Mälardalen University Press Licentiate Theses
No. 152

# TOWARDS A GUIDELINE FOR REFACTORING OF EMBEDDED SYSTEMS

**Sara Dersten**

**2012**


School of Innovation, Design and Engineering

ISBN 978-91-7485-070-3
ISSN 1651-9256
Printed by Mälardalen University, Västerås, Sweden

# Abstract

The electronics in automotive systems give great possibilities. It has contributed to environmental improvements through reduced emissions and reduced fuel consumption, safety, driver assistance, and quality through better diagnostic capabilities.

Automotive systems are today distributed embedded systems that consist of several nodes that communicate with each other. The increasing possibilities have led to a situation where functions that used to be stand-alone, are today dependent on several inter-connected systems which all contribute to the desired functionality. This has increased the costs and the complexity to deal with the systems.

The automotive industry is adopting a new open software architecture, called AUTOSAR, that is intended to reduce the complexity. AUTOSAR also gives possibilities for coping with large product ranges and for component sharing. The introduction of AUTOSAR is an example of an architecture change without modifying the external functionality. We have chosen to call such changes *system refactoring*.

However, if the introduction of AUTOSAR is not successfully performed, there are risks for delayed development projects, which are costly for the automotive companies. Unfortunately, existing engineering standards and literature focus mostly on new product development and less on system refactoring, and this gap needs to be filled. The goal of this research is to provide guidelines for refactoring, which provides support throughout the complete process of system architects in efforts to refactor the system.

This thesis identifies the characteristics of refactoring processes. This is done by empirical studies of the drivers behind refactoring, the effects we can expect from refactoring, and the process activities and characteristics. The result can be used to create guidelines for improving the work of refactoring.

# Acknowledgements

Starting doctoral studies was a type of *life refactoring*. To successfully perform this task, I needed support. There are several persons that have contributed to this support, consciously or unconsciously.

My supervisor, Jakob Axelsson, has supported me during the entire process. He helped me structuring my research work and finding relevant questions and, still, he always let me chose my own ways. My co-supervisor, Joakim Fröberg, helped me get started, both with practicalities and with the study plans. For a short time, I had the privilege to have Rikard Land, as a second co-supervisor, who was a co-author of my first and best written paper. I have at Volvo CE the pleasure to work with Nils-Erik Bånkestad. He has during my research studies, especially in the last part, supported me with interesting ideas for research. The rest of my colleagues, at the E/E system architecture department, have been very understanding when my focus has been on my studies instead of current department issues. Several persons at the research community have been significant for me. It was Christer Norström that convinced me to start the doctoral studies. Before I accepted, I consulted Johan Kraft to find out more. I had many laughs with my former roommates, Rafia Inam and Saad Mubeen. Later I got the opportunity to share room with Stefan Cedergren, Peter Wallin, Håkan Gustavsson, Anton Jansen, Anders Wall, and Stig Larsson, who all gave me more insight into the business aspects of electronics. Daniel Sundmark gave me valuable comments on my thesis proposal. I learned a lot when assisting Gordana Dodig-Crnkovic, Jan Gustafsson and Daniel Flemström in courses. All practical problems were smoothly solved by Monica Wasell and Carola Ryttersson. In fact, all doctoral students and personnel at Mälardalen University are very nice. Unfortunately, there is limited space to mention everyone, since I want this section to fit in one page.

My dear friends, Jasmin and Isabell, have been very understanding when I have been occupied writing papers or travelling to conferences.

My family, has shown great patience and made lots of sacrifices, so I finally could complete this task, especially my nearest and dearest, Christian and Ylva.

Thank you all for supporting me.

# List of Papers

This thesis is based on the following papers.

Paper A — Dersten, S., Fröberg, J., Axelsson, J., Land, R. (2010) Analysis of the Business Effects of Software Architecture Refactoring in an Automotive Development Organization. *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) 2010*, 269–278

Paper B — Dersten, S., Axelsson, J., Fröberg, J. (2011) Effect Analysis of the Introduction of AUTOSAR. A Systematic Literature Review. *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) 2011*, 239–246

Paper C — Dersten, S., Axelsson, J., Fröberg, J. (2012) An empirical study of refactoring decisions in embedded software and systems. *Proceedings of the Conference on Systems Engineering Research (CSER) 2012*, 8: 279–284

Paper D — Dersten, S., Axelsson, J., Fröberg, J. (2012) Characteristics of a System Refactoring Process in Embedded Systems Development. *Submitted to the 7th Workshop on SHAring and Reusing architectural Knowledge (SHARK) 2012*

# Contents

# Abbreviations

| | |
|---|---|
| ABS | Antilock brake system |
| AHP | Analytic hierarchy process |
| ALMA | Architecture level modifiability analysis |
| ATAM | Architecture trade-off analysis method |
| AUTOSAR | Automotive open system architecture |
| AYC | Active yaw control |
| CBAM | Cost benefit analysis method |
| COTS | Commercial off the shelf |
| EBD | Electronic brake force distribution |
| ECU | Electronic control unit |
| EMC | Electromagnetic compatibility |
| ESC | Electronic stability control |
| ESP | Electronic stability program |
| OEM | Original equipment manufacturer |
| ROI | Return on investment |
| SAAM | Software analysis architecture method |
| SW-C | Software component |
| TCS | Traction control system |
| VFB | Virtual functional bus |

# 1. Introduction

Automotive systems are large distributed systems that consist of several inter-connected electronic control units. In automotive systems the complexity has increased to a level where it becomes very hard to adapt to new technologies in order to fulfill new customer, environmental and legal requirements. One reason is that the electronics system constitutes a more important part of the functionality and the business around it. The functions that before were managed by stand-alone systems, are today dependent on several inter-connected systems which all contribute to the desired functionality. Therefore, a system architecture, i.e. a structure for the system and its components, is needed to ensure that desired requirements are met.

Lately there have been several major recalls of vehicles from different automotive producers [1-3]. The increasing complexity of the automotive electronic systems is blamed for those incidents. To deal with the problem manufacturers and automotive suppliers together developed an open standardized architecture for automotive systems. The result is a common software architecture for automotive systems called AUTOSAR. The automotive industry hopes that AUTOSAR will reduce this complexity.

Companies world-wide are now introducing AUTOSAR into their products. This means that the architecture is changed, without any changes in product functionality that is visible to the user. We have chosen to call such changes *system refactoring.*

However, introducing AUTOSAR may not be as easy as the companies think. It will give effects, not only in the electronics systems, but also across the company organization. Production systems have to be adjusted; the development environment needs to be updated; and processes and responsibilities have to be developed. If these factors are not set in time, the development projects that are going to use the new software architecture may be delayed.

A missed deadline is very costly and the automotive companies want to avoid this. Therefore, there is a need for the companies to prepare themselves in time before the introduction of AUTOSAR. Current practices and the processes described in the systems engineering standards are mostly concentrating on new product development and less on system refactoring. We think there is a gap here that needs to be filled. Therefore we aim to provide guidelines for system refactoring to be used in the architecture process. The

goal of the research behind this licentiate thesis is to acquire the required knowledge for constructing these guidelines.

## 1.1 Complexity in automotive systems

The complexity in automotive systems can be demonstrated by the Electronic Stability Control (ESC), also referred to as Electronic Stability Program (ESP). It improves safety by recognizing unstable driving conditions and taking appropriate actions. To prevent over-steering and under-steering, braking is applied to the vehicle wheels. ESC is common in all types of vehicles, including cars, trucks and busses [4, 5]. ESC relies on several other vehicle systems: Antilock Brake System (ABS), a safety system which prevents the wheels from locking up; Electronic Brake force Distribution (EBD), a system that varies the braking force applied on each wheel; Traction Control System (TCS), a system which regulates the power supplied to the wheels; and Active Yaw Control (AYC), a system that uses an active differential to transfer torque to the wheels that have the best grip on the road. Traditionally each of these systems consists of at least one electronic control unit (ECU) which together with connected sensors and actuators handles system functionality. Nowadays, modern systems must be able to cooperate across different domains. These interconnections add dependencies in the system, like temporal dependencies or state dependencies of control units [6].

## 1.2 Automotive development

The development of automotive systems usually uses a product-line approach and component-based development. Introducing these methodologies in a traditional system includes refactoring since the system has to be adjusted to fit a new component model or product platform. For that reason, we have based our research and literature studies on the introduction of these development approaches. We will here give a brief background to them.

### 1.2.1 Component-based development

In component based development software systems are built from existing components. This means that components can be reused and shared between product releases and product variants. The advantages are reductions of time-to-market, development cost and maintenance cost [7, 8]. Since a reused component is already used and tested in different contexts, there might also be a possibility that the component is more reliable than a newly developed component. The components used in component based development

can be developed in-house, bought from an external subsystem developer or as off-the-shelf components (COTS).

### 1.2.2 Component-based development in automotive systems using AUTOSAR

AUTOSAR (AUTomotive Open System Architecture) [9] is a component-based model for automotive systems. It provides a common software infrastructure for automotive systems based on standardized interfaces and components. Key features are modularity, configurability, standardized interfaces and a runtime environment. A layered software platform facilitates the achievement of the technical goals modularity, scalability, transferability and reusability of components. Automotive manufacturers and suppliers hope that AUTOSAR will help managing complexity.

In the AUTOSAR architecture, each ECU incorporates a basic software component which includes infrastructural services such as operating system functionality, vehicle network communication, memory services, diagnostics and ECU state management. The basic software component is built as a layered structure where each layer is abstracted from the lower layers and hence independent of hardware implementations. The application layer is located on top of the basic software. An application is built up by one or several AUTOSAR software components (SW-Cs) that are located on one ECU or distributed on several ECUs. The AUTOSAR SW-C contains parts of the application functionality and is atomic, meaning that it only can be located on one ECU. The AUTOSAR SW-Cs can also be responsible for handling of specific sensors or actuators.

The AUTOSAR SW-Cs are communicating through the Virtual functional bus (VFB), a middleware responsible for mapping of communication messages. Usually the address and source information in the communication messages are specified by the sending application component. In the AUTOSAR methodology the address and source information of all communication messages are configured in system development. During run-time this information is mapped to each message by the VFB. This methodology requires specific development tools to help OEMs (Original Equipment Manufacturers), and suppliers to design and map SW-Cs, ECUs, networks, sensors and actuators.

### 1.2.3 Product-line development

Another example of a typical system refactoring is the introduction of product-lines. The idea with product-lines is to reuse the same basis, a platform, in several members of a product family. The platform methodology is normally structured as layers, as components or as a combination of these. On

top of this platform each specific product adds its own core functionality or features. In this way, one can concentrate on specific properties of each product member instead of inventing the same things over and over again. One example is construction equipment. Both an articulated hauler and a wheel loader need power management and communication between electronic control units. However, they differ a lot in core functionality. The wheel loader needs to have complicated control for lifting its arms when the articulated hauler might have advanced suspension systems.

The reason why product-lines are so beneficial are not only due to re-use of software code [7]. Product-line approaches save time during the requirement phase since almost all requirements can be reused between products and releases. Also many architectural problems are already solved and the system architects can concentrate on core functionality. Other aspects such as project planning might also be easier when less functionality has to be developed in each project. One important factor for a successful utilization of a product-line approach is variability management [10]. However, there might be several issues related to the introduction [11]. The use of product-line architecture requires increased knowledge by the engineers. Other problems are conflicting quality requirements of components in different context or in different products.

## 1.3 System architecture

Refactoring of the system and its infrastructure involves selection of technology solutions and leads to compromises between desired, but conflicting, characteristics; activities which typically involve system architects. Usually, the general system architecture process focuses on the early phases of system development, when structural and conceptual decisions are made for a new product. The process of refactoring includes the same types of decisions, even though there are significant differences. In a typical scenario for refactoring, an architecture that supports multiple products and product generations already exists, when a revision is needed to meet future demands. This architecture must be improved to meet the desired properties. As both a general architecture process and a refactoring need to consider system properties, we have chosen to use the general system architecture process as a starting point for studying refactoring. This section will briefly explain what system architecture is.

### 1.3.1 Definition of system architecture

There are many definitions of system architecture. In this thesis the IEEE definition [12] is used:

> "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."

In embedded systems this relates both to software and hardware. It might be how the software is organized and allocated to the hardware, choice of communication protocol and physical links, but also which development environment to use.

### 1.3.2 System architecture as Lego blocks

System architecture can be exemplified by building a Lego construction. First we have to define the structure of the Lego blocks, or the components. We must decide how the blocks fit together, e.g. the bulges on top the blocks, and their dimensions, e.g. length and width. Second, we have to define the relation between the blocks, or components. We may decide that a yellow Lego block always must be placed on a red block. Third, we must decide the environment to build our construction on. In the Lego case, we might choose a Lego plate where we attach the lower layer of the Lego blocks. Fourth, we have to give some guidelines for the design and development of the construction. It may be to start building the construction from the bottom and up. Figure 1 illustrates system architecture as a construction of Lego blocks.

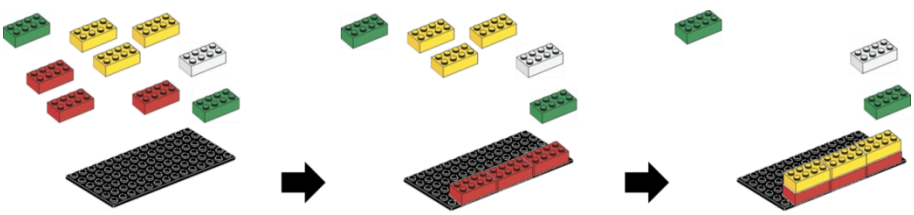

*Figure 1. System architecture illustrated as Lego blocks. The black plate corresponds to a system platform on which system components are connected through well-defined interfaces (bulges).*

In reality, these blocks are software code or hardware components. When we refer to these components, we mean components on different levels. Both in hardware and software components are often composed of other components. Sometimes the component itself is a system, with its own architecture.

These components, or sometimes subsystems, have interfaces which they use to communicate with other components. For a hardware component this interface is often a connector to which an electrical wire is connected. For a software component, the interface might be a shared memory space, a socket, or a procedure call.

The environment that we build the system on might be an operating system, such as Linux, and for hardware the printed circuit board, where our components are mounted. The system cannot be a system by itself without any communication with the outer world. It must be able to input and output data from its environment. In a car this input might be a signal saying that the driver is braking. An output may be a tell-tale saying the car engine is out of oil.

Before designing a system, a procedure or principle for how to construct it must be set. We can choose if any software should be included or not. We can choose if we are going to construct the system from our existing components from earlier developed systems, or if we are going to create everything from the beginning. We must also decide if we follow strict routines. A crucial step, before constructing the system, is of course to decide the aim of the system, if any specific requirements have to be fulfilled, and if certain rules have to be obeyed for the completed system. We must also know about which budget we have.

## 1.4 Thesis outline

This introduction is followed by Section 2 that formulates the research problem with corresponding research questions and explains the research methods used for answering these questions. In Section 3, the results are presented and discussed. Section 4 gives an overview of related literature and research. Section 5 concludes the thesis results and contributions, and proposes future work. The thesis is followed by an appendix with the appended papers.

# 2. Research scope

This section will present the research problem and the stated research questions, followed by the method we used to answer these questions. The four empirical studies we have conducted will also be presented, and these are further described in the appended papers.

## 2.1 Problem formulation

Many companies that develop embedded systems will at some point perform a refactoring of their system architecture. One example is Volvo CE, a provider of construction equipment. To cope with a product range of at least 150 machine models, Volvo CE uses a product-line approach where an electronics platform is shared between the products. This platform includes internal and external system communication, diagnostics, logging, I/O handling, systems handling etc. On top of it, machine specific applications are added. Volvo CE is now facing an updating of the platform. The next architecture is AUTOSAR based and includes technology, methods, and tools for the electronics systems of all products developed by the Volvo group.

The change to the new architecture may affect many aspects of Volvo CE electronics systems, such as aftermarket tools, software structure, communication protocols and development tools. The system architects at Volvo CE have a major work ahead of them, but still know little about how the company and the products will be affected. If the new platform is not successfully introduced, there are risks of delayed development projects.

Several standards for system development exist today. For the architect, the standard ISO/IEC 42010 [12] is of interest. It concerns how architectural descriptions should be expressed to facilitate communication around, and the development of, the architecture. ISO/IEC 15288 [13] describes the life cycle processes associated with human-made systems, and also processes needed for support of the life cycle processes. It is aligned with ISO/IEC 12207 [14] which is more concerned with the development of software systems. IEEE 1220 [15] gives a more detailed description of the life cycle processes than the other two. There are also a number of books in the area of system architecting, such as “The method framework for engineering system architectures” by Firesmith et al. [16], “Software architecture in practice” by by Bass et al. [7] or “System Architecting” by Muller [17].

All these standards and books mostly concentrate on the development of new products or new features, and contain only some smaller elements of processes for system evolution. Despite the extensive literature, there is a lack of descriptions of activities in the system refactoring process. Most literature and research focus on new product development.

This gap between existing development knowledge and the system refactoring process causes problems for the system architects at Volvo CE and at other companies. It is not easy to understand the benefits and costs, and to explain how it will affect the company in terms of reduced costs and hence motivate management about the proposed changes. When the system architect detects the need of refactoring the system, he needs to argue why extra resources are required on system architecture activities. He needs to investigate how the changes affect the system and the organization, and how the organization should be prepared. During this process many decisions must be made under time constrained conditions. The problems that may occur are:

- Poor predictions of effects on development effort and costs, due to system adjustments, education needs, new test environment etc.
- Risk of important stakeholders missed or involved late, from aftermarket, product planning, production etc.
- Risk of unwanted or unplanned technical effects when performing refactoring, e.g. quality problems, and supplier compatibility.
- Risk of unwanted or unplanned organizational effects when performing refactoring, e.g. undefined roles, responsibilities, and processes.
- Lack of organizational support, due to poor communication between system architects and management, and between co-workers.
- Risk of delayed time-to-market, due to poor planning and unexpected effects.

The purpose of our research is to find out how we can help the system architect in the work of refactoring a system and from that create a guideline. The guideline will assist the system architect in preparing and explaining system refactoring to the organization. This thesis describes the initial research where the guideline is outlined by exploring the system refactoring process.

## 2.2 Research questions

Three research questions have been stated to explore the system refactoring process.

### 2.2.1 RQ 1: Which effects can be expected from a system refactoring?

We believe that system refactoring causes effects on both system properties and on the principles for its development and evolution. We also believe that system refactoring causes effects throughout the whole life cycle of the product, and the corresponding processes in the company.

By answering this question we will understand the consequences of a system refactoring, in terms of impacts on system properties, on the company and on their intra- and interrelationships. This is important for decisions relating to the choice of the technical solution, and to planning and preparations of system refactoring changes.

### 2.2.2 RQ2: What are the drivers of system refactoring decisions?

We believe that the drivers behind refactoring of embedded systems are both business-related and technical. We also believe that practicing system architects tend to analyze the technical aspects more than the business aspects.

By answering this question we can guide the system architect when collecting information and performing analyses, that will be used as decision-support by management.

### 2.2.3 RQ3: What would a guideline need to contain to support system refactoring?

We believe that certain activities are more important than others, in the system refactoring process. We also believe that the characteristics of the process differ from the normal system architecting process. We also believe there is a difference in the need of guidance to succeed with an activity.

By answering this question we will understand what activities need to be described in a guideline for the system refactoring process.

## 2.3 Research method

The starting point was to help Volvo CE prepare themselves for a system refactoring, but we saw a knowledge gap of the system refactoring process

and its effects. Therefore, we chose to gather information from companies developing products with similar systems. To answer our research questions, we have chosen to look at the system refactoring process in companies producing distributed embedded systems, especially for automotive systems.

Most of our studies have been in Swedish companies. In total, 15 companies and 44 respondents have been involved in our studies. This is because of availability, and because they are representative of other similar companies in the world. The studied companies have their operations or parts of operations spread around the globe, with activities in Europe, Asia, North America, and South America. However, three companies have their development organization located only in Sweden. We have studied system refactoring from the perspective of electronics systems development as spectators that tried to penetrate into the process from outside to collect data from the visited companies or from other studies.

Our research process can be described as a cycle, oscillating between theory and reality. We had a purpose and knew what to achieve, which raised a first question. To answer this question, we chose relevant methods of data collection and analysis. Using the knowledge we received from reality we were able to start fill the theory knowledge gap and identify new issues that were needed to be answered to fill the gap completely. This research process is illustrated in Figure 2

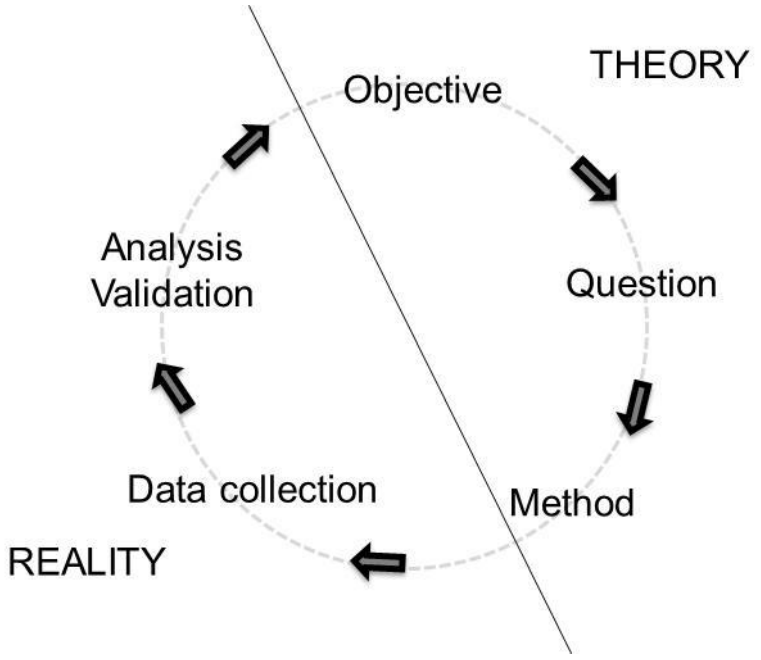


*Figure 2. An illustration of the research process oscillating between theory and reality.*

The research described in this thesis can be described as a first step towards achieving our ultimate goal, which is to create a guideline for the system refactoring process to be used by system architects. Therefore the first part is descriptive, where different phenomena have been explored, with the purpose to describe the reality today. The questions we have been asking are of the exploring type "What?". We have chosen to look at the characteristics that describe the system refactoring process, i.e. what starts the process, what is included in the process and what the outcome of the process is.

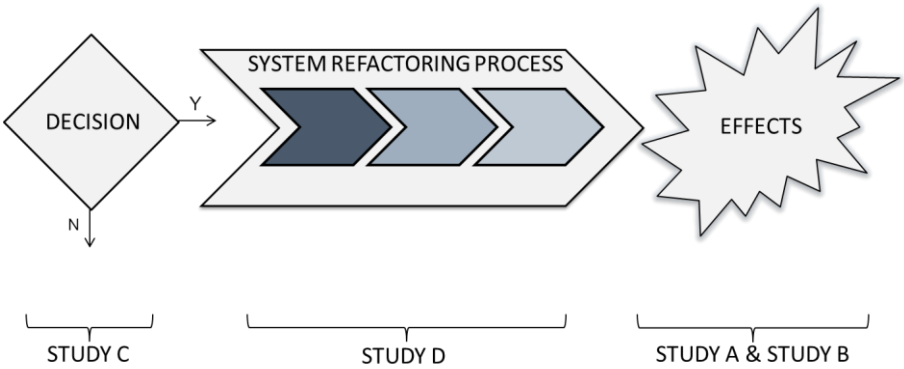


*Figure 3. The relation between the four conducted studies and the system refactoring process.*

The empirical collecting of data has been mainly qualitative. The methods used for collecting data are case studies [18, 19], systematic literature reviews [18, 20], interviews [18], and survey questionnaires [18]. Both qualitative and quantitative analyses have been performed on the collected data. This inductive approach is suitable when we try to create general theories from human experiences and from environments where a lot of complex relationships reign [21].

Four studies were conducted to answer our three stated research questions. The relationships between the studies and the investigated parts of the process are illustrated in Figure 3. Below, we will present the four studies in terms of purpose, method, analysis and validity threats. For one conducted case study, the context is further explained to give background of the company situation. The validity threats [18], we are discussing in this section are against:

*Construct validity* is related to the ability of the results to be generalized to theory and concerns the design of the experiment.

*Internal validity* is related to the fact that the results are a causal effect of the used methodology.

*External validity* is related to the ability to be generalized into practice.

*Conclusion validity* is related to the ability to draw the correct conclusion about relations between the treatment and outcome.

### 2.3.1 Study A. An explorative case study of system refactoring effects

**Purpose**

To answer the first research question an explorative case study was performed in a company that was going to perform a major system refactoring. The study aimed to investigate the introduction phase of the software architecture at Volvo CE, a producer of construction equipment. We wanted to see how the company was affected by changes in different architectural elements in terms of costs and benefits.

**Context**

Volvo CE is part of the Volvo group, which is one of the world's leading suppliers of commercial transport solutions with products like trucks, busses, construction equipment, drive systems for marine and industrial applications, and aircraft engine components. Today the Volvo group has customers all over the world, mainly in Europe, Asia and Northern America.

Volvo 3P is responsible for product planning and global vehicle development for the global truck operations of the entire Volvo group. In order to manage the increasing complexity of the electronics systems in new generation vehicles, Volvo 3P has performed a radical system refactoring on the electrical and electronic architecture and introduced AUTOSAR (see Section 1.3). Volvo 3P hopes that this will reduce the development cost, give more flexibility to meet new technologies and standards, to be able to be first on the market with new features, to meet brand differentiation while maintaining a high degree of commonality, and to achieve multi-site development.

Volvo CE is a producer of construction equipment. Their product range includes 150 different machines, such as wheel loaders, excavators, haulers, and road machinery. The electronics system constitutes an increasingly important part of the functionality in a modern construction machine. In order to meet the demands on business, safety, and development time Volvo CE adapts the development method to a more product-line oriented approach based wholly on an electronics system platform. This includes working on development processes, architecture, tools, and system modeling.

By sharing tools and components, such as engines, within the Volvo group, the companies can take advantage of higher volumes and reduced costs. Therefore, Volvo CE will also adapt to the new Volvo 3P architecture. The architecture consists of a common software platform which includes

communication, diagnostics, logging, mode management and power state management.

### Method

This study was based on interviews with 11 persons at different positions in the electronics development department of Volvo CE. The new architecture was also investigated through reading specifications. The interview questions had a life cycle perspective and were related to effects on system properties and the company when introducing the new system architecture. The interviews were performed in a semi-structured way. Pre-defined questions were constructed but also followed by deeper questions related to the given answers. To ensure that all matters were covered the interviews ended by giving the respondents the opportunity to share additional information. The interviews were audio recorded and notes were taken. Each interview was summarized in text and sent to the respondents for approval before analysis.

The method was chosen because of its explorative character. Interviews gave the possibility to get the answers that were not expected when planning the study.

### Analysis

After data extraction, the identified effects were mapped into a matrix. The matrix rows corresponded to the architecture element that caused the effect. The columns were divided into two parts, capturing affected system properties and affected company functions, respectively.

In this way we were able to identify what in the company or in the electronics system that was affected by a change in a certain architecture element, and also how it was affected.

### Validity

When analyzing the extracted data from the answers we constructed an analysis matrix that helped us ensure that all relevant effects had been considered, which strengthened the construct validity of the study. The study results were based on expectations prior to performing the refactoring and therefore the internal validity was not evaluated. The study was conducted at a specific company and therefore the situation at another company could differ and the results may not be directly applicable at a different company or on a different architecture.

### Presentation

The study was presented at the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) in Lille 2010.

### 2.3.2 Study B. A systemic literature review of AUTOSAR effects

**Purpose**

Since the first study only investigated effects from system refactoring in one company an additional study was conducted to help answer the first research question. The AUTOSAR architecture will be adopted by almost the entire automotive industry in Europe and Asia. Therefore AUTOSAR gave an opportunity to study the implementation of the same architecture in several products and hence an opportunity to compare reported effects of system refactoring with each other. Hence a systematic literature review of implementations of AUTOSAR gave the possibility to summarize these reported effects. The results also strengthened the internal validity of Study A.

**Method**

A systematic literature search was made for papers describing experiences from introducing AUTOSAR.

**Analysis**

The analysis was performed in a similar way as in Study A. An analysis matrix was constructed where each identified reported effect was mapped to the elements in AUTOSAR that caused the effect, and to the functions in the company that were affected and properties of the system.

**Validity**

Since the architecture is introduced stepwise in products and so far only to some extent, only a small sample was found, and which threatened the internal validity. To deal with construct validity a review protocol was developed, where background, objectives, research questions, strategy, sources, and search criteria were pre-defined, according to the advices of Kitchenham [20]. During the process all the found literature and the exclusion criteria have been documented. As the implementation of a specific architecture was studied the results might not be directly applicable for implementations of other architecture. Still, this is an automotive standard and therefore there might be a possibility to generalize the results to other implementations of the same architecture in other automotive companies not covered by this study. There are initiatives in other industrial domains, such as avionics [22], that share similar features, and thus a possibility exists for some results to be applicable in those domains.

**Presentation**

The result of the study was presented at the 37th EUROMICRO Conference on Software Engineering and Advanced Applications in Oulu 2011.

### 2.3.3 Study C. Scenario-based interviews of system refactoring drivers

**Purpose**

The aim of the third study was to find the drivers behind a decision of system refactoring and to answer our second research question.

**Method**

14 interviews were conducted at eight companies that produce distributed embedded systems. The respondents were persons used to make decisions about the system architecture and amongst them were seven system architects and seven managers at different levels in electronics development. The companies and respondents were chosen from their availability and willingness to participate.

All interviews began by giving a start scenario to the respondent. The start scenario represented a suggestion of a change to be made in the embedded systems in the companies' products. The respondent was then asked to request the information he needed to complete the decision of whether the system change should be performed or not. After the respondent answered, additional pre-defined information related to the requested information was given. The respondent was once again asked to request the information he needed to complete the decision. This procedure was repeated until the respondent answered that he was able to complete the decision or at least make a recommendation.

If the respondent asked for information that was not pre-defined and never requested in the previous studies, new information was created on site and stored in the list of pre-defined information. In that way we were able to catch cases we did not expect beforehand.

**Analysis**

The requested information from each interview was categorized into information areas. We investigated the most important decision criteria by performing frequency analyses, where the information areas that most respondents had requested were found. To further elaborate the important criteria, the interviews were re-written in a formal way where the requested information was replaced by the corresponding information area, see Figure 4. Then we compared if the first requested information areas corresponded to the most wanted information.

Differences in answers between system architects and managers, between an industry sector and the rest of the group, and between one company and the rest of the group, were analyzed by Chi-2 calculations.

To capture to what degree the effects on the company is investigated before a decision of system refactoring is made, the requested information was

mapped to one or several company functions. It gave a possibility to see for which parts of an organization the effects were investigated before making the decision.

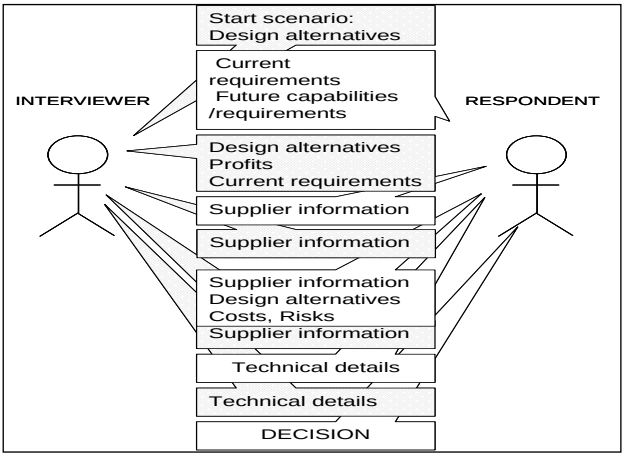


*Figure 4. An illustration of a typical interview described in a formal way.*

**Validity**

The reason why this study was conducted through interviews and not through questionnaires was to cover the reasoning behind the answers. The planning phase of the study included literature studies on architecture evaluation methods, and return-on-investment analyses. To strengthen the external validity the respondents were chosen from companies in different domains. A threat against internal validity was the selection of scenarios. It is hard to select general scenarios which can be applied in all types of domains. The interpretation of the scenarios might differ between different domains.

**Presentation**

The study result was presented at the 10th Annual Conference on Systems Engineering Research (CSER) in St Louis 2012.

### 2.3.4 Study D. A survey of system refactoring activities

**Purpose**

The fourth study aimed to answer the third research question. By identifying which activities are included in the system refactoring process, we could start to sketch out a proposal for a guideline to be used by system architects.

**Method**

Data was collected by using a web-based questionnaire. 34 respondents from 14 Swedish companies that develop products with embedded systems answered the questionnaire. The responsibilities of the respondents were within system architecting, system design, system development, project management, systems engineering, and management.

The survey questionnaire contained a first part that described what a typical system refactoring is. Then the respondents were asked, for each of 35 activities, to rank how important the activity is for the system refactoring process and how helpful a guideline would be to succeed with the activity. The 35 activities were identified through literature studies of conference papers, books, and systems engineering standards, and through findings from earlier conducted studies. The activities were reviewed in several cycles by four persons experienced within system engineering research and practice. Before releasing the questionnaire, a pilot study was conducted.

**Analysis**

In the analysis the activities were grouped into the categories “Most important for the process”, “Important for the process” and “Not important for the process” depending on the lower limits of the confidence intervals of the responses for each activity. The Wilcoxon rank sum test provided the calculations. In the same way the activities were also investigated to see how much support the respondent wanted for each activity. The analysis further investigated differences in responses amongst the respondents with different responsibilities. Through the literature studies we also identified characterizing factors for the general architecture processes. The activities that had been chosen as most important or only important were mapped against it to see if they contributed to each of these characterizing factors. The mapping was then used to identify the characteristics of a system refactoring process.

**Validity**

Possible validity threats against the outcome of this study were the uneven distribution of respondents on the participating companies and the choice of activities that were going to be rated by the respondents. To ensure that the results would reflect the real world as much as possible, we wanted as many respondents that we could find in Sweden. However, this resulted in that the

amount of respondents from each company varied and that might be a threat against the conclusion validity and the reliability of the measures. To avoid threats against construct validity, we tried to identify the activities from several sources, both from academic papers and systems engineering standards as well as from earlier experiences and studies. Also, pilot studies with system architects were conducted, which also strengthened the internal validity.

**Presentation**

The study result has been submitted to the Seventh Workshop on SHAring and Reusing architectural Knowledge (SHARK 2012) that will be held in Helsinki, Finland, in August, 2012.

# 3. Results

As described in the previous chapter, four studies were conducted to answer our three stated research questions. In this chapter the results and answers are presented for each question.

## 3.1 Which effects can be expected from a system refactoring?

The appended papers A and B describe the studies that were conducted to answer our first research question. The results from study A are based on interviews about expected effects, conducted at the electronics development department at a construction equipment producing company, which was just about to perform a system refactoring. Study B is based on a systematic literature review of reported observed effects from the introduction of AUTOSAR, an automotive open standardized software architecture. The effects were reported mostly from industry, but also from academic research.

Table 1 gives an overview of possible outcome from features typically included in architectural changes. These features were included in the studied architectural changes in study A and study B. The overview shows possible effects that we found in these studies and gives examples of in which types of architectures the features can be found. The overview also suggests further areas for investigations. These suggestions are also based on findings of effects from study A and study B. The overview can be used for finding relevant areas to investigate when preparing for refactoring of systems. Given a specific refactoring, e.g. the introduction of AUTOSAR, all the columns must be considered that are relevant, i.e. what has to be examined will be a combination of several columns.

*Table 1. Possible outcomes from features included when refactoring architecture, given with suggestions of further investigations.*

| ***Feature*** | ***Simulation possibilities*** | ***Reuse possibilities*** |
|---|---|---|
| ***Examples*** | *MBD, Hardware abstraction* | *Standardization, CBD, AUTOSAR* |
| ***Possible outcomes*** | Requires tools<br><br>**Development**<br>• Possibility of simulations<br>• Possibility to generate code<br>**Verification**<br>• Usage of test beds without expensive test equipment Cost reduction<br>• More effective tests<br>**Supplier cooperation**<br>• Definition of new processes, roles, responsibilities and corresponding activities<br>**System properties**<br>• *System quality*: Improved quality assurance | **Development**<br>• Time and costs saving<br>**System properties**<br>• *Complexity*: Easier managed<br>• *Maintenance*: Easier |
| ***Investigate*** | | Configuration management processes |
| ***Feature*** | ***Tools, development environment*** | ***Standardized architecture*** |
| ***Examples*** | *CBD, AUTOSAR, MBD* | *AUTOSAR* |
| ***Possible outcomes*** | **Development**<br>• Possibilities for configuration management<br>• More efficiency, lower lead times<br>• Possibilities for system simulations<br>• Possibilities for code generation<br>**Verification**<br>• Adjustments of test systems<br>**Logistics**<br>• Adjustments of logistics systems<br>**System properties**<br>• *Reliability, Safety, Integrity*: Less human errors due to interconnected tools | **Development**<br>• Outsourcing strategies<br>• Commonality sharing<br>• Focus on vehicle features instead of system technologies<br>• Process changes<br>**Supplier cooperation**<br>• Improved specifications for suppliers<br>• Suppliers can develop products/ systems for more than one manufacturer<br>**System properties**<br>• *Safety, Reliability*: Faults found easier |
| ***Investigate*** | System adjustments<br>License costs<br>Education | Business opportunities, processes, responsibilities with suppliers<br>Performance and Safety risks |

| *Feature* | *Hardware abstraction* | *Well-defined and clear interfaces and specifications* |
|---|---|---|
| ***Examples*** | *MBD, AUTOSAR* | *Standardization, CBD* |
| ***Possible outcomes*** | **Development**<br>• Development of hardware and software in parallel<br>**Verification**<br>• Integration and verification faster<br>**Supplier cooperation**<br>• New working models for supplier cooperation<br>**System properties**<br>• *Reusability*: Possibilities to reuse software components between products<br>• Possibilities to reuse hardware components between products<br>• *Flexibility*: Easier to move the software components between different control units, Easier to add new software functionality without the need of integrating an additional control unit into the system<br>• *Safety*: Lower the risk for integration problems | **Development**<br>• Less time is needed on calibration and validation<br>• **Supplier cooperation:** Improved specifications given to suppliers<br>• Improved quality of components delivered by suppliers<br>**System properties**<br>• *Complexity*: More controllable<br>• *Flexibility*: Improved<br>• *Reliability*: Improved reliability, less human erroneous interference<br>• *Safety*: Lower risk for integration problems, faults found earlier |
| ***Investigate*** | | |

### 3.1.1 Effects on company

Both studies imply that a large system refactoring has company impacts. It is important to consider these impacts to avoid delays in product development projects.

**System development**

When the changes in the architecture include new development tools or whole tool chains the development organization should prepare itself in time for several things. Resources must be spent on adjusting current existing tools to the new tool chain. A new tool chain will also affect the way of working and new processes and responsibilities must be set. This also means that the staff needs education on how to work with the new tool and how to adapt it into existing development systems. If this is done properly the benefits, which come at a later stage, can include a more efficient development process with shorter lead times, and improved possibilities for configuration management. A remaining cost will probably be expensive licenses. Typical architectural changes that include new development tools are introduction of model-based development or changes of component models. Model-based development also gives possibilities for code generation and component based development also probably needs less time on calibration and validation in the long run. Another way of shortening lead times in the development process is to develop hardware and software independently of each other, since they can be developed in parallel and hence make both integration and verification faster. Then it is important to use clear interfaces and standards. Our results also imply that standardization makes it easier to change software design at later stages in the process and that manufacturers can focus on product features instead of on system technologies. On the negative side this also requires process changes. Our studies imply that architectures that allow reusability, as product platforms, or component-based systems, save time and costs in system development.

**System verification**

In the verification phase, model-based design, component-based design, hardware abstraction or the introductions of new standards call for new tools. It is then beneficial to choose tools that allow verification by model simulations in virtual environments. These environments can simulate bus loads and simplify verification of subsystems. In this way the verification can be performed already in early development phases and without expensive test beds. Also the use of clear interfaces and specifications means that faults are found earlier. Finding faults early is also important to keep the maintenance effort low.

**System maintenance**

By using model-based design, standardized architectures like AUTOSAR, or components that have been proven-in-use, maintenance seems to be made easier and faults are found earlier. Standardized architectures, like AUTOSAR, seem to give a possibility to maintain a large product range.

**Supplier cooperation**

Using a standardized architecture, like AUTOSAR, or components with well-defined interfaces means that the specifications given to suppliers for purchase are improved. Then there is less space for misinterpretations which in turn gives higher system and software quality. Standardization gives possibilities for reuse and hence opens doors for new business opportunities. The manufacturers can outsource or buy the development of components and subsystems. The suppliers can offer the same features to several customers instead of customized features to specific manufacturers. The manufacturers can then concentrate more on product specific features instead of system details. The initial costs relate to defining new business strategies, such as product portfolios, responsibilities, new tools etc. Changes to a new standard also lock out suppliers that have not adapted to that standard. For instance, a change to AUTOSAR, for a construction equipment manufacturer, may not be beneficial since most suppliers are not within the automotive domain. This also applies to minor changes, such as replacing the protocol used for external communication. Therefore the manufacturer has to investigate effects of this kind of changes, not only on the system level but also on the company level. Other changes that might affect the cooperation between manufacturers and suppliers are if the supplier needs to install or adapt the development environment to fit in the system architecture of the manufacturer. It might lead to a situation where the supplier has to invest in expensive licenses or refuses to take responsibilities for his delivered components or subsystems in the final product.

**Other effects on company**

We have also seen that parts of the company that are not directly connected to the development organization can be affected by system refactoring. Changing the external communication, such as replacing the diagnostics protocol or download mechanism, can affect the aftermarket or production, where new tools have to be developed. Also new diagnostics or logging capabilities might give the product planning and sales new opportunities.

### 3.1.2 Effects on the system

System properties can be divided into two categories. The first category is the properties that are related to the ability of the system to perform its tasks, such as performance and reliability. These are often inter-connected and sometimes effects on one property gives effects on another. For instance, better security against intruders that can alter code or inter-connected systems that lessen human interference improve the system integrity. Then, fewer faults can be implemented, intentionally or accidentally, which also strengthens the reliability and the safety of the system. Other changes in the system that give positive effects on reliability and safety are using standardized architectures or components with clear interfaces, since less error-prone interpretation is needed. Also the overall opinion is that the use of components that have been proven-in-use improves the system safety. Our studies of effects include the introduction of software platforms that give possibilities for the technology to meet future demands. These platforms usually offer a development environment that allows network management so that the system communication can be optimized which seems to improve the system performance. The negative effects are that these software platforms require a lot of memory and run-time themselves, which lowers the overall system performance.

The second category of system properties is the properties of the system that relate to how easy or manageable it is to work with. One such property is system flexibility. When investing in architectures, companies want the selected architecture to be capable of meeting future requirements on new features or legislation. These can be supported by architectures that isolate hardware and software from each other so that software components can be reused or added on several hardware nodes in the system. Other ways of enhancing the flexibility to add new functionality is to use well-defined interfaces between components or to use signal-based communication. Then, only communication databases have to be updated when adding a new software component instead of specifying the addresses for source and destination in all affected components. Systems tend to become more complex over time but standardized, well-defined specifications and processes make them more controllable. This can be achieved by reusing well-known components in several development projects. By using software platforms where several software components are integrated, there is a possibility to integrate more software functionality in each hardware platform. This lessens the system communication and hence dependencies between nodes in the system. Reusability of components also seems to be crucial for maintaining large product ranges due to the well-known interfaces, specifications and documentation.

## 3.2 What are the drivers of system refactoring decisions?

In the appended paper that explains study C, the results of the study that answers our second research question are presented. We wanted to understand the drivers behind system refactoring by looking at what kind of information decision-makers investigated prior to a decision. The areas of information that most of the respondents asked for were regarded as most important for the decision. In this section we will present these information areas in order of importance.

### 3.2.1 Costs

Not surprisingly, information on costs seems to be the most important driver behind system refactoring. These costs relate to development, manufacturing, and maintenance, and they were requested by both managers and system architects, even though managers start to investigate costs earlier than system architects. The developments costs are related to software and hardware, licenses and support of tools. Manufacturing costs relate to hardware and components costs and the production. Maintenance costs are related to product management, i.e. the maintenance of the system components and the warranty costs.

### 3.2.2 Profits

Profits can be gained from both the technical advances and possible market opportunities given by the system changes. The profits gained from technical advances are increased quality, more system flexibility, modularity and better diagnostics. This in turn lowers the costs for development, shortens the development lead times, gives cheaper hardware, reduces risks in projects and simplifies the product management and the ability to add new functionality into the system. The profits from market opportunities relate to the ability of the system to offer new services to customers and the customers' experience of a better quality, which can give an increased sale. The decision-makers compare the profits against the costs of introducing the system changes and perform estimations on when return-on-investment (ROI) can be expected.

### 3.2.3 Supplier information

If the system change includes the possibilities of buying components from suppliers the decision-maker requests information about the supplier and the components. He wants to know the viability of the supplier, such as financial status and survival probability; can the supplier make long-term commit-

ments and what happens in case of bankruptcy? How is the market, are there other suppliers? The decision-maker also wants to know if the component is newly developed and the supplier's experiences of the technology. The supportability of the supplier is investigated and how the supplier handles change issues. What will the cooperation look like, the ownerships and responsibilities? If a components is going to be purchased it must be cost effective, so the business model and license agreements are considered. The component itself must fulfill desired requirements. Much of this investigation involves the purchasing department.

### 3.2.4 Technical details

Both system architects and mangers want to know technical details. They want to know how the proposed system changes will fit the current architecture and which system changes are required to make a solution effective. Typical questions concern interfaces, electromagnetic compatibility (EMC), compatibility with internal proprietary and standardized communication protocols, hardware needs, tool needs, and if changes can be made step wise. Other issues relate to system quality attributes, such as performance and complexity. It is also important to consider the product life cycle stages, such as supportability and aftermarket, and the product strategies, such as the "look-and-feel" of the product.

### 3.2.5 Future requirements

When investing in changes to the system or system architecture the decision-makers must be sure that the future system requirements are considered. Therefore coming functionality, legislation, technologies, and future standards, are identified and investigated, often several years ahead the introduction of them. The decision-maker must be sure that the system will be able to meet these future requirements and investigate the ability of the system to evolve and accommodate new technologies. For example, what happens at the end-of-life for a hardware component? Can it be replaced by a similar component? Does the system provide possibilities for incorporating new services? Can it be reused in future products?

### 3.2.6 Current requirements

Also short-terms goals have to be considered. Existing problems in the current system have to be identified. The decision-maker must understand customer benefits and requested functionality. If there is an infrastructure that is expected to be used across the entire product-line, or across different companies or business units, the decision-maker must consider that the system meets the requested functionality of all of these.

### 3.2.7 Enterprise constraints

Constraints of the enterprise concern strategic goals, including roadmaps, policies, and core business. Financing and available resources also affect the decision, as well as planned development projects and the maturity of the organization.

### 3.2.8 External constraints

There are also constraints from the outside world, such as current laws, regulations, specifications, standards, and guidelines. They may concern safety, communication protocols or vehicle emissions and affect both the system needs and the market deadlines. Existing technologies and competitor products are other drivers for system changes. It is also important that the system is able to meet safety regulations and other legislation on different markets.

### 3.2.9 Design alternatives

In a refactoring decision several solution options have to be identified and considered, such as the possibilities to reuse old subsystems and components, and if there are possibilities to buy solutions from suppliers. Economic analyses of the different design alternatives affect the decision. Supplied products are further investigated according to Section 3.2.3.

### 3.2.10 Risks

Less than half of the participants asked for information about risks during the scenario-based interviews. The requested risk information concerned commercial risks, such as the risk of delays for time-to-market, and technical risks, such as increased system complexity, technical faults, and if new tools were hard to work with.

### 3.2.11 Human requirements

Only a few participants requested information about the human requirements, such as process changes, responsibilities and the need for education. Some participants emphasized the importance of having support from the organization, both from upper-management and colleagues, prior to the decision.

### 3.2.12 Technical management

Only one participant wanted information about the technical management and configuration of the system solution. He mentioned the importance of

having ways of working that ensured that a chosen system solution did not evolve differently between different company departments.

## 3.3 What would a guideline need to contain to support system refactoring?

The last study aimed to answer our third research question and is presented in the appended paper D. The result from the study is based on a survey questionnaire which was given to senior system engineers, mainly system architects, employed at Swedish companies that develop embedded systems.

### 3.3.1 Activities in system refactoring

From a set of 35 activities, which were identified as typical activities in general system architecture processes, 20 activities were identified as important for system refactoring. The ratings were made on a 0-6 Likert scale and activities with a lower bound above 4.0 for the median confidence intervals were considered as "most important". Table 2 shows all the activities and their calculated confidence intervals. The activities that were identified as "most important" are:

- Establish the technical requirements for the system e.g. identify interfaces and design constraints
- Investigate if the existing architecture can be expanded or adjusted to fit new requirements
- Assess whether the identified architecture-alternatives meet the requirements
- Evaluate the effects of the identified architecture-alternatives on the system's non-functional properties, e.g. response times, safety, security, etc.
- Assess the abilities of the identified architecture-alternatives to be evolved, reused, and expanded
- Assess the impact on system life cycle quality factors, such as producibility, verifiability, ease of distribution, usability, supportability, etc. and changes in the corresponding processes
- Select architecture parts to be verified and the verification methods to be used
- Select architecture parts to be validated and the validation methods to be used

Activities with confidence interval lower levels above 3.5 were considered as “important”, and they are:

- Find future product range and customer demand
- Analyze deficiencies in the current system
- Assess ability of the current system to scale for future drivers, e.g. coming laws
- Learn about technologies, system architectures and architectural practices
- Define the characteristics required for the product to be cost effective over competitors
- Define the requirements for different steps in the systems life cycle, such as development, verification, maintenance, etc.
- Establish a requirement baseline of the system architecture
- Develop and identify alternative architecture solutions and selection criteria
- Investigate suppliers (internal or external), in terms of risks, licenses, costs, supportability, responsibilities, viability
- Identify and assess commercial and technical risks
- Update and review the architectural description
- Apply configuration management on the architecture description

*Table 2. The found activities and their confidence intervals of the ratings, given by respondents, for importance and need of guidance.*

| ***Activity*** | ***Confidence levels*** | |
|---|---|---|
| | ***Im-portance*** | ***Guide-line need*** |
| ***Planning*** | | |
| 1. Create an overall technical vision for the embedded system | 3.0-4.0 | 5.0-6.0 |
| 2. Find synergies within different types of products | 3.0-4.0 | 4.0-5.0 |
| 3. Find out company vision, roadmap, core business and policies | 2.5-3.5 | 3.5-4.5 |
| 4. Find future product range and customer demand | 3.5-4.5 | 5.0-5.5 |
| 5. Find future laws and regulations | 3.0-4.0 | 3.5-5.0 |
| 6. Analyze deficiencies in the current system | 3.5-4.5 | 4.5-5.5 |
| 7. Monitor trends in key properties of the system | 3.0-4.5 | 4.0-5.0 |
| 8. Assess ability of current system to scale for future drivers, e.g. coming laws | 3.5-4.5 | 4.5-5.5 |
| 9. Learn about technologies, system architectures, and architectural practices | 3.5-4.5 | 4.0-5.0 |
| 10. Investigate coming technologies | 3.0-4.0 | 4.5-5.0 |
| ***Requirements*** | | |
| 11. Define the characteristics required for the product to be cost effective over competitors | 3.5-4.5 | 4.5-5.0 |
| 12. Define the requirements for different steps in the systems life cycle, such as development, verification, maintenance, etc. | 3.5-4.5 | 4.0-5.0 |
| 13. Establish the technical requirements for the system, e.g. identify interfaces and design constraints | 4.0-5.0 | 5.0-5.5 |
| 14. Investigate enterprise constraints, such as available resources, competencies | 2.0-3.5 | 3.0-4.0 |
| 15. Establish a requirement baseline of the system architecture | 3.5-4.5 | 4.5-5.5 |
| ***Technical solution*** | | |
| 16. Develop and identify alternative architecture solutions and selection criteria | 3.5-4.5 | 4.0-5.0 |
| 17. Identify make-or-buy alternatives | 3.0-4.0 | 3.5-5.0 |
| 18. Investigate if the existing architecture can be expanded or adjusted to fit new requirements | 4.0-5.0 | 4.5-5.5 |
| 19. Investigate suppliers (internal or external), in terms of risks, licenses, costs, supportability, responsibilities, viability | 3.5-4.5 | 3.5-5.0 |

| | | |
|---|---|---|
| 20. Assess whether the identified architecture-alternatives meet the requirements | 4.5-5.0 | 5.0-5.5 |
| 21. Evaluate the effects of the identified architecture-alternatives on the systems non-functional properties, e.g. response times, safety, security, etc. | 4.0-5.5 | 5.0-5.5 |
| 22. Assess the abilities of the identified architecture-alternatives to be evolved, reused, and expanded | 4.0-5.0 | 4.0-5.0 |
| 23. Assess the impact on system life cycle quality factors, such as producibility, verifiability, ease of distribution, usability, supportability, etc. and changes in the corresponding processes | 4.0-5.0 | 4.0-5.0 |
| 24. Assess requirements on competences, roles and responsibilities for the different product life cycle stages | 2.0-3.0 | 2.5-3.5 |
| 25. Analyze costs, e.g. development costs, maintenance costs, manufacturing costs | 3.0-4.5 | 4.0-5.0 |
| 26. Analyze added values on market opportunities and from technological advances | 3.0-4.0 | 4.0-5.0 |
| 27. Identify and assess commercial and technical risks | 3.5-5.0 | 4.5-5.5 |
| ***Verification and validation*** | | |
| 28. Select architecture parts to be verified and the verification methods to be used | 4.0-5.0 | 4.5-5.5 |
| 29. Select architecture parts to be validated and the validation methods to be used | 4.0-5.0 | 4.5-5.5 |
| ***Technical management*** | | |
| 30. Update and review the architectural description | 3.5-5.0 | 5.0-6.0 |
| 31. Apply configuration management on the architecture description | 3.5-5.0 | 4.5-5.5 |
| ***Communication*** | | |
| 32. Create acceptance and understanding of the need for architectural changes from the management organization | 3.0-4.5 | 5.0-6.0 |
| 33. Support management in decision-making on the proposed architecture and other necessary actions to be taken by them | 3.0-4.5 | 5.0-6.0 |
| 34. Create acceptance and understanding of the architectural changes in the concerned parts of the organization | 3.0-4.5 | 5.0-5.5 |
| 35. Provide help and assist in development projects | 2.5-3.5 | 4.5-5.0 |

### 3.3.2 Characteristic of system refactoring

From literature studies we identified the characterizing factors of general system architecture processes. All found activities, shown in Table 2, were mapped to the characteristics they fulfilled. A comparison was made on how the identified important activities for refactoring (see Section 3.3.1), differ in their distribution of characteristics fulfillment against all found activities from literature. For example, activity *Analyze deficiencies in the current system*, was mapped to *effectiveness* (the activity ensures that right system is designed and for the right purposes). One activity could contribute to one or several characterizing factors, for example the activity *Find future product range and customer demand* was mapped to both *effectiveness* and *short/long term balance* (the activity ensures that both long terms and short terms strategies are considered).

Table 3 explains the characterizing factors and gives the distribution amongst activities, chosen as important, and amongst all activities from the survey. As can be seen in Table 3, the system refactoring activities differ in fulfillment of Quality and Acceptance.

*Table 3. The distribution of the fulfillment of characterizing factors amongst activities in the survey questionnaire and activities important in system refactoring (SR).*

| ***Characterizing factor*** | ***Explanation*** | ***Distribution (%)*** | |
|---|---|---|---|
| | | *SR activities* | *All activities* |
| Effectiveness | the activity ensures that right system is designed and for the right purposes | 38 | 36 |
| Quality | the activity contributes to lowering the number of detected faults after a product is released | 28 | 21 |
| Short/long term balance | the activity ensures that both long terms and short terms strategies are considered | 17 | 17 |
| Efficiency | the activity aids in reducing costs and resource-usage | 7 | 8 |
| Timelines | the activity contributes to that the process is completed before deadline | 7 | 8 |
| Acceptance | the activity contributes to distribute the understanding and acceptance of the architecture in the company | 3 | 10 |

The 35 activities, which were identified as typical activities in general system architecture processes, were categorized into six main areas: Planning, Requirements, Technical solution, Verification and validation, Technical management and Communication. The categorization can be seen in Table 2.

The result of study D shows that the respondent chose all activities in the categories Verification and validation and Technical management as important for system refactoring, 80% of the activities categorized in Requirements, 60% from Technical solution, 40% from the category Planning and none from the category Communication.

### 3.3.3 Need of guidance

The ratings of need for guidance were also made on 0-6 Likert scales. Confidence intervals at $p = 0.95$ were calculated for the ratings of the need of guidance for the 35 activities. We found that the lower levels were above 3.5 for all activities, except for two activities. The confidence intervals for guidance need of all activities are shown in Table 2. We interpret the result as there is a need of guidance throughout the whole general system architecture process, except for these activities:

- Investigate enterprise constraints, such as available resources, competencies
- Assess requirements on competences, roles and responsibilities for the different product life cycle stages

## 3.4 Discussion

In this section we will discuss the outcome of the studies and propose a draft outline for the system refactoring process.

### 3.4.1 Study outcomes

This thesis describes four studies of the system refactoring process who aim to answer three questions:

- Which effects can be expected from a system refactoring?
- What are the drivers of system refactoring decisions?
- What would a guideline need to contain to support system refactoring

Study A and B were conducted to answer the first research question. In study A, interviews about expected effects were conducted with people employed at a Swedish company that was just about to start a system refactoring process, whereas the results from study B were based on literature stud-

ies of reported effects of on-going or completed system refactorings from, mainly, industry. It is worth to note that the concerned system changes were very similar in both studies.

The results from the both studies were quite consistent. However, in study A, effects on more company functions were found. An explanation could be that in study A the interview questions regarded the whole product life cycle, such as product planning and sales. In study B, the relevant literature was studied for effects of the introduction of AUTOSAR. All the identified effects were reported by authors that had mainly experience from system development and that did not investigate effects in other company functions.

The company in study A has today started to use the new system architecture in a development project. A risk that was identified in study A concerned the introduction of new tools and that they were not going to be tested enough. What we see today, concerning the tools and development environment is that adjustments of processes, roles, and responsibilities, are far more time consuming than was expected. An explanation could be that the company has bought several companies during the last years and therefore has a development organization distributed world-wide. Hence the definition of a new development process also includes adapting all current processes and securing that it will fit on all sites. Another risk, identified in study A, was that time constraints would affect system decisions so that the solutions would not be capable to be reused in several projects. We have not seen any sign of this yet, probably because the company already was used of having a common system platform across several machine types.

The results from both study A and study B imply that system verification can be highly affected by system changes, but the results from study C show that this is not an issue that decision-makers investigate prior to a decision. None of the respondents in study C asked for any information about how the proposed system changes would affect system verification or verification of specific components. Therefore we think it is important for guidelines to ensure that all concerned parts of the company and development organization is considered. A guideline must contain some kind of help to identify these parts. However, the results from study D imply that defining the requirements for different steps in the systems' life cycle, such as verification, is an important activity in the system refactoring process. The question that remains is whether this activity should be performed before or after the decision is made.

From the results of study D we conclude that not only a guideline for system refactoring is needed but also guidelines in general for system architecting. Almost all activities seem to need some kind of guidance to be successfully performed. Only for two activities, the need for guidance was rated below our stated threshold. The two activities concerned human requirements for the different product life cycle stages and the investigation of available resources and competencies in the company. We also saw in study

C that these issues were not of large interest amongst the respondents. We can probably interpret these activities as typical for system engineering in general and not specific for system refactoring or even general system architecture processes. On the other hand, we got some conflicting results about the importance of technical management from study C and study D. In study C, only one respondent asked for information about configuration and management of the proposed solution, whereas in study D, the activities related to technical management were rated as important in the process. We interpret the results as that technical management is important during the process but not for the decision to start the process.

According to the results of study D, "investigating coming technologies" and "finding future laws and regulations" were not considered as important activities but on the other hand the activity "Assess ability of the current system to scale for future drivers" was considered as important. We interpret those results as the first two activities are part of the general architecture work and are constantly ongoing. The last activity is specific for the work that includes the investigation of the capabilities of the current system, which is essential to start a process of system refactoring. We also saw in the result of study C, that it was important to consider how the system would accommodate future requirements.

We conclude that we have to redraw the process, as described in Figure 3 (see Section 2.3). From the results of study C and D, we see that many activities of the system refactoring process start before the decision is made. These activities concern the monitoring of the current system, requirement definition and generation of architecture alternatives. The results of these activities are parts of the decision-support needed by the decision-makers. In study C, we see that information about costs and profits have to be added to the support, but these cannot be estimated before some activities have been made.

We were somehow surprised by the results given by study D. No one of the activities that regarded communication of system changes and the reason behind them, were considered as important in system refactoring. Our definition of system refactoring is "changing the architecture or system without changing the visible external functionality" and the same definition was given to the respondents and explained by examples. We believed that this kind of changes that are not related to the introduction of new system features, that attracts customers and increase sales, needs more efforts on persuading the management and organization. The results imply that we were wrong and that organizations probably are mature enough to understand the importance of the electronics system architecture in their products.

### 3.4.2 The system refactoring process

From the results of this thesis work, we can now make a mapping of the system refactoring process. This section will describe a draft outline of the process. The process consists of six stages: System monitoring, Definition of requirements, Generation of architecture solutions, Decision-making, Preparation, and Apply selected solution. Figure 5 gives a simplified illustration of the system refactoring process.

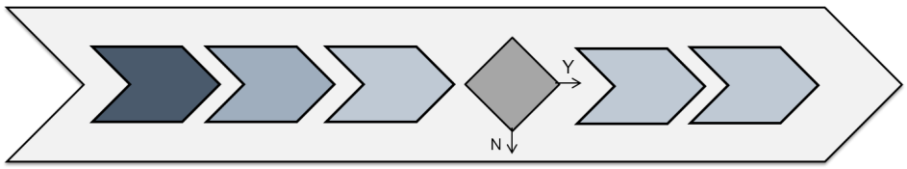


*Figure 5. A re-drawn illustration of the system refactoring process. Decision-making is now an integrated element of the process.*

**System monitoring**

This stage is more a precursor of the system refactoring process, and in fact a part of the general system architecture process. During this stage, the system is monitored for problems and if it is able to meet future requirements. The activities are:

- Find future product range and customer demand
- Analyze deficiencies in the current system
- Assess ability of the current system to scale for future drivers, e.g. coming laws
- Learn about technologies, system architectures and architectural practices

The outcome of this stage is the identification of what system or architectural changes are needed. This is the start of the system refactoring process.

**Definition of requirements**

During this stage the requirements of the system are identified and defined. The most important activity is to identify the technical requirements. The activities in this stage are:

- Define the characteristics required for the product to be cost effective over competitors

- Define the requirements for different steps in the systems life cycle, such as development, verification, maintenance, etc.
- Establish the technical requirements for the system, e.g. identify interfaces and design constraints
- Establish a requirement baseline of the system architecture

The outcome from this stage is a requirement baseline of the system architecture.

**Generation of architecture solutions**

In this step several architecture options that meet the requirement baseline are identified and to some extent assessed. Also the current architecture is examined whether it can be altered or if a new solution is required. The activities in this stage are:

- Develop and identify alternative architecture solutions and selection criteria
- Investigate if the existing architecture can be expanded or adjusted to fit new requirements
- Assess whether the identified architecture-alternatives meet the requirements
- Evaluate the effects of the identified architecture-alternatives on the system's non-functional properties, e.g. response times, safety, security, etc.
- Assess the abilities of the identified architecture-alternatives to be evolved, reused, and expanded
- Assess the impact on system life cycle quality factors, such as producibility, verifiability, ease of distribution, usability, supportability, etc. and changes in the corresponding processes
- Investigate suppliers (internal or external), in terms of risks, licenses, costs, supportability, responsibilities, viability
- Identify and assess commercial and technical risks

The outcome of this stage is one or several architecture solutions that meet the base-lined requirements.

**Decision-making**

In this stage a decision is made whether changes are going to be realized at all and if so, which design solution to choose. The stage includes two main activities:

- Prepare decision-support
- Complete a decision

However, these activities consist of several sub-activities; some of them have already been made in previous stages. The decision-support should consist of:

- *Business case,* including costs and profits.
- *Identified architecture solutions*, including technical details, information about suppliers and identified technical and commercial risks.
- *Requirement baseline*, including future requirements, current requirements, enterprise constraints and external constraints.

*Business case*

The business case should cover the identified costs and profits for each solution. The costs are mainly related to system development, system maintenance and manufacturing, but other affected costs should also be identified and considered. The costs should consider both the initial costs related to the introduction of the solution and the remaining costs after the introduction.

Typical initial costs for system development are development of software, development of hardware, system adjustments, process adjustments and education. Typical remaining costs are tool licenses. Typical initial costs for manufacturing are system adjustments and remaining costs are related to hardware. Typical initial costs for maintenance are also education and system adjustments.

Typical profits that can be gained from each architecture solution are lower lead times for development, and verification, simplified maintenance, less costs in manufacturing, and increased sales.

The identified architecture solutions should be used to find the costs and profits. The changes should be mapped to identify effects on different life cycle stages of the product. For example, does the architecture solution require new development tools? Then, adjustments on processes and tools are required and should be considered. Is there a possibility for model simulations? Then, one can expect shorter lead times for verification. Table 1 shows a mapping of possible effects from typical features of system changes.

*Requirement baseline*

In the previous stages the requirement baseline that should be met by the architecture solutions has been set. Also enterprise and external constraints that are of interest for the decision should be gathered and added to the decision-support. Financing and available resources should be identified to reduce the risks of delays, as well as new legislation that must be fulfilled within the time scope.

The outcome of this stage is a decision on which architecture solution to use.

**Preparation**

This stage includes further exploration of the chosen architecture solution and aims to reduce risks in coming development projects. In previous stages risks have been identified and in this stage actions should be taken to avoid these risks. In this stage also process changes, responsibilities and the need for education should be identified and initial efforts on solving these issues should be made. The outcome of this stage is a plan for risk reductions.

**Apply selected solution**

In this stage the selected architecture solution is applied, most commonly in a planned development project. The activities in this stage are the same as in general system development, but some system refactoring activities remain and are applied in this stage. These are:

- Select architecture parts to be verified and the verification methods to be used
- Select architecture parts to be validated and the validation methods to be used
- Update and review the architectural description
- Apply configuration management on the architecture description

The outcome of this stage is a product that implements the new architecture.

# 4. Related work

In this section we will explore what literature report on the drivers behind systems refactoring and what effects that can be expected. Since most reported work about embedded systems describes introduction of product-platforms or component based development many of the studied papers are in that area. Some described studies will also be in the IT domain and about software refactoring since there are some similarities to the software part of embedded systems. This section will also explore the knowledge about the system architecture process and the role of the architects. The described studies concentrate on findings from real life issues. More related work about the architecture process can be found in Paper D. An important part of the system architecture process is decision-making. Therefore, the last part of this section describes studies of decision-making and methods that aid in decision-making.

## 4.1 Drivers of refactoring

According to Mattsson and Bosch [23], a reorganization without any externally visible changes on functionality most often depends on changed quality requirements. These reorganizations might improve system properties, such as flexibility, maintainability, performance and understandability. Another effect given by reorganizations is lowered effort to develop new products. Mattson and Bosch refer to reorganizations in object-oriented frameworks with reusable components but it might also be applicable for distributed embedded systems, and explains that requirements on costs savings have large impact on the architecture and hence large risks.

One driver might be to get the advantages of using standard architectures. According to Rathmann et al. [24] the automotive industry has solved the problem to cope with constantly new demands from customers on fuel efficiency, safety, driving comfort and legally demands, by adapting their architecture and processes to standards, such as AUTOSAR [9] and CMMI [25]. Therefore the OEMs (Original Equipment Manufacturers) are able to reuse software solutions developed by suppliers. This lowers cost since these solutions are developed and reused in many different vehicle types. He suggests that this might be the right choice even for off-road vehicles. The OEMs producing these vehicles often struggle with low volumes at the same time

as they have to constantly meet new requirements, such as emission regulations.

According to the results of interviews of specialists in systems- and software architecture employed in seven Swedish international companies, conducted by Mustapic et al. [26], the drivers behind the development of new architectures are usually cost reduction or possibilities to use efficient tools, but seldom the arising of new technologies.

These three papers are interesting for our second research question, about the drivers behind system refactoring. They all mention reduced costs as a driver. Strengthening the ability of the system to be developed and maintained, seem to be a way of lowering costs, such as making the system more flexible and understandable. Another way seems to be to spread the costs by reusing solutions amongst several product types.

Wang [27] has a slightly different proposal of drivers for software refactoring. By interviewing 20 software developers from four different companies, he found out that there are also several personal factors that motivate developers to perform a refactoring of software code. Some of these factors are that the organization rewards the developers in different ways when they perform a successful refactoring, but also due to more intrinsic factors, such as the developer getting more self-esteem, or to habits of the developer.

These kinds of personal drivers might be significant for an individual developer that is only responsible for a small amount of software code in an IT system but within embedded systems development the situation is different. The projects are often large with several stakeholders with different technical expertise that have to cooperate and system refactoring is dealing with both software and hardware. Thus, we can assume that the drivers of system refactoring are somewhat different but may also be personal. There is no guarantee that stakeholders do not want to reduce their own efforts instead of considering the total costs of the company.

## 4.2 Effects from refactoring

We searched in papers after the expected effects of system refactoring by looking at reported effects from introductions of reusable solutions. As mentioned in Section 4.1, reduced costs seem to be a main driver of system refactoring. This is also reported as a positive effect when the same solution is reused in several products. However, it is not easy to make the shared solution able to fit in different contexts and it seems that the new way of working affects the organization.

### 4.2.1 Reusing components

To explore empirically if software reuse is as beneficial as one could imagine, Mohagheghi and Conrad [8] conducted a comprehensive case study at a large Telecom company. Three years of data was collected from three releases of two products. The products consisted of a lower layer of reused blocks, common for both the products, and a product-specific application layer. They found a reduction of software errors and development lead times.

Instead of reusing internally developed software components, companies can buy them from suppliers, so called COTS (Commercial-off-the-shelf) components. According to McKinney [28], COTS can lower time-to-market for products. However, both paper of McKinney [28] and Morisio et al. [29] report that the company cannot rely on the availability of the COTS from the supplier. Also integration problems late in the projects were reported as a problem [29]. Another consequence the company has to deal with is adapting the development process to get the benefits of using COTS. These papers show how important it is to consider costs and benefits for process changes as well as for enhanced system quality properties when deciding about refactoring an existing system. To deal with the problems, Morisio et al. [29], suggest that the experienced technical staff should be included in the decision-making process about which COTS to choose and that a specific team is responsible for the activities in evaluating, choosing, using and procuring COTS.

### 4.2.2 Product-line development

Tsakiris [30] reports effects from the introduction of a product-line platform to be used amongst several vehicle brands. One reason for introducing the platform is to manage complexity. He reports problems with the large differences between several vehicles, especially the large variations of control systems. Therefore they established an architectural framework, with a functional view that simplified common understanding of the functions and helped them reduce signals in the system. A special department group was responsible for the data dictionary and signal list of vehicle projects and developers had to request updates via the group.

From a sister company to the above mentioned company, Eklund at al. [31] also present an introduction of a reference architecture in the development of automotive electronics systems. Due to a well-defined design and identified non-functional properties the reference architecture became a success and the cost was optimized and could be used in several vehicle projects. However, to continue being successful and get return on investment, the architecture requires maintenance and distribution costs that are higher than the actual development cost of the architecture. Consequently, it is important for decision-makers to include these factors when deciding about

introducing these kinds of platforms. The authors also point out the importance of support from management on all organization levels since the introduction of the reference architecture will impact processes and organization.

All these above reported experiences from introducing product-line architecture in an organization confirm the findings from a study of Bosch [11]. He agrees with Eklund et al. [31] on that support from management is important and suggests that the managers should be exposed to more details and technical aspects of the product-line project.

Bosch has further studied problems in product-line development in [32, 33]. Bosch and Bosch-Sijtsema [33] report a trend going from product-line development to software ecosystems, meaning that, except for the platform and the internal developers building on top of the platform, there also exists a community of external developers that extend the products after they have been released by the company. Due to these three trends the complexity is increasing. To deal with the problems the companies must stop being integration-centric, i.e. most effort is given to the step where parts are integrated and validated, and start being composition-oriented, which means that teams are working independently, and are not dependent on each other's releases, which is achieved by architectural rules and constraints.

Also Crnkovic et al. [34] present how the introduction of product-line architecture affected a large company that develops embedded industrial systems. Also in this study the development process was affected by the changes. Most of the reported problems were related to interface or architectural mismatches and encapsulation of service in components. They also mean that to be successful in product-line development more effort is needed on the overall architecture.

**Product-line development in IT systems**

There are also reports from using product-lines in the development of IT systems. Some effects seem to be the same as in the development of embedded systems.

Verlage and Kiesgen [35] report experiences from the use of product-line development in a relative small company. They conclude, as Eklund et al. [31], that product-line development is not just an investment in the introduction but also a constant need for cleaning up the platform. The introduction has also affected the company organization since new roles are needed.

Nonaka et al. [36] tested a model that can be used for examining effects of investments on scenarios of architecture reuse. They conclude that these investments should consider the number of planned products over the lifetime of the product-line and required time-to-market for the first product to be delivered.

Slyngstad et al. [37, 38] describe studies of evolution risks in IT systems. The largest risks were "Lack of stakeholder communication affected imple-

mentation of new and changed architectural requirements negatively" and "Poor clustering of functionality affected performance negatively" and the preferred strategies to deal with these problems were "Allow additional time for communication and feedback" and "Refactoring the architecture". From this we learn that system refactoring is an important activity for system architects.

## 4.3 The system architecture process and the role of the architect

This thesis focuses on the system refactoring process. However, we believe that there are similarities between the processes for system refactoring and system architecting. Therefore we have explored the role of the system architects and what the process for system architecting looks like in industry.

Ahmed and Capretz [39] investigated empirically the effect of some key architecture activities in the performance of software product-lines. The measures were reduced costs and development time, market growth and financial strength. Domain engineering, requirements engineering, requirements modeling, commonality management, variability management and architecture artifacts managements supported the product-line performance positively. The study results show how important architecture activities are when introducing product-line development, which is normally preceded by a system refactoring.

To find out more about the system architecture process Mustapic et al. [26] interviewed specialists in systems- and software architecture from companies that all developed complex distributed system. When developing the system architecture for a new product it is important for the system architect to have knowledge and experience from the development of similar systems since the system architecture is reused between projects with only slight changes. Even though it was considered important to have a great knowledge of system development, the interest in the process amongst system architects was quite low. They conclude that it is important to keep core architecture teams relative small that define the fundamental principles of the software and system architecture. It is still important to communicate the derived architecture throughout the organization. A suggestion is that the architect takes the role of a technical leader in development projects.

Faber [40] describes his experiences of the architecture role in agile development. By letting the architects be involved in the development projects, flaws were discovered and corrected in time. At the same time the architect learned about the system. Earlier, when the architect just delivered a completed architecture to the application developers, mistakes were discovered too late.

Wallin et al. present findings in [41, 42] from case studies performed at two heavy vehicle manufacturers and one car manufacturer. In both studies they found that the companies lack clear architecture processes and methods for evaluating business values when selecting architecture. Decision-making was weak and mostly made on gut-feeling. They found that one company did not have a long-term architectural strategy [41]. The other study indicated lack of quality measures during development, and therefore architectural quality issues were not revealed until late in development projects [42]. These results imply the need for creating processes for the system architecture work, that fit global organizations and that clear out the responsibilities of the system architecture departments.

Lindgren et al. [43] present similar findings as Wallin et al. [41, 42]. The authors studied how software architecture is considered in release planning. They conclude that product management has generally little awareness of software architecture. In companies where product management is responsible for release planning it is even more important that software architects participate in decision-making. Most decisions are today made on gut-feeling and methods for balancing quality improvements and feature growth are missing.

As described by Eklund [31] and Bosch [11] there is a need for support by management during refactoring activities. Unfortunately, Wallin et al. [41, 42], report that management often lacks knowledge about electrical and software systems. The authors suggest that management should be educated in the area. This issue should be considered in a guideline for system architecture processes. As described above, Mustapic et al. [26] suggest that the architect takes the lead during development. Faber [40] also suggests that architects should be more engaged in development. These approaches might lead to a better understanding about architecture in the organization and perhaps also clear out the interface of the architecture department.

## 4.4 Decision-making

Decision-making is an important part of the work the system architect does. Even if the actual decisions are usually made by management, the system architect must provide the correct data. According to Hammond et al. [44], decision-making should include problem formulation, definitions of objectives or goals, identifying alternatives and their respective consequences, and trade-offs between the objectives when selecting amongst alternatives. Unfortunately, Tversky and Kahneman [45] clame that systematic and predictable errors are common when people make judgement of probabilities of events; well-educated and experienced persons are not an exception. As mentioned earlier, distributed embedded systems are very complex, and hence very hard for making assumptions about. We have therefore investi-

gated what tools that are available for analyses and decisions-making when making changes in the architecture of embedded systems.

### 4.4.1 The Software Analysis Architecture Method, SAAM

The Software Analysis Architecture Method (SAAM) was developed by Kazman et al. [46]. The method is used for evaluating different architecture alternatives for a desired property. It is based on a well understood system architecture and is therefore started by providing a clear architecture description. Next, concrete tasks, which are typical for the desired property, are chosen. For example the task "Changing the communication protocol" may be appropriate for the property "Maintainability". Each architecture alternative is then analyzed and evaluated for how well it manages to perform the tasks. SAAM gives a structure for the analysis process but does not prescribe any method for how to evaluate the different architecture alternatives.

### 4.4.2 Architecture Level Modifiability Analysis, ALMA

SAAM underlies several other analysis methods, like the Architecture Level Modifiability Analysis (ALMA) [47]. ALMA was developed for determining the modifiability level of architecture elements. The first activity is to determine goals for the analysis, such as predicting maintenance cost, performing a risk assessment or selecting an architecture.

### 4.4.3 Architecture Trade-off Analysis Method, ATAM

SAAM is also the predecessor of the Architecture Trade-off Analysis Method (ATAM) [48]. The primary goal of ATAM is to assess the consequences of architectural decisions in the light of quality attribute requirements by identifying risks, sensitivity points and tradeoff points in the system. This is achieved by finding scenarios, using utility trees and generating system quality attributes. Most of the work is carried out by smaller stakeholder groups, consisting of architects, customers and the evaluators. ATAM proposes the use of larger brainstorming meetings with all stakeholders to prioritize the scenarios. The basic idea is to stimulate the creativity and communication of new ideas. The disadvantage of ATAM is that it is very time consuming but the many hours of work may also be a strength since the architecture gets very thoroughly investigated.

### 4.4.4 Cost Benefit Analysis Method, CBAM

The Cost Benefit Analysis Method (CBAM) [49] is an extension to ATAM and an attempt to map costs and benefits to system quality attributes and business goals by determining the relation between them. CBAM starts

where ATAM ends and adds cost to architect decisions and benefits to quality attributes.

### 4.4.5 Analytic Hierarchy Process, AHP

The Analytic Hierarchy Process (AHP) is a method for managing complex decisions based on mathematics and psychology [50]. This method starts by identifying architecture goals, criteria and alternatives. The stakeholders organize the criteria in a hierarchy by pairwise comparing them against all other criteria. Then the stakeholders pairwise compare the architecture alternatives against each other for each criterion. The problem with AHP is that the number of comparisons rapidly increases by the number of alternatives and criteria. AHP only helps in comparing alternatives against each other. Some other method must be used for identifying effects caused by the alternatives, e.g. ATAM [51].

### 4.4.6 More methods

Larsson et al. [52] present a method that deals with organizational effects. The method analyzes the influences that a change in architecture will have on the development processes. The method uses scenarios to identify the goals and activities needed for an architectural change which then can be used for finding affected processes. They mean that organization, architecture and processes are related to each other and when changing one of these the others may be influenced.

Tang et al. [53] propose a tool for software architecture design reasoning. This tool takes business goals and other design concerns, like functional- and non-functional requirements, organization, and technologies into account. The tool captures design-rationales for relevant decisions, like trade-offs, risks, costs, and constraints, and the actual design. A similar approach is presented by Jansen and Bosch [54]. They present a tool where software architecture can be seen as a set of design decisions and which makes explicit the relationships between the design decisions and the software architecture.

Riebisch and Wolfarth [55] propose an ALMA based method for evaluating alternatives of architectural decisions, for both architectural design and refactoring. They conclude that the use of impact analysis methods help identifying risks of side effects, additional effort and changed behavior of the system and aid in determining if the possible risk is worth the effort.

Leitch and Stroulia [56] propose a model for predicting return-on-investments (ROI) for software refactoring. Since software refactoring is usually performed to lessen maintenance costs they calculate ROI as the savings of maintenance from a proposed refactoring divided by the devel-

opment cost. The refactoring is cost effective if ROI is greater or equal to one.

Clements and Bass [57] propose a way of gathering business goals by proposing a business goal viewpoint. They mean that system architecture have requirements that can otherwise be missed. The gathering of business goals gives the architect a possibility to accommodate the architecture for future demands. The proposed method consists of identifying stakeholders, gathering and prioritizing business goals from the stakeholders and identifying effects on architecture. Typical aspects of business goals are organization growth, financial objectives, personal objectives, responsibilities to employees, responsibilities to the country and society, responsibilities to shareholders, market position, improved business processes, product quality and product reputation.

A method that also captures requirements from business goals is proposed by Kamath [58], who claims that when IT-companies merge or reorganize the business complexity increases which is reflected in the architecture. He proposes a method for linking business architecture to application architecture for IT systems. In that way, the company can easier focus on future investments, outsourcing, and business strategies.

### 4.4.7 Utility of methods

Breivold and Crnkovic [59] have done a comprehensive survey of methods that support software architecture evolution. There are methods focusing on software quality during software architecture design. Later in the design phase, when one or several architecture alternatives have started to take shape, several methods help to evaluate the quality of the software architecture, like evolvability. SAAM [46], ATAM [48], and ALMA [47] (see Sections 4.4.1-4.4.3) are examples of such methods. A third category is methods that consider economic values, such as CBAM [49] (see Section 4.4.4). Other types of methods consider architectural knowledge management and modeling techniques. Most methods are primarily used for analyzing and evaluating software architecture, but also for creating alternatives, understanding requirements and creating business cases.

As stated by Wallin et al. [41, 42] practitioners lack methods for evaluations of business values and decision-making even though we have presented several methods in this thesis. Salonen and Perttula [60] claim in a paper from 2005, that the existing methods do not fit in industry and they suggest that new methods are developed that support concept selection. They studied the utilization of concept selection methods in Finnish industry. “Concept review meetings” was the most widely used method, followed by “Intuitive selection of concept”. Other common methods were checklists and expert assessments. Less than five percent used formal systematic methods, e.g. AHP and Pugh’s matrix.

Although there are several methods proposed by the research community it seems like they are not implemented and used in industry to a large extent. Perhaps, the research community has failed to announce the methods to industry, or maybe the available methods are too time consuming to be effective. Guidelines for system architecting could help by proposing suitable methods for certain decisions.

# 5. Conclusion

There is an emerging need for guidance and support in the system architecture processes. A guideline for system refactoring contributes to a part thereof. Such a guideline should explain what effects to expect for different system changes and helps providing required information for decision-making.

We have seen that the characteristics of the process follow the general architecture process. The focus seems to be on effectiveness, quality and the balancing between short-term and long-term goals. The process seems to emphasize on efficiency and timeliness less, probably since it needs time to achieve qualitative and effective solutions. The reason is that system refactoring can give effects that not only affect the system but also impacts the entire organization. Despite these cross-organizational effects, it is common that not all affected parts are considered when deciding on the implementation of changes. It is therefore important that a guideline for this process ensures that all affected parts are identified and taken into account while making a decision. A guideline must also emphasize on the importance of creating understanding and acceptance within the organization, since this activity is mostly forgotten by practitioners but still important for decision-makers.

## 5.1 Contribution

Our definition of system refactoring is "architecture or system changes without changes to the visible external functionality". The profits from these changes are not as obvious as for the introduction of new product features that give increased sale. This thesis tries to highlight the benefits, and costs, that can be expected from different types of architectural changes and how decision-support can be prepared.

We have mapped the system refactoring process, its activities, its drivers, and expected effects. Now we have enough information to create a preliminary guideline. By help from the results presented in this thesis, companies who are facing a decision about system refactoring, can assess what they should study before a decision is made. Since we have identified probable effects of certain types of system changes, the decision-maker knows what effects can be expected when constructing a business case, including costs and profits. With the identification of effects, companies can now also pre-

pare themselves in time before the system changes are introduced, so that risks, such as delays in development projects, are minimized.

## 5.2 Future work

We have mapped the process and the next step will be to create a preliminary guideline that will be refined through testing in industrial projects. Several of the identified activities will presumably be defined at a lower level and it will be more clearly described when in the process they should be carried out.

Some open questions still have to be answered:

- Did we manage to isolate the process for system refactoring? The found characteristics of system refactoring, from study D, only slightly differ from general system architecting. A follow-up study is needed to ensure that we capture the real system refactoring process.
- How can we ensure consideration of all main effects? Related questions are: "How do we know what are the important effects to consider when preparing or deciding about system refactoring?" and "How far should we look, when is it appropriate to say that we investigated enough?". We have seen that decision-makers focus on what is currently on their minds and seem to forget about the rest of the company organization. We have also seen that technical details are more considered than human related details, such as processes, responsibilities, and competence. Maybe that is not important for the actual decision of performing the system changes, but probably important to consider when preparing for the work of system refactoring. If these factors must be considered in on-going product development projects that are going to use the refactored architecture, they may get delayed and miss their deadlines.
- Is there a precedence order of the activities in system refactoring? In Section 3.4.2, we propose a flow order of the activities in system refactoring. However, this order is only a draft and based upon our findings of involved activities, drivers and effects. We have not yet investigated if any specific activity must be performed before another specific activity.

# References


1 GM recalls 1.3 million cars over steering fault. *http://www.msnbc.msn.com/id/35655693/ Cited 11/24/2010* (The Associated Press, 2010).

2 Toyota återkallar fyra modeller. *http://www.sydsvenskan.se/bil/article627859/Forsta-stamningen-mot-Toyota.html Cited 11/24/2010* (Sydsvenskan, 2010).

3 Nissan recalls 2 million cars worldwide. *http://www.msnbc.msn.com/id/39885304 Cited 11/24/2010* (The Associated Press, 2010).

4 Maintaining stability on the road. *http://www.scania.com/media/feature-stories/technology/esp.aspx Cited 20/09/2009* (Scania, Södertälje, Sweden, 2008).

5 The electronic stability control ESC. *http://www.conti-online.com/ Cited 10/28/2009* (Continental, 2007).

6 Larses, O. Architecting and modeling automotive embedded systems. p. 307 (KTH, Stockholm, 2005).

7 Bass, L., Clements, P. and Kazman, R. *Software architecture in practice*. (Addison-Wesley Professional, 2003).

8 Mohagheghi, P. and Conradi, R. An empirical investigation of software reuse benefits in a large telecom product. *Transactions on Software Engineering and Methodology*, 2008, 17(3), 1-31.

9 Welcome to the AUTOSAR development partnership. *http://www.autosar.org Cited 6/14/2009* (AUTOSAR, 2009).

10 Crnkovic, I. and Larsson, M. *Building reliable component-based software systems*. (Artech Hous, 2002).

11 Bosch, J. Product-line architectures in industry: A case study. *Proceedings of the 21st International Conference on Software Engineering*, pp. 544 - 554 (ACM New York, NY, USA, Los Angeles, California, United States, 1999).

12 Systems and software engineering — Recommended practice for architectural description of software-intensive systems. *ISO/IEC 42010:2007, IEEE Std 1471-2000* (ISO/IEC/IEEE, 2000).

13 Systems and software engineering — System life cycle processes. *ISO/IEC 15288:2008, IEEE Std 15288-2008* (ISO/IEC/IEEE, 2008).

14 Standard for systems and software engineering - Software life cycle processes. *ISO/IEC 12207:2008, IEEE Std 12207-2008* (ISO/IEC/IEEE, 2008).

15 Application and management of the systems engineering process. *IEEE Std 1220-1998* (IEEE, 1998).

16 Firesmith, D.G., Capell, P., Falkenthal, D., Hammons, C.B., Latimer, D. and Merendino, T. *MFESA - The method framework for engineering system architectures*. (Auerbach Publications, 2008).

17 Muller, G. System architecting. *http://www.gaudisite.nl/ Cited 8/11/2011* (Gaudi System Architecting, 2003).

18 Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B. and Wesslén, A. *Experimentation in software engineering: An introduction*. (Kluver Academic Publishers, United States of America, 2000).

19 Yin, R. *Case study research: Design and methods*. (Sage Publications, United States of America, 2002).

20 Kitchenham, B. and Charters, S. Guidelines for performing systematic literature reviews in software engineering. *EBSE-2007-01* (Keele University, UK, 2007).

21 DeProy, E. and Gitlin, L.N. *Forskning - en introduktion*. (Studentlitteratur, Lund, Sweden, 1999).

22 Hall, C. Open systems for military avionics: A technology overview. *ERA 2007-0669* (Avionics Systems Standardization Committee, 2007).

23 Mattsson, M. and Bosch, J. Frameworks as components: A classification of framework evolution. *Nordic Workshop on Programming Environment Research*, pp. 163-174 (Bleking institute of technology, Ronneby, Sweden, 1998).

24 Rathmann, S., Fischerkeller, R. and Schweiker, A. Latest trends in automotive electronic systems - Highway meets off-highway? *Agricultural Engineering International: the CIGR Ejournal*, 2007, IX(ATOE 07 012).

25 CMMI® for development, Version 1.3. *ESC-TR-2010-033* (CMMI Product Team, Software Engineering Institute, Carnegie Mellon University, 2010).

26 Mustapic, G., Wall, A., Norström, C., Crnkovic, I., Sandström, K., Fröberg, J. and Andersson, J. Real world influences on software architecture - interviews with industrial system experts. *Proceedings*

*of the 4th Working IEEE/IFIP Conference on Software Architecture*, pp. 101-111 (IEEE Computer Society, Oslo, Norway, 2004).

27 Wang, Y. What motivate software engineers to refactor source code? Evidences from professional developers. *25th IEEE International Conference on Software Maintenance*, pp. 413-416 (IEEE, Edmonton, Alberta, Canada, 2009).

28 McKinney, D. Impact of commercial off-the-shelf (COTS) software on the interface between systems and software engineering. *Proceedings of the 21st International Conference on Software Engineering* (ACM, Los Angeles, California, United States, 1999).

29 Morisio, M., Seaman, C.B., Parra, A.T., Basili, V.R., Kraft, S.E. and Condon, S.E. Investigating and improving a COTS-based software development. *Proceedings of the 22nd International conference on Software engineering*, pp. 32-41 (ACM, Limerick, Ireland, 2000).

30 Tsakiris, A. Managing software interfaces of on-board automotive controllers. *IEEE Software*, 2011, 28(1), 73-76.

31 Eklund, U., Askerdal, Ö., Granholm, J., Alminger, A. and Axelsson, J. Experience of introducing reference architectures in the development of automotive electronic systems. *Proceedings of the 2nd International Workshop on Software Engineering for Automotive Systems*, pp. 1-6 (ACM, St. Louis, Missouri, 2005).

32 Bosch, J. The challenges of broadening the scope of software product families. *Communication of the ACM*, 2006, 49(12), 41-44.

33 Bosch, J. and Bosch-Sijtsema, P. From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, 2009, 83(1), 67-76.

34 Crnkovic, I., Larsson, S. and Chaudron, M. Component-based development process and component lifecycle. *27th International Conference on Information Technology Interfaces*, pp. 591-596 (IEEE Computer Society, Dubrovnik, Croatia, 2005).

35 Verlage, M. and Kiesgen, T. Five years of product line engineering in a small company. *Proceedings of the 27th International Conference on Software Engineering*, pp. 534-543 (ACM, St. Louis, Missouri, 2005).

36 Nonaka, M., Zhu, L., Babar, M.A. and Staples, M. Impact of architecture and quality investment in software product line development. *11th International Software Product Line Conference*, pp. 63-73 (IEEE Computer Society, Kyoto, Japan, 2007).

37 Slyngstad, O., Li, J., Conradi, R. and Babar, A. Identifying and understanding architectural risks in software evolution: An empirical study. *Proceedings of the 9th International Conference on Product-*

*Focused Software Process Improvement*, pp. 400-414 (Springer-Verlag, Monte Porzio Catone, Italy, 2008).

38 Slyngstad, O.P.N., Conradi, R., Babar, M.A., Clerc, V. and van Vliet, H. Risks and risk management in software architecture evolution: An industrial survey. *15th Asia-Pacific Software Engineering Conference*, pp. 101-108 (IEEE, Beijing, China, 2008).

39 Ahmed, F. and Capretz, L.F. The software product line architecture: An empirical investigation of key process activities. *Information and Software Technology*, 2008, 50(11), 1098-1113.

40 Faber, R. Architects as service providers. *IEEE Software*, 2011, 27(2), 33-40.

41 Wallin, P. and Axelsson, J. A case study of issues related to automotive E/E system architecture development. *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pp. 87-95 (IEEE Computer Society, Belfast, Northern Ireland, 2008).

42 Wallin, P., Johnsson, S. and Axelsson, J. Issues related to development of E/E product line architectures in heavy vehicles. *42nd Hawaii International Conference on System Sciences*, pp. 1-10 (IEEE Computer Society, Waikoloa, Big Island, Hawaii, 2009).

43 Lindgren, M., Norstrom, C., Wall, A. and Land, R. Importance of software architecture during release planning. *7th Working IEEE/IFIP Conference on Software Architecture*, pp. 253-256 (IEEE Computer Society, Vancouver, BC, Canada, 2008).

44 Hammond, J.S., Keeney, R.L. and Raiffa, H. *Fatta smarta beslut*. (Forma Books AB, 2001).

45 Tversky, A. and Kahneman, D. Judgment under uncertainty: Heuristics and biasis. *Science, New Series*, 1974, 185(No. 4157), 1124-1131.

46 Kazman, R., Bass, L., Abowd, G. and Webb, M. SAAM: A method for analyzing the properties of software architectures. *Proceedings of the 16th International Conference on Software Engineering*, pp. 81-90 (IEEE Computer Society, Sorrento, Italy, 1994).

47 Bengtsson, P.O., Lassing, N., Bosch, J. and van Vliet, H. Architecture-level modifiability analysis (ALMA). *Journal of Systems and Software*, 2004, 69(1-2), 129-147.

48 Kazman, R., Klein, M. and Clements, P. ATAM: Method for architecture evaluation. *CMU/SEI-2000-TR-004* (Carnegie Mellon University, Pittsburgh, Pennsylvania, 2000).

49 Kazman, R., Asundi, J. and Klein, M. Quantifying the costs and benefits of architectural decisions. *Proceedings of the 23rd*

*International Conference on Software Engineering*, pp. 297-306 (IEEE Computer Society, Toronto, Ontario, Canada, 2001).

50 Saaty, T.L. How to make a decision. *European Journal of Operational Research*, 1990, 48, 9-26.

51 Wallin, P., Fröberg, J. and Axelsson, J. Making decisions in integration of automotive software and electronics: A method based on ATAM and AHP. *Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems*, p. 5 (IEEE Computer Society, Minneapolis, USA, 2007).

52 Larsson, S., Wall, A. and Wallin, P. Assessing the influence on processes when evolving the software architecture. *9th International Workshop on Principles of Software Evolution* (ACM, Dubrovnik, Croatia, 2007).

53 Tang, A., Han, J. and Vasa, R. Software architecture design reasoning: A case for improved methodology support. *IEEE Software*, 2009, 26(2), 43-49.

54 Jansen, A. and Bosch, J. Software Architecture as a set of architectural design decisions. *5th Working IEEE/IFIP Conference on Software Architecture*, pp. 109-120 (IEEE Computer Society, Pittsburgh, Pennsylvania, 2005).

55 Riebisch, M. and Wohlfarth, S. Introducing impact analysis for architectural decisions. *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pp. 381-392 (IEEE Computer Society, Tucson, Arizona, 2007).

56 Leitch, R. and Stroulia, E. Assessing the maintainability benefits of design restructuring using dependency analysis. *Proceedings of the 9th International Software Metrics Symposium*, pp. 309-322 (IEEE Computer Society, Sydney, Australia, 2003).

57 Clements, P. and Bass, L. The business goals viewpoint. *IEEE Software*, 2010, 27(6), 38-45.

58 Kamath, S. Capabilities and features: Linking business and application architectures. *9th Working IEEE/IFIP conference on Software architecture*, pp. 12-21 (IEEE Computer Society, Boulder, Colorado, USA, 2011).

59 Breivold, H.P. and Crnkovic, I. A systematic review on architecting for software evolvability. *Proceedings of the 21st Australian Software Engineering Conference*, pp. 13-22 (IEEE Computer Society, Auckland, New Zealand, 2010).

60 Salonen, M. and Perttula, M. Utilization of concept selection methods - A survey of finnish industry. *17th International Conference on Design Theory and Methodology*, pp. 527-535 (ASME, Long Beach, California, 2005).