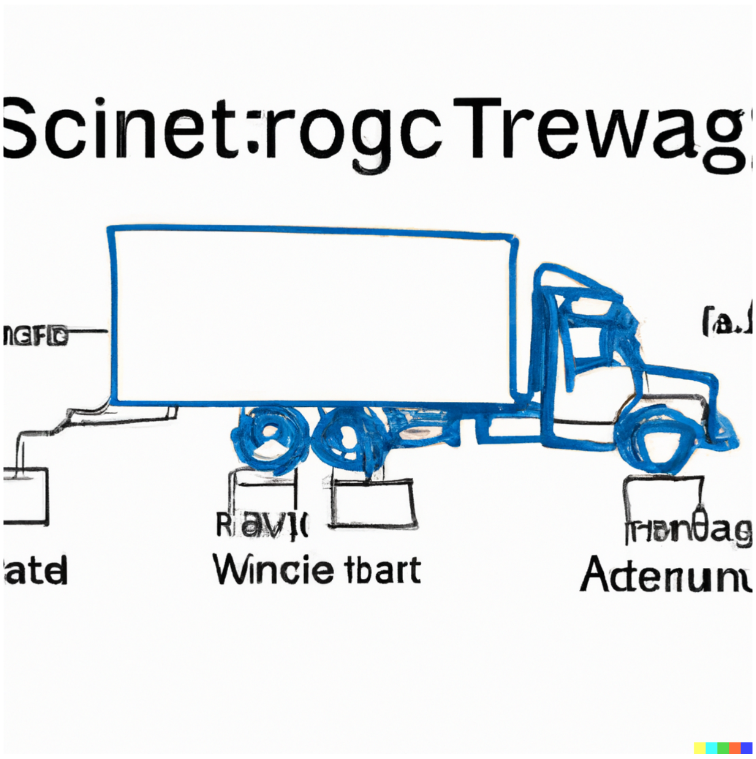
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Factors affecting the migration of a large embedded system

An automotive case study

Master's thesis in Computer science and engineering

Linnea Johansson & Wanting Xu

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Factors affecting the migration of a large embedded system

An automotive case study

Linnea Johansson & Wanting Xu

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Interaction Design and Software Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Factors Affecting the Migration of a Large Embedded System
An automotive case study
Linnea Johansson & Wanting Xu

Supervisor: Regina Hebig, University of Rostock
Supervisor: Philipp Leitner, Department of Computer Science and Engineering
Advisor: Dhasarathy Phasarathy, Volvo Trucks
Examiner: Jennifer Horkoff, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Division of Interaction Design and Software Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: DALL-E creation with prompt *software architecture of a truck*

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Factors Affecting the Migration of a Large Embedded System
An automotive case study
Linnea Johansson & Wanting Xu
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Migrating a small project can be a hard thing to do. Migrating the code at a large company is even more challenging. In this thesis, the focus is on trying to find common elements in how a large automotive company migrates their embedded code.

We interviewed ten people from various teams to find common topics that make it harder or easier to migrate. We also developed an architecture recovery tool called GRASS to look at metrics such as fan-in and fan-out of the source code made of C code. During the latter part of spring, we held a focus group with the people interviewed.

By combining these data, we found that it is harder to migrate code if a team has dependencies on many other teams, or if there are hard dependencies on the supplier's code. GRASS was able to identify these aspects, but the interviewees thought that GRASS needed improvements in order to be useful to them in the migrating process. Additionally, the hardware aspect of an embedded system can make it harder if the hardware is limited in capacity, or if there are real-time requirements that make latencies in the system unacceptable. Lastly, we found that some teams had small parts that they were able to automate, and these automation scripts might be useful to others.

Keywords: software, migration, automotive, AUTOSAR, embedded.

Acknowledgements

We are extremely grateful to both our supervisors Regina Hebig and Philipp Leitner for always being there for us and for all the guidance you have provided. You both have shaped this thesis and helped out immensely.

Special thanks to Dhasarathy Parthasarathy and Cecilia Ekelin from Volvo Trucks for all the valuable help and input. Thanks should also go to all the developers, architects, and the manager at Volvo who gave of their time. Without you, this thesis would not have been possible.

Lastly, we would like to extend our thanks to our examiner Jennifer Horkoff for leading us in the right direction. We also want to thank our family and friends who have provided moral support and table tennis distraction during the spring. Our last honorable mention goes to Jinji the cat, who has held our spirits high.

Linnea Johansson & Wanting Xu, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis, listed in alphabetical order:

AUTOSAR	AUTomotive Open System ARchitecture
E/E	Electrical and Electronic
ECU	Electronic Control Unit
LDC	Logic Design Component
LOC	Lines Of Code
OEM	Original Equipment Manufacturer
RTE	Real-Time Environment
SOA	Service-Oriented Architecture
SWC	Software Component
VFB	Virtual Functional Bus

Contents

List of Figures

List of Tables

1 Introduction

Technology is growing fast, and the business requirements for the automotive industry are dramatically changing. Customer needs force the automotive industry to deliver new functionality while working on improving the safety of the vehicles. Therefore, vehicle companies have to develop fast to keep up with the market.

In traditional vehicle development, improvements have largely focused on hardware. For example, adding a bus body to a truck chassis made an early type of bus [1]. The list goes on, but in recent years the focus has shifted to more software-based improvements. With this, it has also become crucial to be able to update current software in a vehicle quickly.

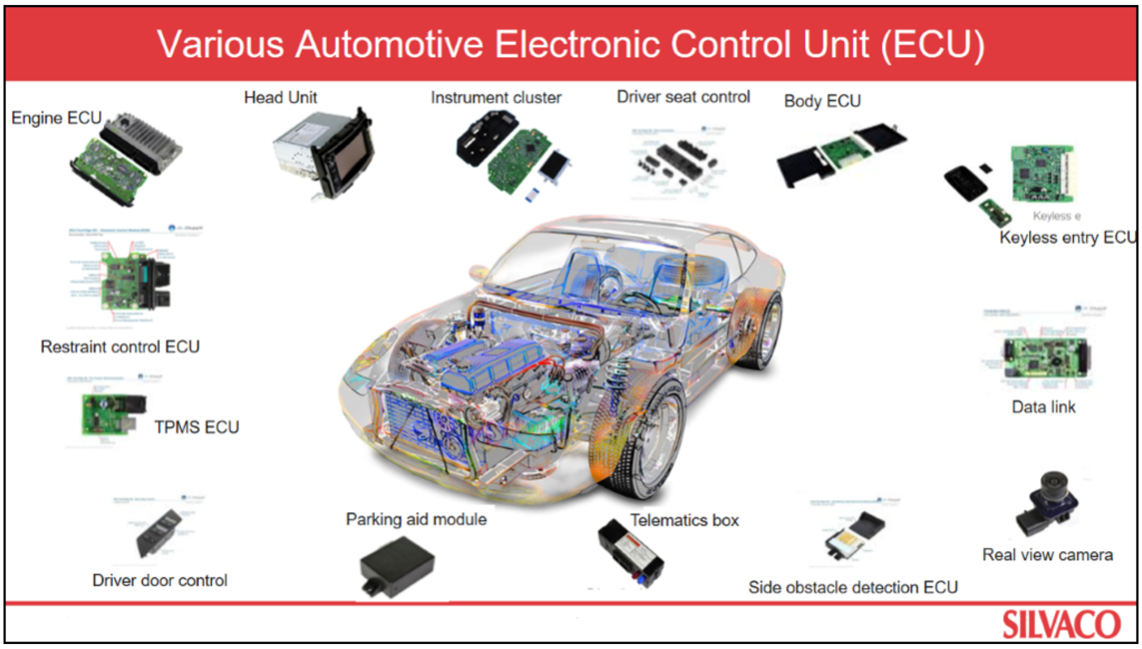


Figure 1.1: Different ECUs in a car [2]. These are examples of common ECUs that are placed in a car, with different functionalities. The ECUs are connected and communicate with each other, normally through CAN.

The vehicle industry has been going through a transformation in recent years. With the rise of self-driving cars, infotainment systems in cars, automatic lane detection, and other improvements of functionalities in vehicles, it is essential for older companies in the business to not be left behind.

Embedded software for vehicles is written to be deployed on ECUs (Electronic Control Units), see figure 1.1, which works like a small computer. In the classic case, there are a lot of ECUs distributed in the vehicle, that handle different parts of the functionality such as the wipers, the brakes, and the lights. The ECUs can send information to other ECUs through data buses. The speed of the information is based on the physical distance between the ECUs, and therefore several ECUs can be in a small cluster to talk to each other in the vehicle.

To reduce software complexity, a standard for automotive software was developed. AUTOSAR was founded in 2002 and has since provided a software framework for companies such as BMW, Ford, Mercedes, Toyota, Volkswagen, and Volvo. The first version of AUTOSAR, the classic platform, was released in 2004 [3]. AUTOSAR released the adaptive AUTOSAR platform in 2017 in order to meet the future needs of the vehicle industry. Adaptive AUTOSAR allows companies to update the functionalities of their products easily and quickly.

Volvo Trucks as a large automotive company is trying to make changes to move to adaptive AUTOSAR to gain the benefits brought by it. Volvo has a long history of producing vehicles and with the development of new technologies, the architecture of vehicles is growing larger and more complicated. Because of the large, complex architecture, they have to make incremental changes to migrate to the new framework. For now, Volvo Trucks are moving toward having one central ECU and moving some functionalities to the adaptive AUTOSAR platform. The details of the migration differs between the teams, however most teams need to adjust their software for the centralized structure. The change to the new AUTOSAR platform is encouraged rather than something most teams have to do at the moment. The purpose is mainly to improve the maintainability of the embedded architecture for future use.

1.1 Statement of the Problem

Volvo Trucks is a big company that needs to make changes to build a more maintainable architecture. However, there are many problems in taking this big step. The current system, A1, is built on classic AUTOSAR. A1 has been developed for a long time with many features and the hardware and software are tightly coupled. If there are changes on either the hardware or software, the other needs to change as well. In A1, the components are distributed and interdependent. Volvo Trucks is currently performing a migration of its software and hardware architecture from the A1 structure to a new A2 structure. A2 is planned to have a centralized structure and use adaptive AUTOSAR which enables service-oriented architecture.

The migration process is hard for a large organization because of several aspects, mainly with finance and human resources. The teams in the organization take on different responsibilities in the migration process. This paper aims to find out what makes it hard for a large vehicle organization like Volvo Trucks to make a move to restructure its automotive embedded system. This research is looking for indications of the problems they face in the migration process. The results are based on interviews and a focus group with different teams at Volvo Trucks, as well as a code analysis done with *GRASS* which is a static analysis tool developed at Volvo Trucks.

1.2 Research Questions

RQ1 - Are there any common practices that the teams are using while designing how to migrate the source code, and while performing the migration?

This research question aims to find out if any common approaches are used by different teams in regard to migration. If the teams can work in similar ways, the organizational changes could be more aligned. If several teams encounter the same issues, this might be worth looking deeper into for Volvo.

RQ2 - Is it possible to use the GRASS tool to identify structures in the source code that are considered to make migration hard by the interviewees?

GRASS is a static analysis tool built for this thesis. Research question 2 is meant to explore if GRASS could help in visualizing issues in the code. It will also explore if the tool is useful to developers or architects.

RQ3 - Which parts of the physical system (for example ECUs) can influence the migration of the code?

For an automotive system, hardware and software are usually tightly coupled. When there are changes that need to be performed on the hardware, then the software also needs to be changed, and vice versa. This research question is meant to see how hardware and software are coupled to get an idea of how to decouple them.

RQ4 - Do the developers and architects think that the whole or part of the migration process can be automated?

Since restructuring a large automotive system is a big challenge and requires many resources, research question 4 is meant to brainstorm some ideas for automation. Automating the whole or part of the process could help a large organization such as Volvo Trucks to move to a new structure.

1.3 Contributions

This thesis aims to contribute in different aspects. The main goal is to close a research gap, the other goal is for Volvo Trucks and similar companies to gain insights into what affects large software migrations.

Provide information about what happens when migration occurs in a large vehicle company: It is not often a migration of this scale that occurs in a company as large as Volvo Trucks. This research will have the opportunity to see what affects migration first-hand. This will be helpful for future migrations done by Volvo Trucks, and companies of the same size and in similar areas.

Provide what problems the teams are facing: A migration of this size takes time and money to plan and perform. There might be different opinions among the employees on whether the migration should be performed in this way, or even at all. For Volvo Trucks, this research has value in that it explores which issues are arising with the software migration that they are currently undergoing. This research will bring a bigger picture to Volvo Trucks, shedding light on teams' progress and issues with the migration.

2

Background

This chapter introduces the background and theory needed to understand the thesis. It covers the structure of motor vehicles, both from a hardware and a software perspective. It also explains Volvo Trucks' current and future software structure. Furthermore, related works are discussed.

2.1 Motor Vehicle Structure

There are different motor vehicles, but most of them are built in roughly the same way. Cars, trucks, buses, and construction equipment all use a motor to drive themselves forward. Combustion engines, electric motors, and hybrid variants are the most common ones. Newer vehicles also usually have some kind of infotainment system, and these parts need to communicate with each other. In a vehicle, this means that both the hardware needs to be connected, and the software needs to be implemented inside these hardware components.

Electronic Control Units (ECUs) are embedded systems in automotive electronics that control one or more features in a motor vehicle. ECUs take control by reading and sending signals to connected devices. An automotive system requires a real-time environment, which means some response time should be guaranteed within a specified time. The communication system that is commonly used between ECUs and devices is called Controller Area Network (CAN) bus. CAN bus is a communication protocol for real-time systems [4]. It allows different parts of the vehicle to talk to each other without a host computer. The nodes on the bus can send and receive signals. The signals can be transmitted serially onto the bus and can be received by all the nodes. Alternatives to the CAN bus are the LIN bus and ethernet, which can complement the CAN bus.

Figure 2.1 shows an example of a hardware architecture in a vehicle. There are different Master Control Units, or main units, that have different responsibilities, and they are normally connected by a CAN bus. It can be useful to have redundant data buses at times, in this image this is visualized between Powertrain Domain and Chassis and Safety Domain, both of which are safety-relevant. Each main unit has its subcluster, in some cases with lower-speed CAN buses within. The reason for the lower speed within the submodules in the Cabin case is that these modules are normally physically closer to each other, and do not have a critical need for faster communication.

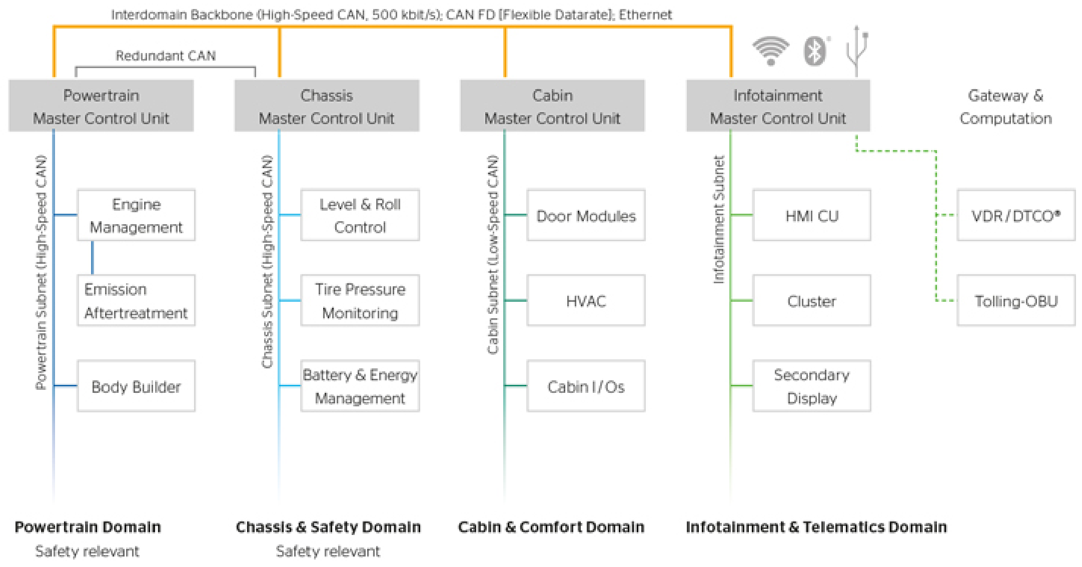

Interdomain Backbone (High-Speed CAN, 500 kbit/s); CAN FD [Flexible Datarate]; Ethernet
Redundant CAN
Powertrain Master Control Unit
Chassis Master Control Unit
Cabin Master Control Unit
Infotainment Master Control Unit
Gateway & Computation
Powertrain Subnet (High-Speed CAN)
Engine Management
Emission Aftertreatment
Body Builder
Chassis Subnet (High-Speed CAN)
Level & Roll Control
Tire Pressure Monitoring
Battery & Energy Management
Cabin Subnet (Low-Speed CAN)
Door Modules
HVAC
Cabin I/Os
Infotainment Subnet
HMI CU
Cluster
Secondary Display
VDR/DTCO®
Tolling-OBU
Powertrain Domain
Safety relevant
Chassis & Safety Domain
Safety relevant
Cabin & Comfort Domain
Infotainment & Telematics Domain

Figure 2.1: Example hardware architecture [5]. There are ECUs in different domains, connected by CAN.

2.1.1 AUTOSAR Architecture

There are various software systems in a vehicle that control and manage different functions. AUTOSAR is a standard created by a worldwide consortium of vehicle manufacturers. It provides a platform to create and establish a standardized software architecture for automotive ECUs.

AUTOSAR introduces the concept of Software Component (SWC) to describe the basic unit of software that needs the AUTOSAR Real-Time Environment (RTE) for execution. A SWC is an independent unit of functionality that can be deployed. A SWC contains a set of c files which are called software modules, they realize the functionality of the SWC [6]. To break down the functionalities into independent SWCs, Volvo Trucks designed Logical Design Components (LDCs) based on high-level descriptive functionalities. LDCs are abstract representations of the SWCs. The communication between LDCs is represented as signals. In the current structure, SWCs contain controllers and handlers. Ideally, the controllers should only deal with logic and send the decisions to handlers. Meanwhile, the handlers should handle the devices based on decisions made by a controller. However, they are not necessarily that strictly separated in the real world.

As vehicles are highly complex and consist of many different hardware and software parts, it is common to outsource some specific parts to suppliers. These suppliers are normally called Original Equipment Manufacturers (OEMs). The parts that OEMs supply to a vehicle company can be seen as black boxes, you can not look inside them and understand all parts.

2.1.1.1 Classic AUTOSAR

The classic AUTOSAR platform was first released in 2005 [7]. It is structured around a layered architecture, see figure 2.2. Classic AUTOSAR builds on microcontrollers, which are a type of ECU, at the lowest level. On top of this, there are three main software layers: application layer, runtime environment, and basic software [8]. The basic software is in turn divided into the services layer, ECU abstraction layer, and Microcontroller abstraction layer. The classic AUTOSAR focuses on component-oriented software structures, with SWCs as independent units. Development with classic AUTOSAR is done in C.

In classic AUTOSAR, the Virtual Functional Bus (VFB) is an essential concept. The VFB is implemented in the runtime environment layer and handles the communication both within and between ECUs. It decouples applications from the infrastructure and supports hardware-independent development [8]. This means that an application does not need any detailed knowledge of lower-level technologies.

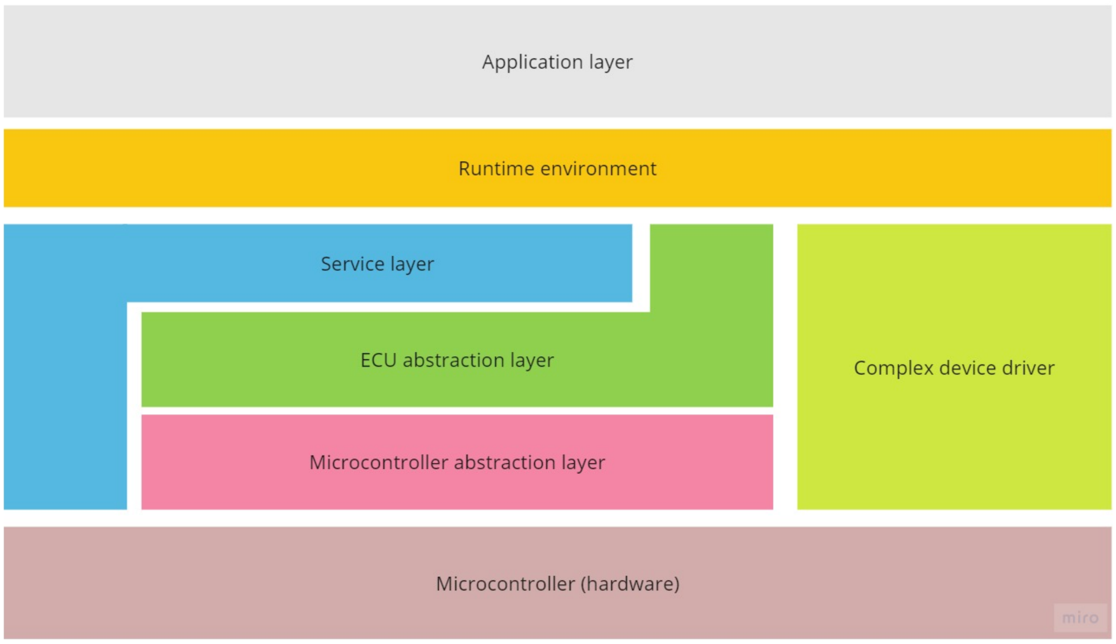


Figure 2.2: Architecture of classic AUTOSAR. The bottom layer contains the physical hardware, the layers then increase in abstraction going up.

2.1.1.2 Adaptive AUTOSAR

AUTOSAR is continually updating its framework. To keep up with technological advancements, they released a new version of AUTOSAR, the adaptive AUTOSAR. Adaptive AUTOSAR was released in 2016 and is continually being updated [9]. The new version of AUTOSAR is built to be a better fit for new problems in the vehicle industry such as autonomous driving, vehicle connectivity, and Car as a Service.

The key principles for adaptive AUTOSAR are that C++ is the preferred language, Service-oriented architecture should be used, and that it has inherent parallelism.

While classic AUTOSAR has a static nature, adaptive AUTOSAR offers dynamic assignments of cores. Adaptive AUTOSAR also makes more use of ethernet which has a high bandwidth [10].

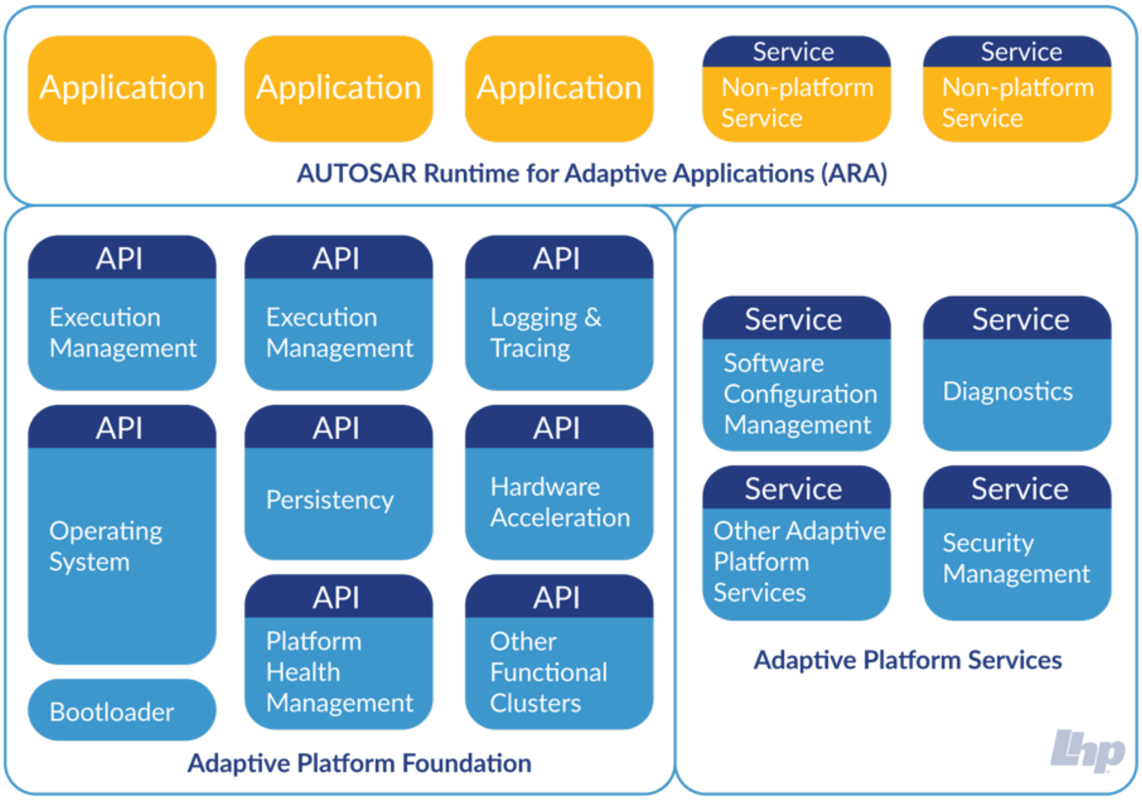


Figure 2.3: Architecture of adaptive AUTOSAR [10]. The adaptive AUTOSAR has a platform foundation where an operating system can be used. It also has a runtime environment and services.

Figure 2.3 shows the architecture for the adaptive AUTOSAR. There is a runtime environment, however, it is smaller compared to the classic version. Adaptive AUTOSAR has room for an operating system, and the idea is that it can contain dockerized applications. The usage of the different AUTOSAR versions will likely be a hybrid of them working together [10]. In the cases where real-time requirements are high, for example, emergency brakes, classic AUTOSAR will likely be used. For more software-heavy applications, such as autonomous driving, the adaptive AUTOSAR will have its place.

2.1.2 Volvo Trucks Current Structure

Following AUTOSAR's standards has made the code base at Volvo Trucks as consisting of several monolithic entities. In the current A1 system, the code has been written with a component-oriented or function-oriented mindset, according to AUTOSAR standards. The code has a layered structure that is designed as a two-layer structure. The design principle is to separate the core logic, which is called the controller, and the handler, which is responsible for handling the physical device.

For example, the controller decides the movement of the wiper and sends the decision to the handler. The handler handles the wiper based on the message from the controller. However, controllers and handlers are not strictly separated in practice. This leads to a kind of spaghetti code structure.

The monolithic entities with high couplings lead to updates taking a long time to implement in the software. In order to make simple changes, shotgun surgery is normally needed [11]. This makes it hard for developers to efficiently update the software and could lead to less maintainable code.

2.1.3 Volvo Trucks Future Structure

The code written according to AUTOSAR standard has resulted in large executable monolith entities. In order to move away from this structure, Volvo Trucks are with the move to an updated A2 system with adaptive AUTOSAR, moving toward a Service Oriented Architecture (SOA). An SOA mindset helps break functionalities down into services and move away from monolithic structures.

Volvo Trucks are changing the software and hardware topologies is another of the main differences that is underway. The idea is to centralize the logic in one central ECU. This central ECU should have a lot of calculating capabilities, and be able to handle a lot of the logic. A big advantage to this centralized design is that it in the future will enable software updates through the cloud. The move to this centralized structure which should utilize adaptive AUTOSAR is internally called *A2 refactoring*.

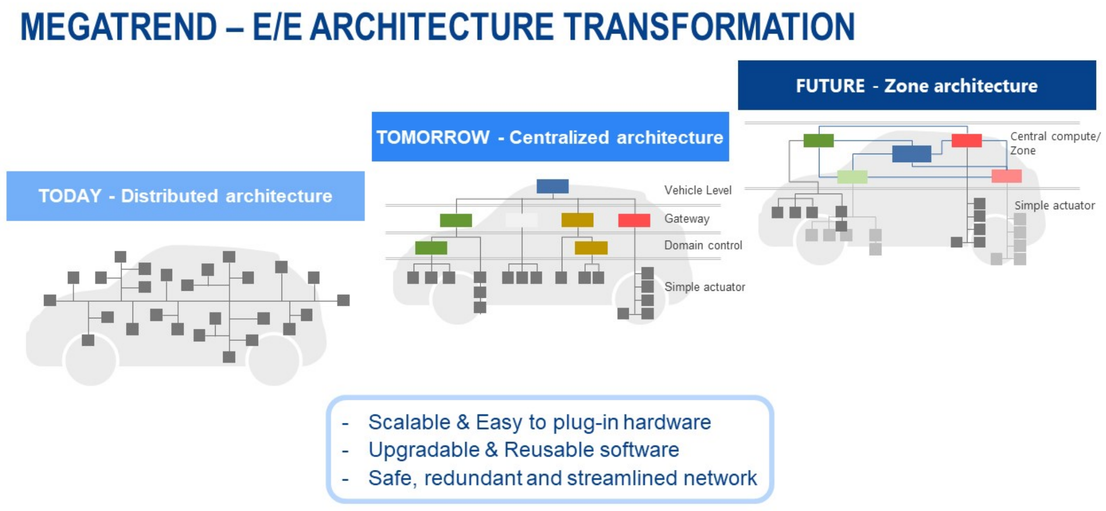

MEGATREND – E/E ARCHITECTURE TRANSFORMATION
FUTURE - Zone architecture
TOMORROW - Centralized architecture
TODAY - Distributed architecture
Central compute/ Zone
Simple actuator
Vehicle Level
Gateway
Domain control
Simple actuator
- Scalable & Easy to plug-in hardware
- Upgradable & Reusable software
- Safe, redundant and streamlined network

Figure 2.4: The trend of E/E architecture [12]. The distributed hardware architecture of today does not support heavy calculations. The architecture that will be used in the near future can be seen as a middle step toward the Zone architecture that will have a lot of computational potential.

The move to a different software structure is a kind of software migration. For the end user, the truck driver, this migration should not be directly noticeable. The functionalities of the truck should stay the same: if you want to push on the gas, you go forward, if you brake with your foot, the truck brakes, etc. However, this kind of large restructuring will have a big effect on software developers. A service-oriented approach will make the source code easier to maintain, and it will make it easier to implement new features. A vision in the vehicle industry is to be able to roll out software updates over the cloud in a short time.

The changes that Volvo Trucks are undergoing are part of a megatrend in the vehicle industry [12]. Figure 2.4 shows how the general trend in the industry looks. As of today, the ECUs are distributed over the vehicles, and connected over the CAN bus. The centralized architecture that Volvo Trucks are moving toward is the "architecture of tomorrow". This architecture will have more computational power to enable more software-heavy features, such as autonomous driving. The idea is that it is a step, and in the future will lead to a Zone architecture.

2.1.4 Current Status

Architects and managers at Volvo Trucks have been thinking about and planning this migration for years. These kinds of discussions can be endless, as the budget and time plan needs to be decided as accurately as possible. As in most large companies, many things need to be done, and the prioritization can be different for different teams within the company. Therefore, some teams have started the migration, while others have not yet begun to plan for it.

For developers at Volvo Trucks, the plan for the migration might not be as clear as it is to architects and managers. Even for architects, it might be hard to see the big picture, especially as there are not many good tools for architecture recovery for embedded systems. Communicating a migration such as this can be difficult, and could lead to uncertainties within teams of developers. The developers and architects need to be on the same level, and have a vision for the big picture.

This case study aims to study the gap between developers and architects. We try to see how the plan for the migration is progressing by interviewing developers and architects from different teams. In order to do this, we also use architecture recovery to bring the big picture of the system to the developers and architects.

2.2 Related Work

Here we discuss some related works that have been done in the areas of AUTOSAR, software refactoring and migration, architecture recovery, and service-oriented architecture.

2.2.1 Software Migration

Salman et al. provided a three-stage methodology for migrating a complex real-time industrial software system [13]. They reviewed some of the existing migration methods. In the paper, their main research question is **How to migrate a complex real-time software from a single-core to a multi-core architecture with maximum software reuse and minimal re-engineering effort?**. The paper provided a methodology for migrating and a review of tools that facilitate the migration process. They found that the software architecture transformation is the main phase in the migration process. A lot of emphasis is put on architecture recovery and evaluating the mechanism of multi-core solutions.

A paper was released about the migration from existing automotive software to AUTOSAR in 2008. In this paper, Kum et al. explained how to construct AUTOSAR applications for already developed software for ECUs [14]. The authors motivated the reasons for moving to AUTOSAR with the rising complexity of automotive software. The paper goes through the concept of AUTOSAR, how to migrate, and provides a case study in the form of the migration of interior light. They concluded that the legacy code should be rearranged according to AUTOSAR structure.

There have been studies regarding software migrations of companies in the past. However, no study has been made with the migration of classic to adaptive AUTOSAR in mind. This is likely since not many companies have made this migration so far.

2.2.2 Refactoring

Refactoring is the act of changing the source code without changing the functionality. Common examples of refactoring are removing duplicate code, renaming methods and variables, and extracting methods to reduce complexities in the code. Refactoring is normally done to increase maintainability, readability, and extensibility.

There has been previous research about how to design as well as how to perform refactoring. Jain and Saha discussed the importance of maintaining large complex software [15]. In the result section of the paper, they concluded that one of the most common reasons to refactor code is to *make code more maintainable*. They also discussed different tools that could be used, as well as the risks and benefits of refactoring. An interesting finding in their paper was some found barriers to not using different tools to help to refactor. The barriers ranged from *Little or No Knowledge About Availability of Tools* to *Company support* and *Languages like Python and CSS have limited support for refactoring tools.*

Vogelsang et al. presented an empirical analysis of feature dependencies in a paper [16]. It showed that the features were highly interdependent in modern vehicles, thus dependency-based refactoring could be an approach to automotive systems. The components that contain many dependencies are implicit communal components in automotive systems. They were considered to be related to technical debt in the paper. Therefore, the paper also proposed an approach to extract dependencies between components based on the data flow to justify their impacts. The results indicate that refactoring on communal components contributed more than other components.

Eliasson et al. performed two case studies on Volvo Car Group and Volvo Group Truck Technology [17]. The research goal was to find out the requirements and challenges of software in the automotive domain. Based on their studies, the paper suggested having two different architectures of the same system. One should be the high-level descriptive architecture and another was the functional architecture which defined the actual plan for daily work. However, they found high-level architecture teams in the two cases had little communication with the rest of the development teams. Thus, the usage of high-level architecture was limited. The paper shows new ways of working were required to adapt to the rapidly growing software system both on the technical and organizational levels.

Refactoring is usually one step that is used in a migration process. Making and keeping the source code maintainable is vital in order to keep the software easy to work with and updated. The reasons for refactoring, which components might be most useful to refactor on, and the challenges of software in the automotive domain have been explored. However, there has been little research on what refactoring approach is used during the migration of software.

2.2.3 Architecture Recovery

Architecture recovery is a technique that can be used in cases where the software architecture of a system is unavailable. This is when architectural information is extracted from the source code. Using architecture recovery can help understand a system's architecture, and can be especially helpful when dealing with legacy systems, or with systems that do not have good documentation.

KTH and Scania collaborated in the 2010s during a project called the ESPRESSO project. Together they created a tool to extract the architecture from a Scania-specific C code base in 2014 [18]. Their paper talks about the weight of understanding the code base as is and the difficulties in knowing the software architecture of an embedded system. The paper outlines their approach to creating a graphical tool for visualizing the architecture.

In order to get an overview of the software at Volvo, it was decided to develop a program for architecture recovery for the Volvo-specific system.

2.2.4 Service-Oriented Architecture

Lee and Wang proposed a method for designing and analyzing automotive software [19]. This paper talked about how the complexity of automotive systems increased because more software functions were required. Centralized software generally decreases evolvability with many connections. Thus, the automotive industry needed to adopt a service-oriented architecture. However, they integrated ECUs based on features without software architecture, which might cause optimality problems. This paper discussed that the deployment of software components with software architecture instead of considering vehicular features as a priority would provide benefits. Since quality attributes are the centre of software architecture, it's beneficial for software design and E/E design. Therefore, they proposed a component-based architecture style to help design high-level software architecture for automotive systems and analyze evolvability, development effort, and dependability. They decided on three types of connections and analyzed the system based on types of software components and connections. However, this architectural style still had problems in identifying real-time characteristics.

Most research related to SOA has been done on web-based applications. The most notable research done on the vehicle industry proposed that a component-based architectural style is a better fit than SOA. This research did not look at the move from one to the other, but rather which is a better fit for the vehicle industry. Our paper will look at the move from a component or function-oriented mindset toward an SOA mindset.

2.2.5 AUTOSAR

AUTOSAR is a framework that has been around since 2002, and there has been some research on how to implement AUTOSAR and the effects of implementing AUTOSAR. One of the most frequent writers on this subject is Dersten from Mälardalen University.

Dersten et al. discussed the effect of moving to Classic AUTOSAR from a business point of view in 2010 [20]. In their results, they argued that the move affects both the system properties and the company functions. Examples they brought up are improved flexibility, reliability, and maintainability as an effect of moving to signal-based CAN on the system properties side. This move they argued would affect the development process on the company functions side and improve efficiency on the business side.

Dersten also researched guidelines for refactoring embedded systems in another paper [21]. Many companies have started to introduce AUTOSAR to their automotive systems without any functionality changes to their end users. The automotive systems are largely distributed and have many inter-connected ECUs, which leads to high complexity. This paper tried to form a guideline for refactoring an embedded system into the AUTOSAR framework to ease the refactoring. The paper proposed some important activities for the guideline. Establishing technical requirements, investigating if the existing system can be expanded or adjusted, and evaluating the effects of non-functional parts, for example, the response time, were three of the most important activities in the result.

Dersten et al. further discussed the information needed to decide on a system refactoring in a paper published in 2012 [22]. Their paper focused on embedded software and systems and gathered information from both System Architects and Managers. They found that the most important information to make a decision about refactoring was related to cost, profit, technical details, supplier information, and requirements.

Nghia Vo et al. conducted a case study about building automotive components within the AUTOSAR environment in 2009 [23]. The study focused on the challenge that the automotive industry was facing, which was the significance of software and software-based functionalities increased rapidly. Their research showed that software complexity was the major reason for the delay and cost overrun. Based on this paper, AUTOSAR is an approach to address the complexity issue since it has different abstraction levels of the overall system.

All of these studies have focused on companies using Classic AUTOSAR. There has been work regarding this move since its introduction in 2006. The first version of the adaptive AUTOSAR platform was released in 2017, with updates in the following years. There have been studies made regarding different aspects of adaptive AUTOSAR, for example, the progress of the Adaptive AUTOSAR platform, comparisons to other platforms, and regarding the communication methodology [24]–[26]. To the best of our knowledge, there have been no or few studies made regarding the move from classic AUTOSAR toward the adaptive AUTOSAR platform.

2.2.6 Summary

Software refactoring and migration happen every so often. Looking at the methodology of either has already been done in different ways [13]–[15]. The research on software migration in embedded systems is however not comprehensive. With the need for more software-heavy vehicles, and a new adaptive AUTOSAR framework, we see a need for research. The research gap we will fill is about the migration of a large software system in an automotive setting, toward a centralized structure.

This study uses interviews and a focus group to gather qualitative data on what happens during a migration of this scale. We combine the qualitative data with quantitative data gathered via GRASS, a static analysis tool.

3
Methodology

To answer the research questions, this project used a case study approach. A case study is an in-depth investigation, normally using qualitative research methods, of a single phenomenon [27]. The phenomenon, in this case, is the software migration currently undergoing at Volvo Trucks.

This chapter presents the methods that were used during the case study. It goes through the interviews, the focus group, and the code analysis. For the interviews, how the data collection was conducted, the interview design, and the data analysis is gone through. In the focus group section, we go through the discussion points we included, as well as the motivation. In the code analysis section, we go through how we used the GRASS tool.

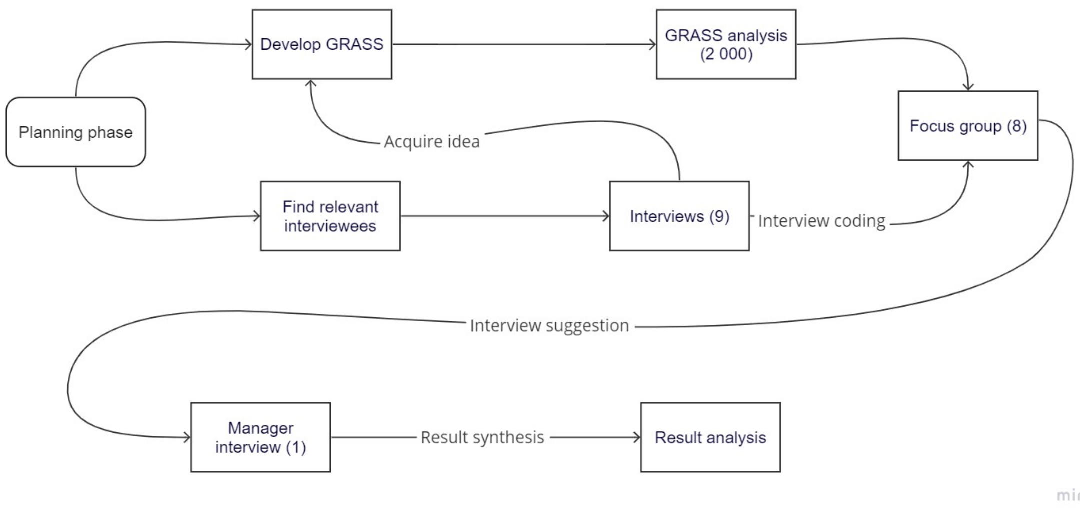


Figure 3.1: Method process during the thesis. There were nine interviews done in the first round, eight people (excluding us) joined the focus group, and we later had one additional interview. There were around 2000 components analyzed using GRASS.

Figure 3.1 shows a rough outline of the whole process. We found interviewees and developed GRASS in parallel. During the interviews, we found an idea for GRASS, which was implemented. GRASS was used to analyze around 2000 components from Volvo Trucks codebase. After the first nine interviews and code analysis were done, we planned the focus group. In the focus group, we included elements found in interviews and the code. Based on the results of the focus group, there was a suggestion to include more perspectives on the migration. An additional interview with a manager was therefore planned. The results are based on these previous steps.

3.1 Interviews

To gather qualitative data, we conducted interviews with different teams. The interviews were mainly designed to answer **RQ1**, **RQ3** and **RQ4**, but also touched **RQ2**.

3.1.1 Data Collection

Participants in this study were recruited via internal contacts and recommendations within Volvo Trucks. We initially aimed to recruit developers from up to five different teams. The results of the first interviews with developers indicated that it might be interesting to include architects in the study as well.

A requirement to participate as an interviewee was that they were involved in the movement toward the new architecture, either working on or designing the architecture or functionalities. In addition, the interviewee had to be able to give us time to perform the interview. Participants had an interview that took place either at the Volvo Trucks office or remotely in cases where a participant was sick or abroad. The interview took up to one hour and each interviewee was invited to partake in a focus group at the end of the study where different topics were discussed.

We had nine different interviewees during the first round of interviews, and one additional interview during late spring. The interviewees, their position at the company, and which team they belong to are listed in table 3.1. In total, we have conducted ten interviews.

ID	Position	Team
A	Developer	Development Team 1
B	Developer	Development Team 2
C	Developer	Development Team 3
D	Developer	Development Team 4
E	Architect	Platform Architecture Team
F	Architect	Platform Architecture Team
G	Project Manager	Development Team 5
H	Architect	Development Team 6
I	Architect	Platform Architecture Team
J	Group Manager	Electrical Architecture Management Group

Table 3.1: List of interviewees.

3.1.2 Interview Design and Questions

We interviewed ten people from different teams. The interviews were semi-structured. Four open questions were the basis for each interview, and then the interviews delved deeper into areas that were relevant for that specific team and person. Each interview took around one hour.

In the interview questions, we call the software migration *A2 refactoring*. This is because the migration toward the new system A2 is called *refactoring* internally at Volvo Trucks.

The interviewees included developers, architects, and a project manager from the development teams, as well as platform architects from the platform architecture team and one manager from the architecture management group. In order to get answers from different perspectives, we adjusted the questions for the development teams, the platform architecture team, and the management team. For the development teams, the main questions were as follows:

1. How do you approach refactoring for A2?
2. Have you encountered or do you think you will encounter any problems during the refactoring process?
3. Is the hardware architecture influencing how you are planning the refactoring?
4. Do you believe parts of the refactoring process can be automated?

For the platform architecture team and the final interview with a manager, questions 3 & 4 stayed the same but question 1 & 2 changed as follows:

1. What kind of workflow for the architecture team and development teams could help this refactoring process?
2. What problems do you think the development teams will encounter?

Table 3.2 shows how we map the interview questions to our research questions. The table also includes our motivation for the interview question.

The other questions could differ based on the answers from the interviewees. All the interviews were recorded and transcribed in order to conduct a reliable analysis.

3.1.3 Data Analysis

To understand what aspects are most important and see patterns, we used card sorting to analyze our interview data. Card sorting is a method to organize information in order to find the best matches [28]. This method is mainly used for UX research but is also efficient for categorizing results from groups of interviews. Generally, there are three different ways to run card sorting: open, closed, and hybrid. Closed card sorting is when you have some pre-defined categories that you sort the cards into. Open card sorting is when you instead do not have any pre-defined categories,

Research question	Interview question	Motivation
Are there any common practices that the teams are using while designing how to migrate the source code, and while performing the migration?	How do you approach refactoring for the new architecture?	This question was to examine how different teams approach the new architecture and try to find if any general practices can be used for most teams.
Is it possible to use the GRASS tool to identify structures in the source code that are considered to make migration hard by the interviewees?	Have you encountered or do you think you will encounter any problems during the refactoring process?	This question was trying to understand how the teams consider the problems while they take this big move. We wanted to connect their problems or uncertainties to our code analysis in order to get a better idea of the code analysis.
Which parts of the physical system (for example ECUs) can influence the migration of the code?	Is the hardware architecture influencing how you are planning the refactoring?	The idea was to explore the specific problems of changing an embedded system. Architects have to consider the hardware while designing the software architecture. However, the developers might not know how the hardware affects their migration efforts. So, there is a gap between the architects and developers. This question was trying to see how this affects the migration.
Do the developers and architects think that the whole or part of the migration process can be automated?	Do you believe parts of the refactoring process can be automated?	This question was designed to brainstorm if there could be some automation that can help the migration in general.

Table 3.2: Mapping of the research questions, interview questions, and a motivation.

but make them up as you go. We used a hybrid version of card sorting to categorize the answers and we did it digitally with a tool called Miro. We picked out some categories that were related to our research questions to start with. The categories we used at the start were the following:

- Approach
- Hardware influence
- Challenges

- Automation

After the transcription of interviews, we categorised each interview by itself. We then extracted interesting quotes into various cards from the interviews and put them in the connected categories. After that, we had a better understanding of the interview data. Then, based on what we noticed in the first round of card sorting, we added categories that were mentioned by several different interviewees. We coloured the cards by the interviewee and moved them into the new categories. The categories for the second round were as follows:

- Approach
- Team communication & Team dependency
- Hardware influence
- Automation
- Requirements & Ownership
- AUTOSAR
- Benefits
- Challenges

After individually categorising the interview data, we combined them in an Insights Extraction step. We clustered the data gathered, relating them to the research questions.

3.2 Focus Group

A focus group is a small group selected from a wider population, whose purpose is to inform opinions on a particular subject or area [29]. There were two main purposes to hold this focus group. First, we wanted to discuss some different opinions from interviews to validate our data. The second purpose was to inspire us with ideas in order to drive our research.

The focus group was planned after all nine initial interviews had been conducted. All the interviewees were invited to this focus group to discuss their different opinions and the results of the interviews. The discussion was planned after the last interview had been conducted. We planned to hold the discussion for two hours, where the first hour had the most crucial topics for the research. The interviewees who had the time were welcome to discuss for the second hour. We booked a conference room so that participants who might not be in Gothenburg could attend since one interviewee was based abroad.

During the focus group, we introduced the themes of the thesis, and the work we had done so far, but the main focus was on the discussions. The two hours were recorded and transcribed for further analysis.

Table 3.3 shows the attendees of the focus group. During the focus group, Developer C and Architect H did not come. Furthermore, Architect F and Project Manager G left after the first discussion round. In addition to the interviewees, another member of the Platform Architecture Team who had been helping us with contacting interviewees and organizing the focus group, joined. The manager who was interviewed after the focus group was invited to the focus group. Unfortunately, the manager, one developer, and one architect could not attend due to conflicting meetings.

ID	Position	Team	Attendance
A	Developer	Development Team 1	Joined
B	Developer	Development Team 2	Joined
C	Developer	Development Team 3	Did not join
D	Developer	Development Team 4	Joined
E	Architect	Platform Architecture Team	Joined
F	Architect	Platform Architecture Team	Joined first session
G	Project Manager	Development Team 5	Joined first session
H	Architect	Development Team 6	Did not join
I	Architect	Platform Architecture Team	Joined
J	Group Manager	Electrical Architecture Management	Did not join
K	Organizer	Platform Architecture Team	Joined

Table 3.3: List of focus group attendees.

3.2.1 Discussion Points

The different discussion points were influenced by both the research questions and the results of the interviews. We had questions about the graphs and metrics created with GRASS, and questions regarding approaches, organizational aspects, the benefits and challenges, automation, and time for the participants to ask questions.

To validate whether the graphs and metrics could be useful, we showed data and images and discussed what they show and how they can be used. On the theme of GRASS, the following questions were discussed:

- Can we say something about how hard or easy these files would be to migrate based on size (lines of code)?
- Can we say something about how hard or easy these files would be to migrate based on their dependencies?
- Are these graphs useful to you, if so, how?
- Are these metrics (size and dependencies) useful to you, if so, how?
- What else might be useful?

On the themes of the approaches to migration and/or refactoring, the following questions were asked:

- What approach did you take in the process, and why?
- Could every team have the same approach?
- Other ideas for approaches?

During the interviews, we found that there might be organizational changes that could make the migration easier. To validate this idea we discussed the following points on the organizational scale:

- Are there any organizational changes that might make the process better for you?
- What are the responsibilities of developers and architects respectively?
- Are there ways to make this better?

There were some different ideas for automation during the interviews. These questions were discussed regarding automation:

- Have you applied any automation in the process, if so, how?
- Do you think more automation is viable?
- Is it generalizable?

We also wanted to see if we could find a consensus on what challenges and benefits developers and architects see. These questions were discussed in this area:

- What benefits do you see with A2?
- What challenges do you see with A2?

In order to catch any questions that participants wanted to discuss, we included time in the end for open discussion on any topic.

3.3 Code Analysis with GRASS

During the summer of 2022, a tool called GRASS (stands for GRAnd Survey of Software) started being developed. GRASS was first meant to be a static analysis tool, with counts for different aspects of C code, such as the number of if-, for-statements, and the total number of functions in a file. During the autumn of 2022, GRASS was extended by us during an internship course to include the number of in- and out-signals in the code and some graphs that visualize these. The tool was further developed during the thesis work in spring to provide more insightful graphs. Based on the interviews, a metric related to the size of the files was added to GRASS.

3.3.1 Reasons for using GRASS

During the internship course done during autumn 2022, we conducted a few informal interviews where some interviewees mentioned couplings between components as an issue. Based on these interviews, we learned that dependencies were one potential metric that could be interesting to look into. The interviewees told us that Volvo Trucks does not have a good tool to visualize the whole software architecture and check the signal flow between components. That is the main reason for the extension of GRASS, to visualize the dependencies between components through graphs. These graphs were used to calculate metrics on the architectural level, to see if they could indicate the complexity of the migration and how to decouple the structure.

3.3.2 Technical Details of GRASS

The static analysis tool was built with Python, around the Volvo Trucks system. Below are the details of how GRASS was used in the thesis.

3.3.2.1 data_scraper

A function in GRASS called data_scraper was developed to capture the design information of the ECUs. This function collects data from a program called SE-tool. SE-tool is a software for documentation where developers working at Volvo insert data about signals. The data is from the hardware perspective and contains the LDCs based on ECUs and the signals between the LDCs. This data is extracted through an API call and contains information on which signals go in and out of specific LDCs.

Based on the data from data_scraper, we created graphs to provide better visualization of the structure. The graphs contain information about LDCs and signals as well as the connections between LDCs.

3.3.2.2 Graphs and Metrics

GRASS can construct graphs based on LDCs or signals. LDCs are represented as nodes and signals represent edges. The graphs are constructed using NetworkX's MultiDiGraph [30]. These graphs can contain several directed edges between nodes. The graph containing all LDCs and signals shows how interdependent and complicated the system is. The graphs reveal the dependencies between LDCs and between ECUs.

We created a few different graphs based on the data. One based on LDCs, to visualize the smallest part. These graphs contain information on how many signals are received and sent out by an individual LDC. Early in the process, we tried to create graphs that track the flow of signals. This proved to be impossible, as signal names are unique, and there is no way to follow the information. We also created graphs for dependencies within individual ECUs. The final graphs we created were based on the whole system. In these graphs, we coloured the nodes by ECUs, and we grouped the nodes by colour. This graph visualizes the dependencies between ECUs.

Based on these graphs, we calculate some metrics. Fan-in and fan-out are calculated by counting the number of edges that are incoming and outgoing from an LDC respectively.

Another calculation is the signals communicated inside the ECU or outside the ECU. The reasoning behind this calculation is that different teams take responsibility for different ECUs and it might make migrating harder when there is a lot of data communication *between* ECUs, especially when these ECUs are controlled by different teams. It could be even harder for teams responsible for the ECUs deeply connected to ECUs coded by suppliers.

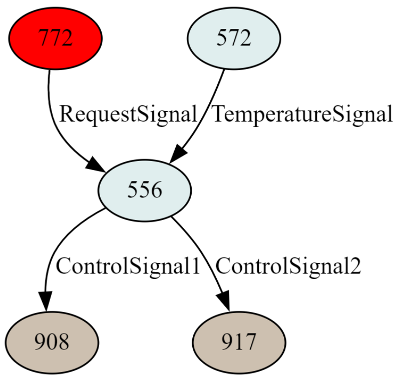


Figure 3.2: Example graph based on the LDC ECU3/Controller2 in table 3.4. This graph visualizes two incoming and two outgoing signals for node 556. The nodes are colored by ECU, the red color indicates that it comes from a supplier.

ID	LDC	Fan-out	Fan-in	Fan-out red	Fan-in red
556	ECU3/Controller2	2	2	0	1

Table 3.4: Metrics calculated from the graph in figure 3.2. The node, or LDC, with ID 556 has two incoming and two outgoing signals. The 'red' indicates that the LDC is from a supplier.

Examples of the metrics calculated can be seen in table 3.4. As not to leak any confidential information, the data presented in the example tables and graphs from GRASS is faked. The `ECU3/Controller2` is calculated from the graph in figure 3.2. Fan-in is calculated by adding the different signals going into that LDC, and fan-out is by adding the signals going out. This specific example had one signal from an ECU that we do not have the code for since it comes from a supplier, which is signalled by the red colour.

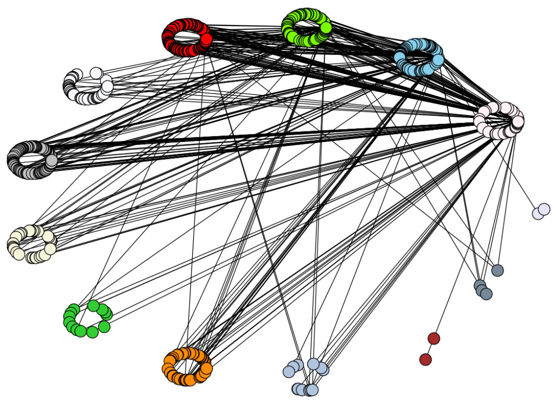


Figure 3.3: Illustration graph of dependencies between ECUs. Each node is an LDC. The nodes are grouped by ECU.

Figure 3.3 shows an illustration of dependencies between ECUs. Each colour represents a different ECU, the red colour indicates supplier ECUs that contain code that is not written at Volvo Trucks. This graph illustrates the dependencies between different ECUs, and fan-in and fan-out were calculated on this level as well.

4
Results

In this section, we go through the results in relation to the research questions. We relate what participants said in the interviews and the focus group to the research questions. For research question 2 we were inspired by pre-study and interviews to develop GRASS.

When we discuss the embedded systems, A1 is the old system, and A2 is the new one. We kept the quotes as close to the transcribed version as possible. The names of components and details are removed, to ensure not leaking confidential information. Some quotes repeated the same word, in these cases, we removed the duplicate words to ease the reading experience.

4.1 RQ1

Are there any common practices that the teams are using while designing how to migrate the source code, and while performing the migration?

4.1.1 Interview Results

This question was mainly informed by the developers. Some of the architects had ideas of how this could be done as well. Unfortunately, the interviewees were not affirmative in that there could be any common practice. There were various approaches when the teams perform this migration. Most interviewees thought that having a common approach for all teams was hard. The reason was that they were facing different problems and that they had not been aligning their work between teams. Another reason was that the final version of A2 was not settled, the requirements were still changing. The interviewees were not sure about what hardware communication network they would use in the end, and which parts should be moved to adaptive AUTOSAR. The system they were going toward included moving some functionalities to the centralized ECU, but there were still discussions about the rest of the parts. Therefore it is hard to provide an approach that can be applied by the teams. Additionally, they had time constraints and a lack of human resources.

Diverse progress of migration activities

Developer A told us that they are still in the concept stage and that they are rewriting parts from scratch. Product Owner G on the other hand wants to reuse as many parts as possible. Developer D is moving their application between ECUs. Developer C is not performing the migration for the sake of the migration but is focusing on refactoring to create a good code structure in general. Developer C's team is also planning on writing their applications from scratch. The teams were in different stages of the migration.

> "We are not moving any part from A1 to A2. But we are doing a fresh start" - Developer A

> "So I take the application, it's a huge monolith, run in a [ECU1] now, it's a ECU, [ECU1], and then I take everything, and I try to put it in the [Central ECU]. " - Developer D

Varying approach to using AUTOSAR

The approach to using adaptive AUTOSAR was varying between the teams. The interviewees could see the benefits of moving to adaptive but saw some issues in the implementation. Developer C chose to write a wrapper so that a function could run in either classic or adaptive AUTOSAR. The communication within an ECU plays a factor in how architect G is planning the use of adaptive. In the migration process, both classic and adaptive AUTOSAR were used, thus the architects had to consider the communication between the two platforms, which would be performed within an ECU.

> "So the wrapper can be made for function running in classic AUTOSAR, and some sort of the wrapper, when you can have a wrapper to make the same function run in adaptive" - Developer C

> "If we have the decision maker in the classic core, then we plan to implement another component in the adaptive core. Just to communicate to any other adaptive SWCs. And talk to Linux." - Architect G

When discussing the progress of the teams, the architects from the platform architecture team did not know exactly how the individual teams were approaching the migration. Architect I told us he does not have direct contact with a lot of the teams, and Architect E thought that some teams might not even know what the migration is about.

> "I can not tell how they are approaching it unfortunately. It is so different. Some doesn't even know what it is about" - Architect E

Unclear requirements from architects

Some architects and developers felt that the communication between them sometimes caused misunderstandings. This could be a result of unclear requirements from architects.

Architect F told us the lower-level requirements might be expressed differently by different architects. F pointed out that this could cause confusion and misinterpretations within the development teams. Architect I felt the same and talked about how the translation of high-level requirements to team level was not very clear. They thought there is a gap between architects and developers which seemed to be the same feeling as that of Developer A.

> "One major gap that we think is that it looks like it's well defined on the high architecture level, but when it comes to the lower level on the execution part, there are still a lot of gaps between. " Developer A

Ideas of improving communication

Developers in certain teams might be too busy with maintaining the current product. Architect F thought that it could affect the communication between architects and developers if the developers did not understand the high-level requirements. Architect I thought it would be easier for developers to understand how to perform the migration if the architects could provide some kind of support or example instead of requirements.

> "Because the most, I guess the most reasonable thing to do is to provide some of the support for refactoring as library and supporting code or something like that." - Architect I

Time constraints

Several interviewees told us that time is a limiting factor in the migration process. Both developers and architects seemed to think that this is a big issue. Many other things are happening at large companies while they change systems and architectures. The mission for each team is different, which means that the teams cannot spend the same amount of time on the architecture migration.

> "People are busy working on their own stuff. They don't really have time. And it's not that high a priority" - Developer D

During the later interview with the manager, we learned that he also saw the lack of time and resources as a problem, like other interviewees. However, as a manager, he felt that the problem was more about the management. He told us that time is always a problem, but they have to figure out how to go in a direction within the limited time and resources. As a company, they have to keep delivering new features to meet customers' needs since they do not want to lose market shares.

> "There's always a pressure from up then, we need to deliver these new trucks, these new features it must become fancier and nicer." - Group Manager J

> "We cannot deliver any new features for a year or two. And then we will lose market shares because the other brands they start, they deliver all the time." - Group Manager J

Need to align work

Some interviewees thought that all teams need to start the migration to align the work. Developer A told us that it would help to find out what dependencies the teams have. She also said if at least all the teams can start this migration so that they can align their work.

> "I think everyone has to start to work toward A2, each and every application team, or each and every development team will have to start to work toward A2, and only then we will get a fair idea of what dependency do we have on who" - Developer A

Architect E felt that it is hard for the teams to perform the migration if they cannot align their work. Architect F also said that it was hard to have a general model for all the teams without planning it with all of the individual teams. If the teams were aligned, they could share their experiences and make quicker progress. It would also help architects develop guidelines and templates for teams to use.

> "So it's hard for a central team to come up with a more detailed model. You need to do that together with teams." - Architect F

Challenges to aligning work

The teams are facing different problems, so it is difficult for them to align their work. The interviewees talked about how the teams prioritised different things. Developer A told us the teams, therefore, planned the migration at different points in time. Most teams have many dependencies on both suppliers and other teams, so they needed to get information from them to continue their work. Both developers and architects thought that they have trouble aligning their work.

> "My team is just one part, but the other team. Although that's totally different, so we are not aligned." - Developer C

Need to change the mindset

Another organizational point that was discussed, was the mindset of developers at Volvo. Architect E told us that some people might simply be more comfortable with maintaining things rather than developing new skills and migrating to a new platform. Architect I thought that a lot of people at Volvo are disconnected from the code. They pointed out that Volvo is not a software company, Architect E also called Volvo a "maintenance company". The architects shared the same feeling that people inside the company need to change their mindset.

> "People. Mindset. It's all mindset. We need to change the mindset." - Architect H

Decision against adapting the organization

Some interviewees told us that an idea could be to split the organization in two. The idea had roots in that it could enable some developers to focus solely on the migration and the new product, and others could focus on the current product. Developer C mentioned that they had heard about another vehicle company that had made a similar split. C felt that such a split could potentially speed up the development process since developers would be able to focus on one thing rather than several.

> "if you want to speed up, yeah, a split organization, put the right people in that one, give all the resources that they can use and yeah make it happen" - Developer C

Architect F told us that this had been on the table, but managers had quickly decided against it because of budget issues. Architect E told us that they had proposed to rewrite the whole system from scratch, but that suggestion had not been chosen in the end.

> "That's why I think, some of us here, have proposed that we should start from scratch, from the beginning." - Architect E

4.1.2 Focus Group Results

The developers talked about their approach in the focus group. Some of them applied similar ways to migrate their software toward A2, some used very different approaches.

Lack of best practices

The developers thought it was hard to say which approach would be the best since each team only tried one approach and had to trust their guts. They also discussed that there could be a common practice that works for all the teams but they needed to try that out to know if that would work. One possible practice that was discussed was to move the existing stuff to A2 instead of refactoring or rewriting.

> "It's hard to say what would be if it would be better in another way. I only try this way. It's just so you need to rely on your engineering intuition." - Developer D

However, during the later interview with the manager, the manager didn't support writing wrappers around the existing code. He thought that would mean using the same old thing, but he expected developers to make other changes, such as refactoring the code.

> "But I think most people try to reuse or rest reshuffle what they have. But I feel also that is a bit deceptive. That means that you just put new clothes on something was old that we wanted to change from the beginning. So the old thing is still there, and then we don't get the benefit that we wanted. In the first place." - Group Manager J

Changing plan

One architect said it was not completely decided how the new architecture, A2, would look like. This uncertainty makes it hard to talk about a common practice at this stage. Both architects and developers were not sure about how to solve some specific problems with the new platform, so no one knew what the final version would be. There was a discussion during the focus group about what the A2 entails. Some attendees thought A2 at least meant to have a more centralized structure. Others pointed out the new AUTOSAR version as the minimum. It was revealed that there is a backup plan, which is to continue with the A1 platform. In that case, the migration is postponed until further notice. This information came as a surprise to the developers in the focus group.

> "It's not known today what the term A2 stands for." - Architect E

> "At least have a [Centralized ECU]" - Developer D

Challenges to aligning work

During the focus group, we learned about more challenges to the alignment of work. The timeline also makes it difficult to ask the teams to follow one specific workflow. This is because the teams are taking other requirements for the current architecture too.

> "Yeah, their timeline didn't really match what we had and. It's sort of forced a lot of people into one direction and like then all of a sudden you need to maintain those if you, if you, you know, three workflows, I don't know needs. I think it's quite hard I guess" - Developer D

Another topic brought up in the focus group was the naming conventions. During the presentation of metrics in the focus group, Architect E pointed out that the naming of a file was unclear to them. The naming of this component was discussed at some length. They also discussed whether it is common to perform certain actions, a discussion that was prompted by another name of a file.

> "I think the naming is strange, like you shouldn't have [name of file] in the [controller1]." - Architect E

When discussing the dependencies, the names of components were also a discussion point. It was also a surprise to an architect that a component had a certain dependency.

> "I'm surprised that you have fan-in to that one actually, but that's another one" - Architect E

Need to change the mindset

In the focus group, the need for changes in mindset that was found in the interviews was confirmed. It was discussed that it is more comfortable to continue using what you already know. There was an idea that it might be beneficial to have the developers who are interested in the migration working with it. The move from classic to adaptive AUTOSAR was also discussed in this aspect, it is considered *safe* to continue working with classic AUTOSAR.

> "Okay with A2, then you can have those that are interested to actually do some change." - Organizer K

Service-oriented thinking

It was also pointed out by Architect E that the new service-oriented thinking is new to Volvo. Developer D agreed and pointed out that people did not seem to want to absorb the new ways of thinking and working. During our previous interview, architect E also pointed out that Volvo Trucks does not focus very much on continuous learning. This could prove a challenge when moving toward the new architecture with new technologies.

> "Challenge No.1, what is object service oriented for a control engineer? How can you control engineering in a service-oriented world? Is not known to anyone. It's an unknown world, challenge No.2." - Architect E

The manager also told us that the mindset is important in the migration process. He thought that they had to change their mindset to think more service-oriented so that they can talk about the software changes toward SOA easily.

> "Going from a flow-oriented function-oriented to some kind more object-oriented, service-oriented kind of thinking where we. Where we focus on the information rather than on the action." - Group Manager J

4.1.3 Summary

From the interviews, we learned that there were different approaches for the teams. Some focus on reusing as much as possible, and some are rewriting the code from scratch. The use of adaptive AUTOSAR is also varying, some are not even considering moving to adaptive. The architects have some suggestions for how the development teams could approach the migration. Another idea was to minimize the gap in communication between developers and architects, and for architects to provide more support for developers. Some challenges that were found during the interviews were the lack of time to work on the migration, the problem of aligning the work across teams and the mindset at Volvo. One idea to perhaps solve some problems was to split the organization in two so that developers could focus on either maintaining the A1 or working on the A2.

During the focus group, we learned that the plan for A2 is not set in stone. The lack of alignment across teams was also discussed. An idea from the focus group was that it is considered safer to keep the software in its current state rather than refactor or migrate it. The new service-oriented thinking was an example of a change of mindset that need to happen.

4.2 RQ2

Is it possible to use the GRASS tool to identify structures in the source code that are considered to make migration hard by the interviewees?

We used and improved the GRASS tool based on interview ideas. The results of GRASS were then discussed in the focus group to answer RQ2. The answer to this research question was that it is possible, but the results from GRASS now were not that inspiring to the developers. During the focus group, they discussed some ideas of GRASS which might make GRASS more helpful in practice.

4.2.1 Interview Results

One thing we learned from informal interviews during the autumn, which was confirmed during the interviews for the thesis, was that dependencies were a point of interest. Both architects and developers talked about how dependencies affect their plan for this migration. Another thing mentioned in the interviews was the size of Software Components(SWCs).

Dependencies

The problem of dependencies was mentioned in several interviews. Both that it can be hard when having many dependencies, and also that a single hard dependency can be difficult. Developer A mentioned there was a specific interface that two of the interviewees were waiting for, which was produced by a supplier. Since the teams did not have this interface, they had to wait until it is done which they expected would take at least a few weeks. This was the strongest example of a hard dependency we found. On the other hand, many smaller dependencies between teams also affected the plans of migration for most teams.

> "Because we don't have that [supplier interface] for the signals we need already, we cannot move further." - Developer A

An architect told us that there has been no architecture work that covers the whole system. When no one has a clear view of the current system, it can lead to issues. Architect H also mentioned that the dependencies were changing since the architectural plan had not reached a final version. This made it even harder for the teams to handle the dependencies.

> "It's not fully clear because I have never seen any architecture work that covers the whole system." - Architect E

> "But then the dependencies are the only problems. Dependencies are static even today." - Architect H

Size of SWCs

Another thing mentioned by Developer D we found could be interesting was the size of the files. Developer D thought that the size of software components had an impact on how they handled this migration. Some software components just contain a large c file to realize the functionalities, which can make it hard to decipher.

> "Nothing says, the size of a component is sort of undefined, and sometimes it can be really large, which would be a blob basically. "
> "I think the size of a software component sort of pushes you into the blob thing. Autosar is a C-based framework. It's also very kind of blob-y and shotgun surgery in a kind of sense. Not everyone likes to sort of separate stuff into modules and they just do one large C file."
> - Developer D

4.2.2 Code Results

We developed GRASS to show the couplings of the components and count the LOC. GRASS provides graphs to visualize the dependencies in the whole architecture, and we used some metrics to get clearer indications. The tool also calculates the lines of code.

Graphs

Using the graph function of GRASS, we created graphs representing the connections between LDCs. In figure 4.1 we can see two different versions of example graphs, containing the same information. These graphs provide a better visualization of the system.

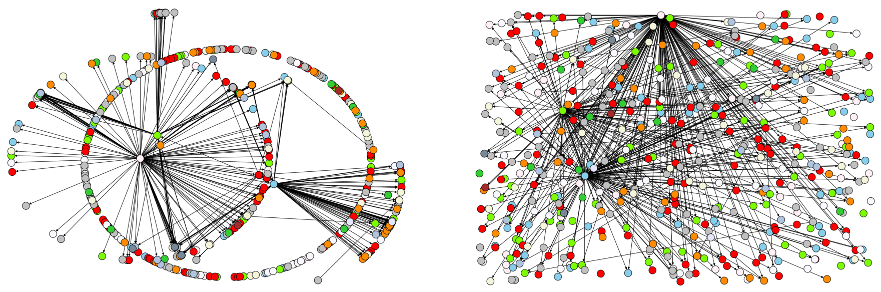


Figure 4.1: Illustration of graphs with kamada-kawai and random layout respectively. Kamada-kawai is an algorithm for creating graphs.

Coupling / dependencies

The code analysis provided some interesting results. In the graphs and metrics, see figure 4.1 and table 3.4 we can see that there are many components with a lot of dependencies. The components with high coupling might make it harder for developers to perform a migration of the software. Table 4.1 shows some examples of the metrics we used to see the dependencies between the components. Fan-in means the signals were received by the components and Fan-out means the signals were sent to other components. The supplier components are not under development inside Volvo, therefore we calculated these dependencies separately. There were large differences in different SWCs, the ones with the most dependencies had a few hundred, while others have zero or a few. The graphs that showed the dependencies of the whole system circulated Volvo Trucks and were used by architects to show the complexity of the system.

LDC	Fan-out	Fan-in	Fan-out to supplier	Fan-in from supplier
ECU3/Service	300	10	200	5
ECU3/Handler	1	0	0	0
ECU2/Controller	200	200	100	100

Table 4.1: Example of dependencies that were calculated. Some LDCs had hundreds of dependencies, others barely any.

Size of SWCs

We also measured the sizes of SWCs by calculating the LOC of the files inside SWCs using GRASS. This metric provides a sense of how large the files are. Many of the larger files were generated files, which had been created using SIMULINK, TARGETLINK, or MATLAB. These generated files were mainly going to be regenerated during the migration. We also found files with only a few LOCs. Table 4.2 gives some examples of the LOC analysis of the files. The files with the most LOC had thousands of LOC, while the smallest had around 30 LOC.

Directory	File	Lines of code
dir1	file1.c	11000
dir2	file2.c	10000
dir3	file3.c	123
dir4	file4.c	30

Table 4.2: Example of file sizes that were calculated. There were big variations in the sizes of files.

4.2.3 Focus Group Results

We showed some results from the code analysis in the focus group and discussed if the results of GRASS could be helpful for the migration process.

Dependencies

The attendees all agreed that the dependencies were one of the biggest issues, but most of them did not feel that the metrics and graphs could help them perform the migration. Some people thought the results could give a sense of how high coupling their components have. Someone said that the metrics and graphs could give a feeling of the complexity and a little information about whom you should talk to, but it is hard to indicate what you should do based on the numbers.

> "If you have a lot of dependencies on others and you don't even know how they, how do they want to do it like in the next generation platform.

> But you need to ask them and you need to ask around like I don't know, 100 different teams." - Developer D

The main problem discussed was that even though they felt that they got the awareness of the coupling of the architecture from GRASS, they still did not have the resources to fix that. It was also discussed that even though the complexity might look bad based on the graph, no one would know how a *good* architecture would look in terms of dependencies.

Usage of GRASS

One attendee felt that if there are no good examples to compare the results with, they cannot get a feeling of how they should handle this. Organizer K came up with the idea that the graphs and metrics could be used over time to monitor the complexity of changes.

> "I mean, it looks bad, but what would good look like?" - Organizer K

> "You could use them over time though. If you have this and then you do something then you can compare" - Organizer K

An architect noted that it is interesting that there can be different interpretations of the complexity of the system based on the graphs. This indicates that the choice of algorithm to place the nodes affect on the person who looks at the graph. Figure 4.1 shows how the same data can result in different graphs which can be interpreted in various ways.

> "I think it's interesting that depending on how you draw them you probably make a different interpretation of the complexity of the system, because you have quite a few variations. [...] It's a different view when you have them ECU-wise I guess allocational, but then it doesn't look that bad anymore. Maybe you make a different interpretation of the state of the situation." - Architect F

Ideas for GRASS

Some attendees discussed that it could be more interesting if there were other metrics combined. It was agreed upon that the most time-consuming thing for the development teams is communication with other teams.

> "One thing that is key for all this fan in and fan out is that if there is, there are engineers that work with those entities, then normally need to collaborate with humans in the other ones, and that is what consumes time in our development organization." - Architect E

They also said that it may be more useful if GRASS could organize the dependencies based on teams instead of ECUs. However, the functionalities owned by the teams are not fixed, so it is hard to trace the dependencies while they are still changing. The teams might change responsibilities and work on other parts of the system. There is also nowhere to collect data on which team works on what at any point, that is reliable for a longer time.

> "You had a group in there with ECUs, but maybe you should group it per team." - Organizer K

Discussion of sizes

Regarding the size of the files, the attendees took a look at the examples we showed, and most of them thought the lines of code did not tell the real problem. One of the larger example files we brought up was a generated file, and it was concluded that it's acceptable to have a file that large since it was not written by humans. The attendees thought that the real problem is more linked to the separation of concern or functionalities, which is not implied by the lines of code. One attendee brought up that the size of the files could indicate how much time the developers need to take to understand the file. It was agreed upon that this could affect the difficulty of refactoring, but is not as relevant in the migration process at Volvo.

> "Of course it takes more time to get the understanding of what the files you were is doing and more lines of code you have." - Architect F

Someone mentioned maybe the size of the software components could perhaps be more telling, to count the number of files inside the software components.

> "Because here is the size of the file, but maybe the number of files is because it is just one big file. Then you have to do any sort of load or if you have multiple files at least you have some thinking. " - Organizer K

4.2.4 Summary

During the interviews, we learned that the dependencies are an issue for the developers. Another finding was that no architecture work covers the whole system. An interesting idea from an interviewee was to look at the sizes of SWCs.

We used GRASS to provide an architecture recovery of the system. This resulted in different graphs, metrics of the dependencies and the sizes of files. These results were discussed in the focus group.

The dependencies were discussed at length in the focus group. That communication between teams is one of the hardest parts was agreed upon. It was also agreed upon that the GRASS graphs seem to indicate that the system looks complex, however, no one knows what an example of a good system would look like. The metrics of dependency and size might be an indication of complexity, but the focus group had ideas that could further improve GRASS. Having the number of dependencies on different teams was one idea that was discussed. Another idea was to have a progression of the system, to see its evolution. The size metric could perhaps be more telling if it dealt with the number of files in a SWC rather than the sizes of files.

4.3 RQ3

Which parts of the physical system (for example ECUs) can influence the migration of the code?

There were several parts of the hardware that influence the migration according to the interviews. One thing they mentioned a lot was the communication in hardware, that there might be more delays while they migrate to the new architecture which needed to be avoided. The developers also had trouble handling the latencies in the process. Moreover, the interviewees told us that the performance of the hardware also needed to be considered.

4.3.1 Interview Results

Some interviewees did not see any big influences on migration based on the physical or hardware structure. This might be because they were not working with the code close to the sensors, but rather the logic. They only needed to consider if the code worked on the hardware.

> "We don't know how the hardware is connected." - Developer A

> "I think the only thing to know is to test the hardware. To see the capacity and then you refactor again, right." - Developer C

Two main issues were discussed most regarding the hardware during the interviews. Both developers and architects thought the communication between hardware components could impact how they design A2 and migrate to A2. Another issue was the centralized structure, which requires the high performance of the new central ECU. Based on the performance of the ECU, the teams needed to consider what they could migrate to the central ECU.

Hardware communication

The communication between ECUs was mentioned several times by the interviewees. The interviewees thought that when they migrated their software to A2, the communication might add some delays if the communication between ECUs is not planned well.

> "But every time we write something on that CAN bus, all the other ECUs that are connected to that CAN bus will wake up to see if the message is for them. That's a big issue." - Developer B

Some interviewees said it would be very different if they use another communication network, but for now, it's not fully clear what they will use.

> "Maybe they give us a separate ethernet line or something or a separate CAN line, then that's different. But I guess the first release will probably be restructured hardware, the same software" - Developer B

> "It's big, it's a big, big change if you go from CAN to ethernet for example and IP-based communication." - Architect E

From the interview with Group Manager J, we also learnt that due to economic reasons of hardware, the automotive industry got involved in these types of changes later. The automotive industry has to make changes with both hardware and software, no matter what changes they want to make. The quality of the hardware needs to be good enough to support the development of software. Therefore, they have to consider the cost of hardware, which has been expensive in the past years.

> "We are more sensitive to cost. The hardware. It means that we want as cheap hardware as possible in the track, meaning that we don't have so much computational power. We don't have so much memory."
> - Group Manager J

However, since the hardware components are cheaper now, they got a chance to start this journey. He also told us they started to introduce Ethernet to Volvo, and they wanted to move there more and more.

> "A computer now had maybe 10 megabytes something, but then there in the mid-90s it started exploding. It became cheaper, cheaper so, and now it has become so cheap that it has also become somewhat affordable to us. And that opens up for our industry also to engage in this. In another way. So access to cheap computation, cheap memory and cheap communication bandwidth network. Like Ethernet."
> - GRoup Manager J

Latencies in the network

Architect E also told us that there are latencies in any network and that it is a complex discussion regarding the hardware. One of the developers also talked about the latency issues they had. They however did not think that the hardware should have a huge influence on the software. The problem was more about how to handle these latencies.

> "The latency is 10 milliseconds, which is unacceptable of course if you want to control like an emergency brake or whatever." - Developer D

> "It is a quite complex discussion when it comes to latencies, bandwidth and deterministic stuff."
> - Architect E

Adaptive AUTOSAR introduces delays

The delays were not only because of the hardware communication. It became clear that the interviewees thought that the use of adaptive AUTOSAR will introduce some delay. There was not a consensus in the interviews about how big a delay the adaptive AUTOSAR will bring, but it seems significant.

> "Such kind of real-time requirements will be taken care of on the classic AUTOSAR end. And not on the adaptive, because due to some latency, or some delay between the communication between different applications, such kind of time critical and the real-time signals would be still in the classic and the adaptive is most time-consuming and where relies on most processing of the data, so such kind of information or such kind of logic would lie in adaptive." - Developer A

> "But every team that wants to move something there, still have communication with the ECUs that are connected to the CAN bus system, they will need that communication. Because the CAN can only connect to the classic AUTOSAR on the [central ECU]." - Developer B

Capacity of the centralized structure

There were many functionalities placed on different ECUs, but they needed to be migrated to the central ECU. Architect F mentioned in the interview that the capacity of the new ECU may have some impacts on they perform the migration. They told us that there were some concerns about the capacity of the central ECU. These concerns were related to that many computer-heavy functionalities were supposed to be placed in this new ECU. Architect F felt that all these functionalities might not fit in the ECU.

> "But since we're not utilising the adaptive part but mostly the classic part of [Central ECU], mostly one of the enablers with the A2 was increased computational capacity."
> - Architect F

> "And if we don't utilize the adaptive part of it, it might be overpopulated already from the beginning."
> - Architect F

Performance of the hardware

Developers B and D felt that the performance of the hardware could be an issue in A2. The performance of the new centralized ECU is not the same as in the ECU they have functionality in at the moment, and this has raised some potential issues for them.

> "We are having some trouble with the performance in the [Central ECU]"
> - Developer B

> "Yeah, because, earlier it was the backbones, we had two CAN backbones, where all the ECUs were connected. Now, due to this new topology, everything is going to the main ECU, and then gatewaying back, yeah that might be an issue in some cases."

> \- Architect H

Developer B also mentioned the boot time of the central ECU is a bit long, which could also cause issues.

> "The [central ECU] has a really long boot time at this point. They say it's gonna be shorter but it's still gonna be a few seconds it seems." - Developer B

Meanwhile, Architect H said the new platform also shared the same time performance issue. This was a problem they had to work out.

> "For example, for the adaptive AUTOSAR to boot up, it takes some, one minute for example." - Architect H

4.3.2 Focus Group Results

During the focus group, it was discussed that the move to adaptive AUTOSAR is a factor in the latency area.

Hardware communication

Adaptive AUTOSAR reaches its full potential when ethernet communication is utilized, however, it is not fully clear which hardware communication alternative will be used in A2. It was discussed that one of the reasons that CAN might be used for A2 is that it is what is known.

> "A big bottleneck is the communication between adaptive and CAN." - Developer B

> "But that was because we know CAN and that's why we continue with CAN." - Architect E

Capacity of the centralized structure

We also found out that the central ECU in fact will have a lot of potency in the cores. This means that some of the issues that were spoken about in the interviews might not be applicable anymore.

> "I can also just mention that the [central ECU] has a lot more potency in the cores. So the [advanced functionality] could probably run there." - Developer D

4.3.3 Summary

The communication system between hardware was discussed multiple times during the interviews. Most interviewees thought the CAN bus would introduce more delays when they migrate when migrating toward the new architecture. They thought it would be better to use Ethernet but CAN bus is more well-known in the vehicle industry and the company. However, the delays needed to be decreased to perform the migration successfully. The latencies were also considered a major issue. The interviewees told us it was always a hard discussion when it came to latencies. This was a problem related to the CAN bus. Therefore, most interviewees said that the communication system might need to be changed anyway. Another thing they were worried about was the performance of new ECUs. Especially when they needed to move functions to the new ECU, there would be a lot of logic processed there. Meanwhile, the adaptive AUTOSAR would also bring significant delays. Thus, the architects and developers needed to consider more when they decided which parts should be migrated to the central ECU. Both during the interviews and focus group it was discussed and concluded that delays in the hardware and the performance of the ECUs influence the migration.

4.4 RQ4

Do the developers and architects think that the whole or part of the migration process can be automated?

There were some mixed results. The architects and developers had different insights on automation. The architects focused a lot on the whole process since they looked at an overview of the architecture, while the developers mainly considered smaller parts when they were performing the migration. Developers were generally positive about applying automation in the process, they thought that some parts can be automated and that this could help their process. Architects on the other hand felt that it is hard to automate the process. An architect told us that the final goal was not clear, and therefore more information was needed to utilize any automation.

4.4.1 Interview Results

Based on the interviews, developers thought some parts of the migration process can be automated. However, the architects did not know if the teams could apply automation for any part. Most interviewees agreed that there was little to no possibility to automate the whole migration.

Architects view

Some architects were not so positive about the idea of automation. This was mostly because they felt it was hard to automate the whole process and they did not have that much knowledge about the smaller parts of the process for each team. An architect also pointed out that no software entity is the other alike, and it is hard to automate because of this. Architect E thought that no parts could be automated since in order to automate, you need to know what the end goal is.

> "To refactor something you need to know where to go, so then you need to program where to go. And it has to be 100% certain, otherwise you can never go." - Architect E

Architect H on the other hand thought that both parts of the migration process, and parts of the architects' job, could be automated. Automating the traceability between the code, the models, and the UML diagrams was one suggestion H brought up. Architect F also thought that there could be some kind of automation regarding the traceability of the system. Another suggestion Architect F had, was to generate keywords describing the signals. F further thought that some kind of gap analysis and design change recommendations could be automated

> "then the traceability will be difficult between the code and the models and even the UML basically. So I think that can be automated" - Architect H

> "If you can identify the descriptions or maybe based on the signal input, output, and generate keywords on what information is this, and map that against the [new layered structure] or the [old layered structure] principles, you could at least automate some kind of gap analysis, and design change recommendation." -Architect F

Developers view

The developers were more positive in this aspect. Although they agreed that the whole process would be hard or impossible to automate, some developers thought that some smaller parts of it could be automated. Developer A had an idea that could help several teams. This automation idea was in an early stage and had not yet been implemented. Some of the developers had already automated parts of their workflow. Developer B had for example automated the translation of signals. The automation consisted of a script that makes the signals fit with the adaptive AUTOSAR framework instead of the classic. The developers thought that these small automations could be generalized to other teams, with minor modifications. Developer C had not performed automation yet but was positive and thought that writing wrappers and unit tests could be automated. There was a consensus however in that no one thought that the whole process could be automated.

> "If we can see a way in which this [interface] thing can be automated, it will be very helpful for both the [adaptive] team, working, and also the [classic] team." - Developer A

> "But the syntax is completely different in adaptive and classic. So we did automate that translation to not have to create 300 signals manually." - Developer B

Developer B thought there should be some automation regarding traceability, reinforcing the idea that the architects also had. They thought that it would make the whole process much easier.

> "I think the whole creation of adaptive applications should be automated. Because right now just creating a new project in adaptive is way trickier than it should be." - Developer B

Good CI/CD pipelines ease the work

Some interviewees told us there was a good continuous integration and continuous delivery/continuous deployment (CI/CD) pipeline which helped them integrate and deploy. The pipeline itself had some automations incorporated. However, the pipeline could be further improved in some cases so that the developers could get more assistance. Developer D thought that the merging could be automated.

> "And already you have all the process from the deployment that is automated but that can be improved." - Developer C

> "Of course, we have smaller steps in our flow that could be automated, like merging, you could automate that. Since everything takes so long when you push a change, you want to merge, you have to sit and wait for the regression and the progression and all that stuff." - Developer D

4.4.2 Focus Group Results

In the focus group, most attendees thought it was not possible to automate the whole process. However, they draw a general agreement that some parts can be automated and can be applied by all the teams. Although these kinds of automation might only help small parts of the migration, they would at least save time for the development teams.

Automation of the whole process

Everyone in the focus group agreed that it's very difficult to apply automation for the whole migration process. Not only because they were not sure about the final version of A2, but also because the adaptive AUTOSAR, which was the new platform they were going to use, was not very good at handling the communication protocol they were using. One of the attendees thought that even though some parts could be automated somehow, it still required information which they could only input manually.

> "So it's creating those in the tool just not be possible but only one good thing for automation in I don't know A2 is this really about the CAN database question is a bit more automated now. It wasn't very manual before." - Developer D

> "But it's still someone still needs, even if you have a code generator for that one, you still need someone to tell the code generator what they just convert from and map it so it's" - Architect E

Automation of some parts of migration

Most developers in the focus group believed some parts could be automated and can be used by other teams. They thought even though these kinds of automation only worked for small parts, they were still helpful. They could at least save time for the development teams, though this depended on what types of problems the teams had.

> "With some small tweaks. So because I guess each might look a little bit different. But it's also, I mean the automation is, it's not like I'm gonna use it every time it was to make one huge boring task quicker." - Developer B

> "But I think we should automate. If you see something that can be automated. Of course you two like if it's just a matter of writing a script or if it's a complete problem, tactics, that's I think that's more work. " - Developer D

4.4.3 Summary

In general, some teams applied automation and this worked well. The developers believed that automation could be used for the teams. The attendees also felt more parts can be automated to save time. Nevertheless, the attendees thought it would be impossible to automate the whole migration process due to multiple reasons.

5 Discussion

In the discussion, we discuss the results and reflect on the different data sources that make up this thesis. We also discuss the limitations and validity threats.

5.1 RQ1

RQ
Are there any common practices that the teams are using while designing how to migrate the source code, and while performing the migration?

ANSWER
We did not observe that the teams use common practices. It is hard to suggest a common practice for the teams due to the changing plan and different problems for the teams.

According to the results from the interviews and the focus group, the changing plan of the new architecture was a major problem in the migration process. As a large company, Volvo needs to keep delivering new products, thus they have to add new functions while performing the migration. Meanwhile, the teams needed to handle different problems. Therefore, the architects had to change the architecture for the new functionalities and problems. Subsequently, there were more communication needs between architects and developers. That also caused issues, as the teams could not align their work while they had a lot of dependencies between the teams.

Since the existing architecture was complex, it was hard to have a fixed plan from the beginning. The architects told us that no one could consider everything in such a complex architecture and deliver a good design. Additionally, the technologies are growing fast, thus it was not reasonable to have a fixed plan which would be reached after a couple of years. Since the plan for the new architecture is changing, it was hard to provide a common practice for the teams.

The developers need to develop new functionalities while maintaining the existing products. So, they had strict time and resource constraints. The teams required more time and human resources to perform the migration. The constraints differed between the teams, thus they started the migration at different times.

We learned from the interviews that all the teams need to start to migrate at approximately the same time so that they can align their work. It would then be easier for the teams to share information and resources, so they can learn from each other. Additionally, it would be easier for the teams to communicate if the dependencies among the teams could be reduced.

As the market is developing, the mindset of the company also needs to be changed to adopt the developments. The architects told us that as Volvo was trying to move to SOA, it required the people here to think more service-oriented. In order to make the migration happen faster, they also need to work on continuous learning. The company cannot perform the migration successfully unless the people involved change their mindset to accept the need for changes instead of maintaining the existing architecture.

In general, it is difficult to provide a common practice in a large company such as Volvo Trucks. Many things are happening when they decide to migrate. However, the whole company should start the migration together in order to find some common practices or guides that can be used for most teams.

5.2 RQ2

RQ
Is it possible to use the GRASS tool to identify structures in the source code that are considered to make migration hard by the interviewees?

ANSWER
GRASS can identify couplings in the structures. The couplings indicate the difficulty of migration.

We learned from the focus group that GRASS can identify structures in the source code that indicates the difficulty of the migration. However, the discussions indicated that GRASS might not be useful to the teams in their migration process. The focus group came with suggestions on how to make GRASS better suited for use in their work.

The first suggestion was to use GRASS to track the evolution of the software. This could be useful in seeing the changes in complexity over time. An idea was to have a current version of the graph in the lunch room, to visualize and perhaps motivate developers toward working on the visible issues.

Another suggestion was to specifically look at the dependencies between teams. During the focus group it was discussed one of the main time-consuming things for the teams is the need to communicate with all teams that they have dependencies to. It was discussed that the total number of dependencies only indicated that *it looks bad*, but it was not clear what that meant. If GRASS instead could look at dependencies between teams, it would be clearer which teams need to communicate with the largest amount of other teams. This could perhaps indicate which teams most need to look over their dependencies and be a clear indication to some teams that they should become more self-contained. However, the ownership of SWCs changes regularly in the process. It is hard to decouple the SWCs due to the changing ownership, therefore, the dependencies are an issue they need to consider while performing the migration.

In conclusion, GRASS can indeed identify structures that were spoken about during the interviews. The knowledge about these structures alone, however, was not confirmed to be in itself useful to the development teams. Using GRASS in other ways could create a tool that is useful to developers, architects, and managers.

5.3 RQ3

RQ
Which parts of the physical system (for example ECUs) can influence the migration of the code?

ANSWER
The communication between ECUs, the performance of the new ECUs and the physical location of the ECUs matter.

The hardware and software in a vehicle are hard coupled. This means that changes in one usually mean changes in the other as well. To perform the migration at Volvo Trucks, both the hardware and software architecture are redesigned.

The choice of classic or adaptive AUTOSAR depends on the hardware communication chosen, and the performance of the new ECUs. If ethernet is used as the hardware communication, there are more benefits from adaptive AUTOSAR. However, adaptive AUTOSAR requires more computational power, and so the performance of the new ECUs affects this choice as well. The AUTOSAR version has further implications for the software, such as which programming language will be used. These factors affect how the teams plan their migration, as they have to rewrite their code or write wrappers if the choice falls on adaptive AUTOSAR.

The latencies in the network, and of AUTOSAR, also affect the migration. The teams are not clear on what the latencies will be, as testing is still being done. Some teams, such as the ones handling emergency brakes, cannot have large amounts of latencies concerning their function. The migration of these functions can subsequently not be decided until the latencies in the network are discovered.

In conclusion, the hardware has major implications on how the software will be migrated. Since the new systems' hardware is not decided, it is hard for development teams to design their software. The main areas that could help development teams decide on the software migration approach are knowing the performance of the new ECUs, the communication network and the latencies in the system.

5.4 RQ4

RQ
Do the developers and architects think that the whole or part of the migration process can be automated?

ANSWER
The developers are positive and think that parts can be automated. For example, the conversion to a new signal format has already been automated by a team, and they think that this automation can be used by others. They all think that the whole process cannot be automated.

During the interviews, we learned that some teams have started automating some tasks. These tasks were specific to the team and their migration, however, they thought that other teams could use the automation scripts with minor adjustments. One of the main problems with sharing the automation scripts was that the teams had different progress regarding the migration. To efficiently share information on the automation processes, the teams should, as in RQ1, be aligned. The alignment of teams would enable more information sharing between teams.

Another idea that was introduced during the focus group was to provide some kind of template and somewhere to share information about the teams' migration. If the end goal is clearly defined, it would be possible to guide the teams more, and perhaps have a template on how to perform the migration. It could also be useful to have a shared forum where for example the automation scripts are shared. This could help teams who have not yet come as far in the migration process if they have similar problems.

As the end goal is not clearly defined, the idea of automation for the whole process is hard. In addition, the teams are working on widely different problems. These two factors make the idea of having some kind of automation for the whole process impossible. We think that in a company as large as Volvo, the value of automating the whole process would be extremely valuable. However, it seems to us as if it is in large companies that these things are most difficult.

5.5 Implications for Developers and Architects

The implications for developers and architects who perform similar migrations are mainly the need for a clear plan and the importance of communication.

Create a clear plan

These results point to the need for architects to have a clear plan before the developers start the implementation. When the plan keeps changing, we can see that developers have a hard time parsing the requirements. Understanding new requirements and adapting the plans takes a lot of time, and this could be avoided. Creating a clear plan is not an easy task by itself, but it seems as if the time needed to do this could be worthwhile.

Improve the communication between architects and developers

In order to create a plan that fits the developers, we also see a need for more communication between developers and architects. We learned that some teams have frequent meetings with architects, and these teams had a better grasp of what their requirements were. During the focus group, the discussions quickly turned to the naming of files and the reasons for different approaches. This implies that there is a gap in the communication between developers and architects, which could be smaller if there were more communication between the two groups. If the developers and architects had more communication, these issues would likely be brought up in other forums.

The communication between development teams also has an impact. To align the work, and be more efficient in the migration, more teams should be on the same page. This would also enable more information sharing regarding the approaches and automation.

5.6 Threats to Validity

Here we discuss some validity threats of the research. We use the classification defined by Per Runeson and Martin Höst [31].

Construct validity: Can the method accurately answer the research questions?

Internal validity: Are there other factors that might affect the results?

External validity: Can the results be generalized?

Reliability: Is the data dependent on the specific researchers?

5.6.1 Construct Validity

It is hard to look at approaches to migrations without being in a company in the migration process. There are different ways to collect data within a given company. A different approach could have been to send a survey to as many people as possible to gather insights. We chose to have interviews in order to gather qualitative data, and then have a focus group. The choice to not have a survey was that we did not find a reliable way to send out the survey to relevant employees. Instead, we received aid in finding interviewees who were indeed involved in the migration. Moreover, interviews allowed us to dig deeper into the questions. Since the employee who helped us find interviewees was heavily involved in the migration, the most representative interviewees should have been included. In doing an interview case study with a focus group, we believe we had access to good qualitative data.

5.6.2 Internal Validity

There are a lot of developers and architects at Volvo Trucks. To find interviewees during the spring, we relied heavily on a few employees who guided us in this aspect, and we accepted everyone willing to participate in the study. It is possible that the majority of volunteer interviewees were more engaged than the average employee since they voluntarily chose to take part in the thesis. Unfortunately, it is hard to avoid this bias. However, we believe that since we had recommendations from contacts for whom to involve in the study, we had access to the people who were most related to the migration process. These interviewees are a good representation of the process of the migration, and they were able to speak of the experiences of their teams.

Some of the code used at Volvo Trucks is supplied by OEMs, and this code was not included in the data set. The source code included does therefore not include all of the code used in a truck. Since this study looks at software migration within a company, it is reasonable that we only look at the code being developed within the company.

Other changes are occurring at Volvo Trucks than the ones that this thesis focuses on. While they are making this migration toward a centralized hardware structure, they are also making changes in the layers of their code. From a Controller/Handler logic to a more sophisticated logic with several layers. This may be a confounding factor. Subsequently, this could be affecting the opinions and thoughts of the developers and architects working with the migration. We made sure to be clear about what the discussion was about, whether layers, centralized structure, or AUTOSAR, to minimize the risk of mixing the results.

The focus group took place 1-2 months after the interviews. It is possible that the interviewees' opinions had time to change during this time. Not all interviewees had the possibility to attend the panel discussion, this might skew the results toward the opinions of the interviewees who attended the session. We tried to minimize this bias by planning the focus group two months ahead of time and choosing a time when as many people as possible were able to attend.

We interviewed developers, architects, one project manager, and one manager. They confirmed each other's observations, and we believe that the conclusions we draw are reasonable.

5.6.3 External Validity

This case study looked at one case, within one company. It is reasonable to deduce that if the same kind of migration occurred in a similar vehicle company, the results would be similar. However, it is hard to know how far the generalization holds. The move toward a centralized structure is something many vehicle companies are likely to perform, and we believe this study provides insights for them into what might be worth looking into.

5.6.4 Reliability

The results of the case study could be different if another set of interviewees, and questions were used, or if the card sorting was done in another way. If another researcher were to perform the same research, the results might not be identical to ours. However, in the method, we declare the questions used, how the interviewees were selected, and how the card sorting was done. In addition, the interviewees were unknown to us beforehand, which means we reduced the risk of being biased toward any specific interviewee. We believe that these explanations would constrain the differences in results if the study would be replicated.

6

Conclusion and Future Work

Here we come to conclusions based on the results and suggest some areas for future work.

6.1 Conclusion

This thesis looked at the different perspectives of a large company in the light of software migration. It was found that there are multiple different approaches by development teams and that it is difficult to align the work. The GRASS tool can be improved to be used in order to help developers and architects. Since the hardware and software are tightly coupled, the hardware substantially affects the software development. There are parts of the migration process that can be automated, and a forum for sharing these could be useful.

Many things affect the migration process at a large company. Developers, architects and managers all need to be onboard. The organizational aspects are at least as important to a software migration as the technical side is. In summary, we conclude that migrating from one system to another is a huge undertaking. Everyone involved has to understand the complexity and be willing to make changes together. We believe that having well-established communication between teams of developers and architects makes such a process easier.

6.2 Future Work

Covering other companies' migrations, looking further into the communication between developers and architects, investigating if there would be a ripple effect, and looking into other factors, are areas we see could use further work.

We found that the developers and architects thought that the organizational aspects were as important or even more important than the technical aspects of the migration. The choice of creating a split organization, and how much to prioritize the migration in relation to maintaining the current product are examples of things that could be further analyzed. An idea could be to study the differences between organizations performing the same type of migration.

The communication between developers and architects was in some cases lacking. A lack of communication affects how the developers approach the migration and could cause gaps in the architects' understanding of how developers deal with different issues. We think it would be interesting to dive deeper into this issue and study how the communication between the groups affects the migration of software. Extending the research to also look into how managers perceive changes and communicate with architects could be another point of interest.

During the interview with the manager, he brought up the idea of a ripple effect happening in the organization. He thought that if some developers started performing the migration, more developers would want to, and have to, join. We think it would be interesting to test this hypothesis. This would require a longer-term study, to see the effects of teams starting the migration over a longer time.

Another idea could be to look at other factors at Volvo. Not only is the migration that Volvo Trucks is performing changing the software architecture and moving toward SOA, but they are also changing its layered approach. The changes in layer are connected to the changes in AUTOSAR. When using classic AUTOSAR, the layers mainly consist of controller and handler. With adaptive AUTOSAR however, the layers become more complex. Some interviewees brought up these changes during the interviews, and we think it could be interesting and insightful to look at how this aspect affects the migration. There was a talk during the focus group on how to use the GRASS tool in relation to the layers. We believe that the GRASS tool could be used to create graphs with different clusters for the layers, and thus could be a tool in a study looking at the layers.

Bibliography

[1] A. H. Easton and G. C. Cromer. “Bus.” [Online]. (2023), [Online]. Available: https://www.britannica.com/technology/bus-vehicle (visited on May 17, 2023).

[2] Silvaco. “Design IP for automotive SoCs: Trends and solutions.” [Online]. (2023), [Online]. Available: https://silvaco.com/blog/design-ip-for-automotive-socs-trends-and-solutions/ (visited on Mar. 14, 2023).

[3] AUTOSAR. “History.” [Online]. (2023), [Online]. Available: https://www.autosar.org/about/history/ (visited on Nov. 28, 2022).

[4] G. M. Smith. “What is CAN bus (controller area network) and how it compares to other vehicle bus networks.” [Online]. (2021), [Online]. Available: https://dewesoft.com/blog/what-is-can-bus (visited on May 3, 2023).

[5] Continental. “E/E architecture: Tailor-made system architecture.” [Online]. (2023), [Online]. Available: https://www.continental-automotive.com/en-gl/Construction-Mining/Technology-Trends/E-E-Architecture (visited on Apr. 3, 2023).

[6] D. Parthasarathy, C. Ekelin, A. Karri, J. Sun, and P. Moraitis, “Measuring design compliance using neural language models: An automotive case study,” ser. PROMISE 2022, Singapore, Singapore: Association for Computing Machinery, 2022, pp. 12–21, ISBN: 9781450398602. DOI: 10.1145/3558489.3559067. [Online]. Available: https://doi.org/10.1145/3558489.3559067.

[7] AUTOSAR. “History.” [Online]. (2023), [Online]. Available: https://www.autosar.org/about/history (visited on Apr. 26, 2023).

[8] AUTOSAR. “Classic platform.” [Online]. (2023), [Online]. Available: https://www.autosar.org/standards/classic-platform/ (visited on Apr. 26, 2023).

[9] S. Fürst and M. Bechter, “AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform,” in *2016 46th annual IEEE/IFIP international conference on Dependable Systems and Networks Workshop (DSN-W)*, IEEE, 2016, pp. 215–217.

[10] E. Hernandez. “What is adaptive AUTOSAR?” [Online]. (2020), [Online]. Available: https://www.lhpes.com/blog/what-is-adaptive-autosar (visited on Apr. 26, 2023).

[11] R. Guru. “Shotgun surgery.” [Online]. (2023), [Online]. Available: https://refactoring.guru/smells/shotgun-surgery (visited on Mar. 29, 2023).

[12] T. Tsumuraya. "The evolution of E/E architecture and software platform for R-Car/RH850." [Online]. (2021), [Online]. Available: `https://www.renesas.com/us/en/blogs/evolution-ee-architecture-and-software-platform-r-carrh850` (visited on Apr. 3, 2023).

[13] S. M. Salman, A. V. Papadopoulos, S. Mubeen, and T. Nolte, "A systematic migration methodology for complex real-time software systems," in *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, 2020, pp. 192–200. DOI: `10.1109/ISORC49007.2020.00041`.

[14] D. Kum, G.-M. Park, S. Lee, and W. Jung, "AUTOSAR migration from existing automotive software," in *2008 International Conference on Control, Automation and Systems*, 2008, pp. 558–562. DOI: `10.1109/ICCAS.2008.4694565`.

[15] S. Jain and A. Saha, "An empirical study on research and developmental opportunities in refactoring practices.," in *SEKE*, 2019, pp. 313–418.

[16] A. Vogelsang, "Feature dependencies in automotive software systems: Extent, awareness, and refactoring," *Journal of Systems and Software*, vol. 160, p. 110 458, 2020, ISSN: 0164-1212. DOI: `https://doi.org/10.1016/j.jss.2019.110458`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0164121219302328`.

[17] U. Eliasson, R. Heldal, P. Pelliccione, and J. Lantz, "Architecting in the automotive domain: Descriptive vs prescriptive architecture," in *2015 12th Working IEEE/IFIP Conference on Software Architecture*, 2015, pp. 115–118. DOI: `10.1109/WICSA.2015.18`.

[18] X. Zhang, M. Persson, M. Nyberg, *et al.*, "Experience on applying software architecture recovery to automotive embedded systems," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014, pp. 379–382. DOI: `10.1109/CSMR-WCRE.2014.6747199`.

[19] J. Lee and L. Wang, "A method for designing and analyzing automotive software architecture: A case study for an autonomous electric vehicle," in *2021 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI)*, 2021, pp. 20–26. DOI: `10.1109/ICCEAI52939.2021.00004`.

[20] S. Dersten, J. Froberg, J. Axelsson, and R. Land, "Analysis of the business effects of software architecture refactoring in an automotive development organization," in *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, IEEE, 2010, pp. 269–278.

[21] S. Dersten, "Towards a guideline for refactoring of embedded systems," Ph.D. dissertation, Mälardalen University, 2012. [Online]. Available: `http://www.es.mdh.se/publications/2441-`.

[22] S. Dersten, J. Axelsson, and J. Fröberg, "An empirical study of refactoring decisions in embedded software and systems," *Procedia Computer Science*, vol. 8, pp. 279–284, 2012.

[23] G. N. Vo, R. Lai, and M. Garg, "Building automotive software component within the AUTOSAR environment - a case study," in *2009 Ninth International Conference on Quality Software*, 2009, pp. 191–200. DOI: `10.1109/QSIC.2009.34`.

[24] G. Reichart and R. Asmus, “Progress on the AUTOSAR adaptive platform for intelligent vehicles,” in *Automatisiertes Fahren 2020: Von der Fahrerassistenz zum autonomen Fahren 6. Internationale ATZ-Fachtagung*, Springer, 2021, pp. 67–75.

[25] J. Henle, M. Stoffel, M. Schindewolf, A.-T. Nägele, and E. Sax, “Architecture platforms for future vehicles: A comparison of ROS2 and adaptive AUTOSAR,” in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2022, pp. 3095–3102.

[26] A. Arestova, M. Martin, K.-S. J. Hielscher, and R. German, “A service-oriented real-time communication scheme for AUTOSAR adaptive using opc ua and time-sensitive networking,” *Sensors*, vol. 21, no. 7, p. 2337, 2021.

[27] J. R. Feagin, A. M. Orum, and G. Sjoberg, *A case for the case study.* UNC Press Books, 2016.

[28] N. Babich. “Card sorting best practices for UX.” [Online]. (2019), [Online]. Available: `https://xd.adobe.com/ideas/process/information-architecture/card-sorting-best-practices/` (visited on Mar. 22, 2023).

[29] A. H. D. of the English Language. “Focus group.” [Online]. (2022), [Online]. Available: `https://www.ahdictionary.com/word/search.html?q=focus+group` (visited on Apr. 26, 2023).

[30] NetworkX. “Networkx.” [Online]. (2023), [Online]. Available: `https://networkx.org` (visited on Feb. 3, 2023).

[31] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical software engineering*, vol. 14, pp. 131–164, 2009.