field of systems engineering. These two fields approach each other and are already overlapping. It is in the border between these two research fields that software-intensive systems belong. Much can be done in this area and today most research approaches focus on one of these fields instead of trying to bridge them.
Chapter 6 Bibliography


www.sei.cmu.edu, 2010. SEI.

