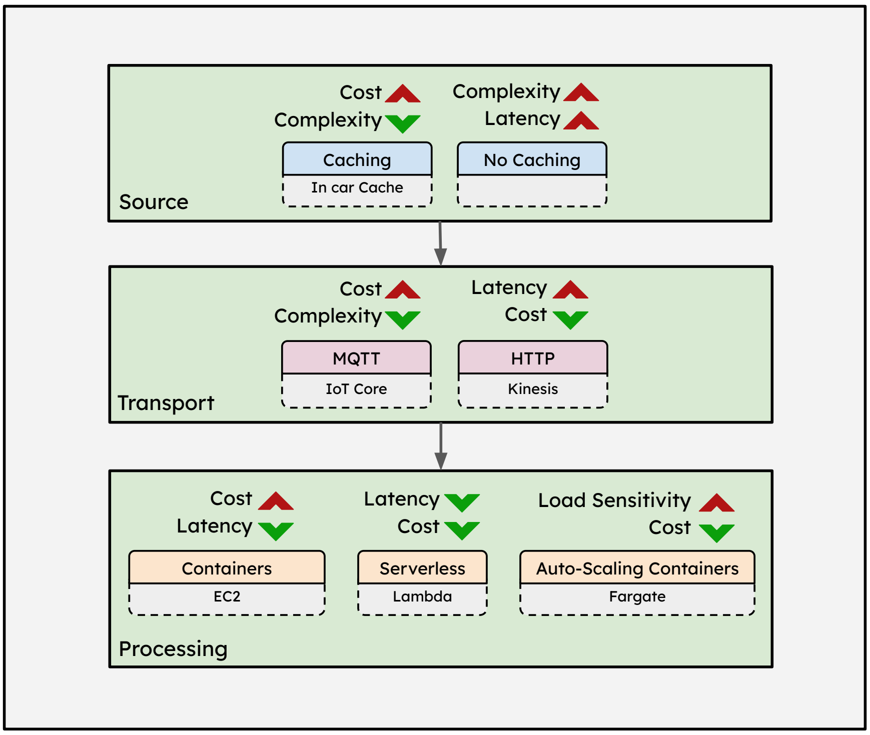

Methods for Understanding Tradeoffs in Software Architecture Design Spaces

A data-driven approach for cloud systems

Master's thesis in Software Engineering and Technology

JACOB MESSINGER, KARL ÖQVIST

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Methods for Understanding Tradeoffs in Software Architecture Design Spaces

A data-driven approach for cloud systems

JACOB MESSINGER

KARL ÖQVIST

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

Methods for Understanding Tradeoffs in Software Architectures Design Spaces
A data-driven approach for cloud systems
JACOB MESSINGER , KARL ÖQVIST

Supervisor: Rebekka Wohlrab, Computer Science and Engineering,
Chalmers University of Technology
Supervisor: Henrik Magnusson, Vehicle Data and Diagnostics, Volvo Cars
Examiner: Birgit Penzenstadler, Computer Science and Engineering,
Chalmers University of Technology

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

Cover: Tradeoffs for component selection in a cloud software architecture.

Typeset in LaTeX
Gothenburg, Sweden 2023

Methods for Understanding Tradeoffs in Software Architectures Design Spaces
A data-driven approach for cloud systems
Jacob Messinger, Karl Öqvist
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Designing software architectures that utilize cloud services in vehicular systems includes navigating a large design space of design decisions. Each decision has the potential to enhance certain qualities of the system but also entails a tradeoff by worsening other qualities simultaneously in most cases. Previous research into architectural tradeoff analysis of these design spaces has successfully introduced approaches to describing architectures and visualizing tradeoffs. This thesis uses a real-world use case in the vehicle industry to simulate software architectures developed by architects at Volvo Cars. This is performed to collect data to be then able to perform an architectural tradeoff analysis. Furthermore, this thesis gives insight into the current practices and challenges with understanding architectural tradeoffs for software architects in the industry. This thesis uses a design science approach over three iterations, where an interview study and evaluation workshop between iterations give crucial feedback on the development. The analysis and visualizations included Principal Component Analysis (PCA), Decision Tree Learning (DTL), PCA loading score plots, and radar plots. The findings show that the analysis and visualizations can be useful for architects to make well-founded design decisions and better understand the tradeoffs in software architecture.

Keywords: software architecture, architectural analysis, cloud systems, design spaces, design decisions, decision making, tradeoff analysis, architectural tradeoff

Acknowledgements

Firstly, we would like to express our sincere gratitude to our academic supervisor Rebekka Wohlrab for her exceptional dedication, knowledge, and support in mentoring us through this project. Without her, this thesis would not have been possible.

Secondly, we would like to express our appreciation to Henrik Magnusson at Volvo Cars for his generous allocation of time, expertise, and resources toward ensuring the success of this research project.

We would like to extend our gratitude to Birgit Penzenstadler for her comprehensive evaluation, constructive critique, and guidance, which have contributed to the scholarly merit of this paper.

A heartfelt thank you goes out to Volvo Cars for allowing us to write our thesis with them. A special thanks to everyone who took part in the interview study and workshop. We could not have asked for a better company to write this thesis with.

Lastly, to our families for their constant support, love, and inspiration, which have made this achievement possible.

Jacob Messinger and Karl Öqvist, Gothenburg, May 2023

List of Acronyms

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

ATAM	Architecture Tradeoff Analysis Method
AWS	Amazon Web Services
DTL	Decision Tree Learning
PC	Principal Component
PCA	Principal Component Analysis
PFO	Pareto Front Optimization
RQ	Research Question
SQS	Simple Queue Service
VCC	Volvo Car Corporation

Contents

List of Figures

List of Tables

1
Introduction

Modern cars today are made up of complex software systems, where an increasing amount of critical parts of the vehicle are software driven. These software systems continuously record, process, retrieve, and compute data, which aids driving experience, safety, and the development of future vehicles. Volvo Car Corporation (VCC) employs thousands of developers to create these systems, working on everything from diagnostic data to steering and braking to the infotainment system and climate control. As complexity grows, so does the demand for well-designed systems that fulfill the goals of creating the best possible vehicle while maintaining quality attributes such as complexity and scalability. All this while running a profitable company, which entails reducing costs, time to develop the products, and adding new features. In a large organization such as VCC, the tasks of designing the systems are, first and foremost, the responsibility of software architects, working to fulfill the goals and meet the system's requirements in the best way possible. When designing software architecture, design decisions can greatly impact the system's quality [1]. However, most design decisions are based on an informal process during the design phase rather than being data-driven with concrete evidence for how the system will perform and what factors influence the system's quality [2][3].

One of the key concerns when designing software architecture is tradeoffs. Tradeoffs can be described as entailing that an improvement in one area negatively impacts another [4]. Most decisions taken in architectural design mean that tradeoffs are inadvertently created. For example, a common tradeoff arises from using more powerful hardware that leads to greater performance but will also increase the monetary cost of the system.

Research has been conducted on ways to support and understand architectural tradeoffs in the design phase of software architecture. The uncertainty surrounding how different design decisions causes tradeoffs and what these would entail on software quality can be explained by linking architectural decisions, tradeoffs, and quality attributes together. Multiple techniques for analyzing and visualizing these connections have been proposed to improve decision-making in the design phase of software architecture [5] [6].

However, tradeoff visualization and analysis techniques have yet to be implemented in an industrial setting. This study extends the current architectural evaluation practices with a more granular analysis of tradeoffs that can be used by practitioners in the industry to support decision-making surrounding software architectures.

Specifically, this study extends and applies the catalog of visualization and analysis techniques for explaining tradeoffs proposed in research that can be utilized to support decision-making by software architects. These techniques can be used to discern and understand tradeoffs within and between software architectures. Additionally, the study shows how a simulator can be utilized to simulate software architectures to perform the tradeoff analysis without relying on real-world data. This study also aims at linking current practices and challenges with understanding and eliciting tradeoffs in distributed vehicular systems in the industry with research conducted in academia.

1.1 Research Objectives

This study investigates the challenges of understanding architectural tradeoffs in a cloud-based vehicular system. The aim is to collect experience and knowledge in the current decision-making practices in complex design spaces of such a system. Furthermore, the purpose is to investigate how tradeoffs can be elicited and characterized to support decision-making in a distributed cloud context.

This thesis aims to answer the following research questions to bridge the gap between what has been proven in research but is yet to be implemented and tested in the industry. Research Questions (RQ) 1 and 2 revolve around the current practices utilized in the industry and gain an understanding of the tradeoff spaces specific to the chosen system. The current research points to a difficulty in understanding tradeoffs for the practitioners, and so RQ3 aims to aid in this area.

RQ1 What are the challenges with understanding architectural tradeoffs in a cloud-based vehicular software system?

RQ2 What are current practices when making decisions in design spaces for a cloud-based vehicular software system?

RQ3 How can architectural tradeoffs be elicited and characterized to support decision-making in a cloud-based context?

1.2 Report Structure

The structure of this report is as follows:

Section 2 will present all relevant background theories, including software architecture, tradeoffs, cloud services, the scenario used for this thesis, and relevant data analysis and visualization techniques. Section 2 also presents the related works on which the thesis is based.

Section 3 will present the research method used for this thesis and give a brief overview of the approaches and techniques in the development of simulations and analysis, as well as present how the interview study that is a part of this study was performed.

Section 4 will present all the results of the study. This includes the findings of the interview study, a detailed description of the simulations and resulting data, the analysis and visualization methods developed during the study, and the result of applying the methods to the simulation data.

In Section 5, the main conclusions and takeaways that answer the research questions posed in this thesis will be presented. Furthermore, the results will be discussed and elaborated on. This includes a discussion on the correctness of the simulation and the usefulness of the analysis and visualizations in the industry and academia.

2
Background

This section will introduce all relevant areas of theory that constitute the basis for this thesis and is required to understand the results and discussions of this study. Firstly, In Section 2.1, all necessary background information on the definitions of commonly used words and phrases in this thesis, such as software architecture, tradeoff design spaces, and quality attributes, are described. Furthermore, the context of this thesis is explained by introducing Volvo Cars and their needs. Section 2.2 presents the scenario used in this thesis, followed by introducing all included cloud components that would realize a solution to the given scenario and how this would be connected together into an architecture. This is followed by Section 2.3, which goes into the necessary information on the approaches utilized by this thesis to create visualizations. Section 2.4 introduces an introduction to related work to this study and also a commonly used method for tradeoff analysis by practitioners.

2.1 Software Architectural Design

Software architectural design can be seen as a set of design decisions deemed to best fulfill the system's goal [7]. Software architects at Volvo Cars are put in front of a myriad of design decisions intended to improve the system while ensuring quality attributes such as keeping costs and latency down. Some of the decisions are on how data should be computed, when and how to send the data from the car to the cloud, how the data and computations should be divided, and how the different parts of the system should function together. All these decisions constitute what this thesis will refer to as the *design space*. It is the architect's role in the design phase of architecture to navigate the design space and make the decision that solves the problem with good quality of the system.

2.1.1 Design Spaces

As previously introduced, the design space can be described as the set of characteristics or design choices available when designing a software architecture. There is, in most cases, not only one software architecture that solves an issue but multiple ones that all are a product of different decisions that have been taken. The different dimensions in the design space can be expressed in multiple ways. It can be described as design choices such as components and services used in the architecture, settings, and parameters of the components, and also relevant metrics of the architecture, which can be in the form of quality metrics or a metric that can be linked to a quality attribute or requirement. The dimensions are not necessarily independent but can affect other dimensions in the design space. For example, a set of design decisions can constitute a 'legal' solution that must be taken together to be realized in practice. Correlations between the dimensions for a given design space can then be computed to express the likelihood of meeting the requirements set on the metrics of the design space, thus giving guidance on which designs should be considered as good solutions to the problem [8].

2.1.2 Cloud-based Architectures

Relying on cloud-based services for computations, data management, diagnostics, and storage in software architectures has risen in popularity the recent years. So are the architectures in this thesis where all elements of the architecture apart from the car are cloud base. The cloud elements of this thesis are created using cloud-based services from Amazon Web Services (AWS). AWS is one of the biggest cloud providers widely used in the industry [9].

Some of the perks of using cloud-based solutions are that companies only need to pay for what they use, they don't pay upfront costs for hardware, they don't have to host the servers and databases, they can more easily and dynamically scale the system, and they can assign computational power as they see fit with few limitations on available performance. Additionally, vendors of cloud-based services generally promise very good and consistent reliability and availability of their services.

Drawbacks with cloud-based services are few, especially with proper knowledge of utilizing them to their full potential. The main issue with cloud-based solutions is that if the use pattern and loads on the system are easy to foresee, then all the cloud services can be managed with minimal overhead or waste of resources. However, when the use and traffic of a system are difficult to predict and might contain surprising peaks of data, then the extra capacity of the services has to be provisioned for or left for the service to scale automatically, drastically increasing the cost. This calls for good architectural design that uses cloud services effectively to maintain good software quality while being cost-effective.

2.1.3 Architectural Design at Volvo Cars

This thesis is conducted in collaboration with Volvo Car Corporation (VCC), a vehicle company based out of Gothenburg, Sweden, and has offices and sells cars all across the globe. Founded in 1927, the primary factors for the company's success have been safety and reliability, and they are considered part of the luxury car brand segment [10].

With the advances in software-driven systems, VCC has gone from being primarily a hardware-driven company to now being considered a software-first company [11]. With this comes a need to coordinate software written by many different developers to be deployed on a finite amount of hardware on the vehicle. Developing a vehicle is a daunting task that takes several years from ideation to production. For example, VCC announced in 2021 their plans to only sell electric cars by 2030 [12]. This also means that decisions made early on can have a large and unknown impact on the possibility of added features and functionality down the line. Consider that the competition in the car manufacturing industry is fierce and moving forward at a rapid pace. It is possible to imagine the difficult decisions that software architects at VCC tackle daily.

One of the goals of recent years at VCC is to increase the connectivity of their vehicles. VCC was the first car manufacturer to connect its vehicles to the cloud back in 2014. Since then, a lot of different features have been added. One example is the Volvo Cars mobile app, where you can lock/unlock the vehicle's doors, send navigation plans, and schedule the motor heater [13]. Currently, all new vehicles that come out of the factories are connected to the cloud, and 95% of sold vehicles are connected with opt-in features such as the Volvo Cars mobile app.

2.1.4 Quality Attributes

The evaluation of software architectures often utilizes metrics that can be connected to quality attributes. This is also how the architectures at Volvo and this thesis is evaluated. A few variations exist on what software quality entails and what the quality attributes are. This thesis uses the definitions defined by ISO 25010:2011, a widely used software engineering standard that provides a framework for evaluating software quality [14]. It defines eight quality characteristics that software products should possess: functionality, reliability, usability, efficiency, maintainability, portability, compatibility, and security. These characteristics are further divided into sub-characteristics, which can be used to guide the evaluation of software quality attributes. The standard also provides guidance on the measurement of software quality and the establishment of quality requirements. The goal of ISO 25010:2011 is to help software developers and quality assurance professionals to define, measure, and improve the quality of software products [1]. The quality attributes used in this thesis are directly derived from the ISO 25010:2011 standard, but not every attribute is included.

2.2 Scenario used in this Thesis

In this thesis, a typical architecture in the design phase at VCC has been created to function as the scenario. The scenario will be used for all modeling, simulation, and analysis. The architecture is intended for a system responsible for delivering diagnostic data from the car's battery and contains the following steps:

1. The vehicle produces 8kB of diagnostic data with a frequency of 1 Hz
2. The data is stored in a local cache that can be set up to act as a batching component for the data by only sending data to the cloud when the cache contents have reached a certain threshold size

3. The data is transported to the cloud
4. The data undergoes computations to calculate important metrics such as time to empty and charge rate
5. The data is stored

To realize the cloud side of the architecture Amazon Web Services (AWS) will be used. Within the AWS ecosystem, many services exist to be used in different aspects of software architecture.

2.2.1 Realizing Cloud Architectures With Amazon Web Services

For this thesis, nine services have been identified to play a part in the scenario described in Section 2.2. Some of these services perform the same job in different ways, and some are unique in their abilities. The services and the way they're used in the software architecture for this thesis can be divided up into four broad types: Transport, Processing, Storage, and Utility. As previously described in Section 2.1.2 a significant advantage of cloud services is that the provider handles much of the management of the services. In general, there are three levels of management with AWS services:

- **Fully Managed:** This means that AWS manages all underlying infrastructure for the service, including hardware allocation and installations, but also monitors and scales the system
- **Managed:** This means that while AWS covers the underlying infrastructure and some management of the service, much of the configurations, setting up data sources, and security of the service is managed by the user
- **Self-Managed:** This means that the user is responsible for allocating resources, managing and configuring the service as well as installing and updating all required software

The following sections describe each service implemented as a component of the topologies analyzed in this thesis. A brief understanding of the services can give insight into their role in a given architecture. Section 2.2.2 describes how they are implemented into specific architectures.

Amazon Kinesis

Amazon Kinesis is a fully managed real-time data streaming service for data transport. The data can be ingested from multiple sources simultaneously and processed efficiently and quickly by other AWS services for processing, analyzing, or storing the data.

Amazon IoT Core

IoT Core is a managed service for connecting Internet of Things (IoT) devices to the cloud. It is used for communicating with the devices and data transportation from the devices to the cloud. The data transportation can be done with many different protocols, the most predominant being Message Queuing Telemetry Transport (MQTT) and HTTP.

Amazon Lambda

Amazon Lambda is a fully managed, on-demand, pay-for-what-you-use serverless computing service. The service can be triggered from events such as data ingest, change from other AWS services, or manual trigger. Since Lambda is on-demand, AWS will automatically scale the system as needed.

The Lambda service can be configured with a varying amount of allocated memory, including all working memory and the program code. It can also be configured to be executed in parallel by up to 1000 instances.

Amazon Elastic Compute Cloud (EC2)

AWS EC2 is a service used to run virtual machines or instances. The instances can be used for processing data or doing any other actions. EC2 is fully provisioned, meaning the user specifies what hardware should be allocated to the instances and how many can be used. The scaling of how many instances are running can be managed automatically by EC2 based on the load or other factors, or the user can manually control it.

Amazon Fargate

Amazon Fargate is a fully managed service that runs containers for data processing or other actions. AWS fully controls the scaling of the number of active containers. The user specifies the hardware resources per container by specifying the number of virtual CPUs (vCPUs) and process memory.

Amazon S3

Amazon Simple Storage Service (S3) is an object storage solution. It allows users to store, read and work on the data through HTTP-request over the internet or other Amazon services that directly integrate with S3. Once an object has been stored or altered in S3, it emits an event that can be used to trigger other AWS components.

Amazon DynamoDB

AWS DynamoDB is a fully managed non-relational database that stores data in key-value pairs. It is fully integrated with other AWS services, such as Lambda, and has low latency and high reliability.

Amazon SQS

Amazon Simple Queue Service is a fully managed message queue. It is integrated with many other AWS services that can trigger the SQS to ingest or egress data.

The queue is flushed either when the queue is filled, a certain timeout has trespassed, or it is triggered to be flushed by some other service.

Amazon Step Functions

AWS Step Functions is a fully managed service for configuring and running state machines. The state machines can be seen as a workflow with a set of states and transitions between them. The workflow can be used to manage other AWS services where each state can perform an action, run a service, or execute a function.

2.2.2 Software Architecture Solutions for the Scenario

Architects at VCC involved in designing the system used as the scenario for this thesis had many suggestions on architectures that would function well. From these suggestions, three topologies, meaning the selection of services and connections between them, have been selected to be implemented in this thesis. There exists some variation of services within the topologies as well that will be considered as their own topology but stemming from one of the main topologies. The main topologies have been given the code names `Simple`, `Stream`, and `Sophisticated` for easier identification.

- `Simple` uses either *Kinesis* or *IoT Core* for data transport and*Lambda* or *Fargate* as processing services. If *Lambda* is used for processing, *SQS* is used as an intermediary service before *Lambda* to utilize its resources better and optimize total computation time

- `Stream` similarly uses *Kinesis* or *IoT Core* for data transport. The data then gets ingested into *DynamoDB* where a *Lambda* instance keeps track of how much data has been received from each car. When ample data has been received, it sends the batched data as a job to *SQS* that acts as a queue. The jobs are then fetched by either *Fargate* or *Lambda* which performs the processing

- `Sophisticated` uses *IoT Core* for data transport and then gets processed by a *Lambda* function that checks against a *DynamoDB* database if the container for the data processing is active (all cars are split up so that they have a specific container for their processing). If active, the data is sent to the container. If not active, the data gets stored in an *SQS* queue while the container gets started by a *StepFunction* service. When it is started, the data in the queue is sent to the container, and *DynamoDB* is altered to say that the container is active so that new data to that container is sent directly

As its name suggests, the `Simple` topology is a simple, bare-bones solution. In reality, it would most likely not function properly without great care taken when developing the software code to ensure data is not lost if services were to crash. This is because there is limited caching capability for the services it uses. Since it also does not divide work between cars, the code would require more data management to ensure correct computations and storage of values as well as timeliness since data from the same car could be processed parallel on different processing services. This might cause issues if the ordering of data is important and thus can't be maintained without additional computing steps or services.

The `Stream` topology offers a solution to some of these concerns because both *DynamoDB* and the *SQS* act as a caching service that can be configured to re-transmit data if the processing services *Lambda* and *Fargate* were to lose data for some reason. Additionally, the data gets batched into processing jobs based on what car they stem from, which ensures that data from the same vehicle is not processed in parallel, which makes it much easier to ensure stricter data consistency.

The `Sophisticated` topology suggests an even more capable solution. In addition to *DynamoDB* and *SQS* being capable of re-transmitting data and acting like caches, it also assigns individual containers to the cars. This allows different data processing or code to run on different cars. The containers are preassigned to the cars beforehand and do not necessarily need to be replicas of each other as *EC2* can run many different containers in the same service. This can't be done using *Fargate* or *Lambda* for the data processing in `Simple` and `Stream` as instances in these services must be replicas of each other. This might prove beneficial for VCC as they operate worldwide and might want to do different data processing based on the region the data stems from.

2.3 Data Analysis of Design Spaces

One of the thesis objectives is to show ways of analyzing design spaces and tradeoffs in software architecture design. This section will give a background on techniques used in this thesis for performing such analysis.

2.3.1 Dimensionality Reduction with Principal Component Analysis

As described in Section 2.1.1, the design spaces of software architectures are multi-dimensional, which is inherently difficult for humans to comprehend, analyze, and visualize due to the many interconnections the data share. There exist many methods to aid in this understanding. In general, they aim to reduce the dimensionality of the data by combining them, keeping only the most relevant, or focusing on one or a few dimensions at a time.

For dimensionality reduction in this thesis as well as in related research as described in 2.4, Principal Component Analysis (PCA) has been used. It is a widely used statistical method that aims to reduce the dimensionality of data while preserving as much of the original information as possible. The approach divides the data into smaller sets called Principle Components (PCs), which are linear combinations of the variables within the data. The PCs are created in such a way that the first PC explains the greatest amount of variation in the data, the second PC explains the second greatest amount of variation, and so forth. Only the PCs that explain the most variation are kept in the analysis. In most cases, the first two or three PCs explain a significant amount of the data variance and constitute the new coordinate system of the data where each PC is a dimension [15].

In detail, the following steps are performed in a PCA [16]:

1. Standardizing the data so that the data have a mean of zero and unit variance. If this is not done, there is the risk of a sample with a large variance dominating
2. Calculating a correlation or covariance matrix that explains the pairwise correlation between all variables in the data
3. Computing the loading score that consists of the eigenvector and eigenvalue from the correlation matrix that explains in what direction the data vary and by how much
4. Decide the PCs by evaluating the loading scores. The highest loading score explains the most variation
5. Selecting the first couple of PCs that combined sufficiently explain the variation and visualize the PCs through any suitable method, for example, a scatter plot of the loading scores of the first PC on the X-axis and the second PC on the Y-axis. This kind of plot is also commonly called a PCA plot.

2.3.1.1 Interpreting PCA Plots

The goal of the PCA is to reduce dimensions but keep the information which can be considered as making the information more compact. Thus, PCA loading score plots, commonly called PCA plots, contain a lot of information but require some knowledge of how to interpret them to extract that information. This section gives a broad foundation on what information can be attained and how that is done.

First, the original dimension in the data has, through the PCA, become a variable or sample, graphically represented as a dot in the plot, with its values dependent on the correlations to all other dimensions. Here on out, the original dimensions will be referred to as a variable.

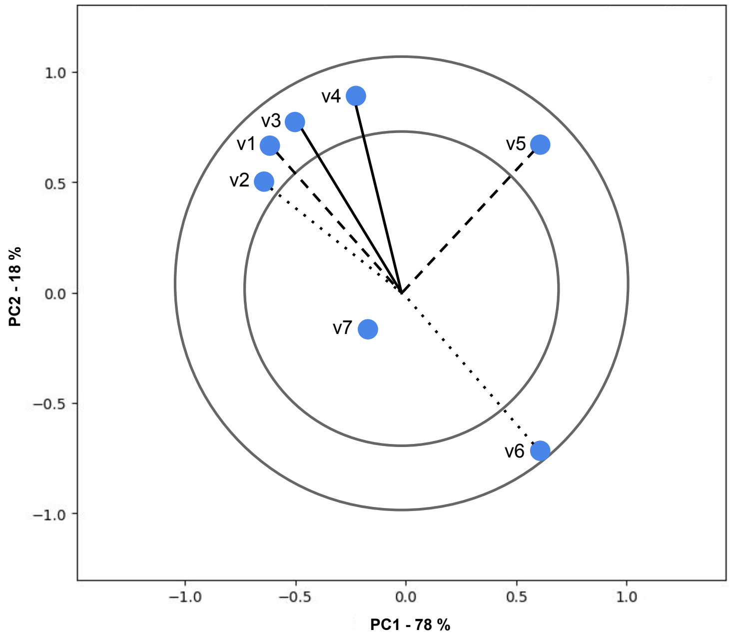

Figure 2.1: *An example PCA plot of seven variables. The filled line shows an acute angle, the dashed line a 90° angle, and the dotted an obtuse angle between variables. The variance that PC1 and PC2 explain are expressed as a percentage on the corresponding axis*

The importance of a variable can be seen by viewing distances from the origin point in the plot. Since the loading scores denote the distance to the new axle in the data for a variable, the larger the loading score, the more variance it accounts for, thus making it more important in understanding the trends in the data [6]. In a PCA plot like the one shown in Figure 2.1, the variable that explains most of the variance in the data is seen further from the origin point. On the contrary, variables close to the origin account for less variance, as seen with v7 in the figure. To aid in this, circles have been added to the plot, with the outer circle having a radius of 1 and the inner having a radius of 0.7.

The other crucial information that can be attained is the correlations between variables. This is done by inspecting the angles between two variables through the origin point [6]. In Figure 2.1, such angles can be seen by following the dotted, dashed, or filled line. An angle close to 90° indicates close to no correlation as with v1 and v5 in the figure. An obtuse angle indicates a negative correlation with an angle close to 180° , indicating a strong negative correlation between v2 and v6 in the figure. Oppositely, an acute angle indicates a positive correlation with an angle closer to 0° , indicating a very strong positive correlation between v3 and v4 in the figure.

Following this, variables clustered tightly together will create an acute angle to each other, meaning that they all share a strong positive correlation. The figure shows this with v1, v2, and v3.

2.3.2 Optimising Data using Pareto Front Optimization

Pareto Front Optimisation is an approach that, from a set of data, creates a Pareto Front, which consists of samples that are optimal solutions to a tradeoff between two or more objectives [17]. An objective can, for example, be *"maximize variable X while minimizing variable Y"* and the Pareto Front will only contain samples that are optimal in such a way that other samples that have a greater value of X also will have a greater value of Y or vice versa.

Optimizations like Pareto Front optimization can be beneficial when applying dimensionality reduction techniques like PCA. Using optimal samples ensures that the PCA tries to explain the data for 'good' samples and does not get clouded by 'bad' or non-optimal data. Since the reduction aims to retain as much information as possible while trying to convey the trends in the data with fewer variables keeping samples that are not of interest regardless might skew the PCA. Using the Pareto Front can help the PCA explain more of the variation with fewer PCs.

2.3.3 Decision Tree Learning

The design space of software architecture is, as previously described, multi-dimensional with many different design decisions. This thesis employs Decision Tree Learning (DTL) to create a supervised model trained on the design space automatically and can help with analysis. DTL is a popular machine learning technique that has been extensively studied and applied in various fields, including finance and healthcare [18] [19]. The technique involves constructing a tree-like model of decisions and their possible consequences to accurately predict the class or value of a target variable for new, unseen instances [20].

The tree is constructed by recursively partitioning the training data into subsets based on the values of selected attributes, with the goal of minimizing the impurity or entropy of each subset. At each node of the tree, a decision is made based on the values of one attribute, and the tree branches out into subsets corresponding to different possible values of that attribute. This process continues until all instances in a subset belong to the same class or have similar values for a continuous target variable [21]. For this thesis, a class could be the classification of how well a topology performs based on the values of the attributes in that topology, which are the parameter values.

One of the primary advantages of decision tree learning is that the resulting model is easy to interpret and explain, as it can be visualized as a tree of decisions that can be easily followed by a human since they closely resemble human reasoning [21].

However, decision tree learning also has some limitations that must be carefully considered when applying the technique. These limitations include the tendency to overfit the training data, the sensitivity to noisy or irrelevant features, and the inability to capture complex interactions between features [21].

2.4 Related Work

This thesis is largely based on previous research on understanding architectural tradeoff spaces, which explores the possibility of linking quality attributes to architectural design variables using machine learning techniques such as Principal Component Analysis (PCA) models, which is a dimension reduction technique described in detail in Section 2.3.1 [5]. By employing such an analysis, architects can better understand the main tradeoffs and relations between the variables that explain differences between configurations. Even though approaches exist to generate architectural designs automatically, it is possible to argue that approaches such as Pareto Front Analysis, as a way to find optimal values while controlling for specific objectives as described in Section 2.3.2, do not constitute a sufficient enough solution to understanding tradeoffs in complex design spaces [5]. It is also possible to argue that this is because such analysis in itself does not link design decisions to how well they satisfy requirements, the data they yield are not easily digestible and understandable by practitioners, and the results are filled with noise that obscures the understanding of relations between variables [5]. This raises the need for a tool or technique to guide architects in the decision-making surrounding architectural setup and design.

Other previous research continues building on these principles by applying them to automatic mission planning for self-adaptive systems and also presents the tradeoffs using other machine learning methods such as clustering, MCA (which can be expressed as being an analysis of qualitative data what PCA is to quantitative data), and Decision Tree Learning (a machine learning algorithm that creates a tree-like model of decisions and their possible consequences, explained further in Section 2.3.3) [5] [6]. Additionally, when data can be characterized as quantitative (in contrast to categorical), PCA plots seem more informative than MCA for joint dimensionality reduction and clustering [6]. Stakeholders in such scenarios are most likely interested in understanding threshold values and what these are dependent on [6].

A systematic literature review has been performed in the field of software architecture [7]. The study categorizes and defines state-of-the-art approaches to architecture evaluation in high-uncertainty environments and the current trends and future directions for architecture evaluation. It also shows that utilizing machine learning techniques in analyzing run-time architecture evaluations significantly improves the decision-making process. However, only 25% of the employed run-time evaluations use machine learning approaches [7].

2.4.1 ATAM - Architectural Tradeoff Analysis Method

One of the research objectives of this thesis is to bring forth a more granular analysis method for the workflow with data-driven decision-making. The new method can then be compared to existing frameworks, where Architectural Tradeoff Analysis Method (ATAM) is a commonly used approach in the industry [22]. ATAM can be explained as a structured technique for understanding the tradeoffs inherent in the architectures of software systems [8] where quality attributes such as cost, reliability, and performance are an outcome of the software architecture that eventually gets implemented. ATAM is a guiding framework for developing a software architecture to fit with respect to several (often competing) quality attributes. By following the guiding principles of ATAM, it is possible to perform analysis and reason about software architectural design decision-making and tradeoffs. It should be duly noted that ATAM is a guiding framework from an abstract point of view and does not constitute specific steps on how to perform the analysis. Instead, ATAM aims to employ the software architect with a workflow suitable for analyzing software architectures. Figure 2.2 shows the different stages of ATAM.

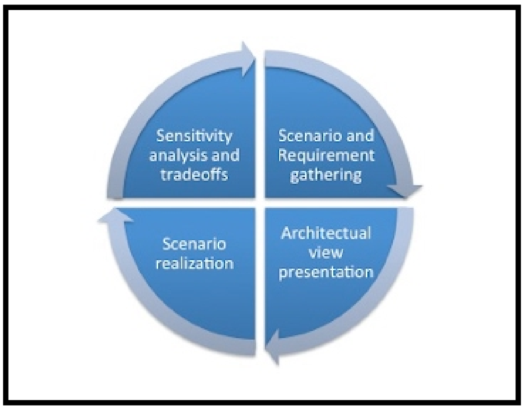

Figure 2.2: *The ATAM cycle, with the different steps of the analysis method described [8]*

ATAM also conveys the notion of competing architectures instead of the inherited intuition of an architect working on **one** architecture [8]. Each set of structural elements constitutes a different architectural solution, described by a set of views, such as a process view for performance-related issues or a dataflow view for reasoning about functional requirements). Electing to include one component instead of excluding it creates a competing model where a decision has to be made based on the quality attributes they affect and the eventual outcome of the system. ATAM is regarded as a continuous process of choosing between competing architectures to eventually reach a consensus on a solution that best fits the needs and requirements as decided before analyzing different solutions. This approach can be utilized when simulating a system which is the case in this thesis. Exploring tradeoffs in and between competing architectures can arm software architects with a more granular analysis than currently possible with ATAM.

3

Research Methods

This section will introduce the procedures and methods employed in this thesis. Section 3.1 presents the study design and approach. Section 3.2 introduces the reader to the approach used in the interview study and gives insight into how a thematic analysis was utilized in the setting of this thesis. It also explains how a workshop was employed in the final stages of this thesis. Lastly, in Section 3.3 and 3.4, the threats to validity and delimitations are presented.

3.1 Study Design

This thesis was conducted using a design science approach [23]. This method has similarities with action research but does not emphasize particular actions to be taken but rather to produce an end product, a design artifact. The design choices in creating the artifact are taken on an empirical basis as the study progress, much like actions are designed and performed in action research. One of the core ideas of the design science approach is that the artifact is created in an iterative fashion that allows the artifact to be thoroughly evaluated in between iterations to ensure steady progress regarding the latest findings in the study.

This study was performed over three iterations, each spanning five weeks and incrementally building the artifact. The artifact in this study is split into two parts: an approach for modeling and simulating cloud architectures and a set of analysis methods and visualizations for analyzing architectural tradeoffs. The analysis methods and visualizations serve the purpose of gaining a greater understanding of architectural tradeoffs, with the end goal of making navigation in architectural design spaces easier and decision-making well-founded.

The modeling and simulations will be developed using Simulink, a software used for the graphical modeling and simulation of dynamic systems in many domains. It is utilized frequently in the industry by companies working in a wide variety of engineering disciplines. It is based on Matlab, a programming language and numeric computing environment. Matlab is multi-functional and allows for implementing algorithms, data plotting, matrix calculation- and manipulations [24].

The analysis and visualizations were developed and performed in Python, a well-used language for many applications, including data analysis and data science [25]. Additional libraries outside of the Python standard libraries were used: paretoset for Pareto Front Optimisations [26], Scikit Learn for Decision Tree Learning [27], and Principal Component Analysis [28], matplotlib [29], and plotly [30] for creating various plots and visualizations.

The research questions for this thesis lent themselves well to this iterative approach with RQ1: *What are the challenges with understanding architectural tradeoffs in a cloud-based vehicular software system?*, giving greater insight into the challenges of understanding tradeoff spaces, some of which were not initially anticipated. RQ2: *What are current practices when making decisions in design spaces for a cloud-based vehicular software system?*, did in a similar fashion identify gaps in the practices used today for making decisions in complex design spaces. All of these findings then influenced the design choices taken when developing the study artifact to explore better approaches to serve as an answer for RQ3: *How can architectural tradeoffs be elicited and characterized to support decision-making in a cloud-based context?*.

The effectiveness and usefulness of the approaches developed as the study artifact in this thesis hinged on understanding current practices and challenges of decision-making in complex design spaces. For this reason, an interview study was conducted to serve both as an empirical basis for the artifact of the study and to answer RQ1 and RQ2 directly. The details of how the interview study was performed are presented in Section 3.2. At the end of the final iteration, an evaluation workshop was held where architects were presented with the analysis results and verified the usefulness of the analysis and visualization methods.

3.2 Interview Study

To answer the research questions and serve as a guide in what to focus on for the simulation and data analysis parts of this thesis, several interviews were conducted with professionals at Volvo Cars that are directly involved in designing, developing, and maintaining software architectures. The subjects had varying experience in the field, varying from 3 to over 25 years in the industry. The structure of the interviews was deemed to fit well into a semi-structured setting where questions from a pre-written interview guide (see Appendix A) could be followed up with follow-up questions on the spot to have an interesting and insightful discussion with the interviewees, which is called a "funnel technique" [31].

The important aspects brought forward in the semi-structured interviews could be analyzed by conducting a thematic analysis. By first coding what was said in each interview into keywords, sub-themes could be elicited that encapsulated different experiences, thoughts, and facts, constituting the driving factors of our design artifact. These sub-themes could then be summarized and included into five main themes. These themes drove the decisions made in the simulator and what to include or factor out. The themes also gave insight into what was important to the data analysis and how to visualize the results.

The coding of the interviews was created in Nvivo, a software tool that enables the user to create and link snippets of the audio recording from the interviews directly to codes. This way, each code includes each instance where an interviewee discusses the subject that the code corresponds to.

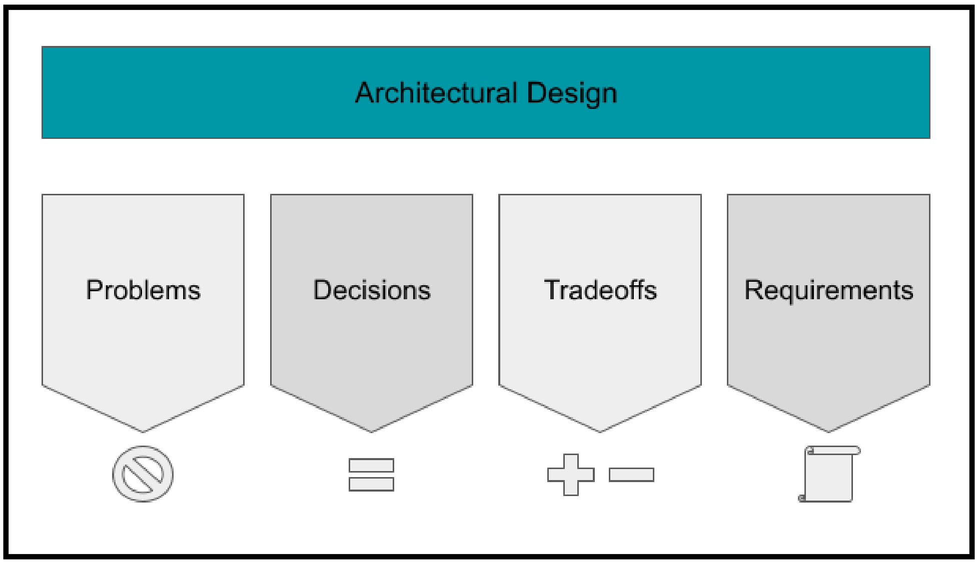

Figure 3.1: *The pillars of the Thematic Analysis*

The architectural design can be said to rest on four pillars which are all interdependent. For this thematic analysis, each pillar (as seen in 3.1) includes several different parts (or elements), encoded from the conducted interviews. For example, codes in the Problem-pillar include several aspects related to the other pillars, such as difficulty understanding tradeoffs, problems with eliciting requirements and understanding the bigger picture, and issues with taking decisions, in the context of software architecture and understanding the consequences.

3.2.1 Workshop

After the conclusion of the interview study and the data analysis, where different visualizations had been created to capture the design tradeoff spaces, a workshop was conducted with architects at VCC. By including a workshop for evaluating the usefulness of different approaches proposed in this thesis, it is possible to answer RQ3: *How can architectural tradeoffs be elicited and characterized to support decision-making in a cloud-based context?* in a concise manner, honing in on the aspects related to supporting decision-making.

In the workshop, the data visualizations created in this thesis were shown to practitioners, and an open discussion on the usefulness of different techniques ensued. No formal interview structure was utilized. Instead, it was explained to the participants that it is interesting to see their reactions and hear their thoughts on the different techniques and if they think such an approach would be useful in their everyday work.

3.3 Threats To Validity

This section covers the threats to validity this thesis is subject to.

3.3.1 Internal Validity

Internal validity is concerned with if the treatment causes the outcome instead of an uncontrolled external factor that influences the outcome. Uncontrolled factors could be unreliable data or biased factors influencing the models and their outputs.

Given that the data from the study has been simulated as the specific dataset needed to conduct the study does not exist, there has been a reasonable step away from creating the most realistic model. Since the purpose of this study is not to create the perfect model but instead to prove research concepts in a real-world setting, a decision had to be made of "good-enough" quality data modeling that captures the systems' main features. However, it should be noted that due to the system being simulated, there exists the possibility that some data or features that might change the overall results of the system are left out. Close collaboration with architects at VCC with knowledge in the domain was instrumental in ensuring a satisfactory level of certainty that the modeled system behaves the way the existing system does. This does, however, mean that any specific values used (for example, when discerning thresholds for decision-making of parametric setup) in this analysis are not to be used as factual statements and instead aim at showing the feasibility of the approaches brought forward in this thesis.

3.3.2 External Validity

External validity is how the conclusion of a study can be applied outside the context of that study.

Because the nature of this study is specific to the use-case scenario, external validity would be considered low due to low generalizability over systems and actors. This is to be expected, and generalizability is not the goal of this research. Instead, the focus is on one particular case that is investigated in detail.

However, the techniques presented in this paper and the conclusions drawn for the current challenges and practices in the industry can be considered generalizable enough to be applied to other research.

Reliability

For other researchers in the field, it is of interest if the results of this thesis can be replicated.

Generally, all of the results of this thesis can be replicated, and many of the visualization techniques used utilize open-source libraries. This thesis also includes an interview guide for conducting the interview study and describes parts of the simulated system. However, the details on how the dataset used to amass these exact results for the given system are not supplied, so a replica would not be possible. Described instead is the general approach and what components were included.

3.3.3 Construct Validity

One of the goals of this project is to examine how practitioners can use tradeoffs in guiding decision-making in architectural design spaces. As tradeoffs and design spaces can both be interpreted differently by different practitioners, there is a possibility that the resulting visual representation of the tradeoff space does not constitute tradeoffs that are of perceived value to the practitioners. This constitutes a construct validity threat.

To mitigate the risk for construct validity, some cautionary measures were employed. In the interview study and workshop, emphasis was placed on explaining this thesis' definition of keywords and phrases, such as tradeoffs and design spaces. Explaining these to the interviewees before asking questions makes it possible to lower the risk of attaining answers where the practitioners have something else in mind.

3.4 Delimitations

This study has been conducted over a specific cloud system for a specific use case at Volvo Cars. This system has been modeled and simulated to fit this project's scope, and thus other parts of the system (both the vehicle and cloud components) have not been included or are of interest to this study.

Due to the lack of an existing dataset from a reasonably large and analyzable system for this project's scope, the research has been performed using simulated data that models a real-world system. This decision was taken in collaboration with Volvo Cars, as the main interest of this study is to achieve a proof of concept of the usefulness of evaluating tradeoff spaces in the industry using various approaches. Any results or conclusions drawn from the results of the subsequent data analysis should not be used to analyze any real system that could include the corresponding real-life components. This is because the simulations have not been thoroughly verified. This study aims to create a proof-of-concept for how data analysis could be utilized when conducting tradeoff analysis in software architecture.

Furthermore, the main focus of this study, as described, is the tradeoff in quality attributes between different architectural topologies, components, and parameters. Hence, all matters not directly related to the chosen quality attributes will not be of interest to this study.

4
Results

This chapter presents the results of this thesis. The first section, Section 4.1 describes the entire process and approach of creating the design science artifact during the three iterations. It further explains the details of the simulator's development, data analysis, and interview study. In Section 4.2, the challenges and current practices with understanding architectural tradeoffs and making decisions in complex architectural design spaces are presented as a result of the interview study. Section 4.3 offers an approach for eliciting and characterizing architectural tradeoffs through modeling and simulation. Finally, in Section 4.4, a set of visualizations and analysis methods for supporting design decisions around tradeoffs are presented. The procedures presented in Section 4.3 and Section 4.4, are performed in the context of the use-case presented in Section 2.2. Finally, in Section 4.5, the workshop findings conducted toward the end of this thesis are presented.

4.1 Process

The thesis followed a design science approach as described in Section 3.1 which entails iterative addition to the artifact. The study was done over three iterations each consisting of multiple steps. Figure 4.1 depicts the steps of each iteration and how information from steps contributes to others.

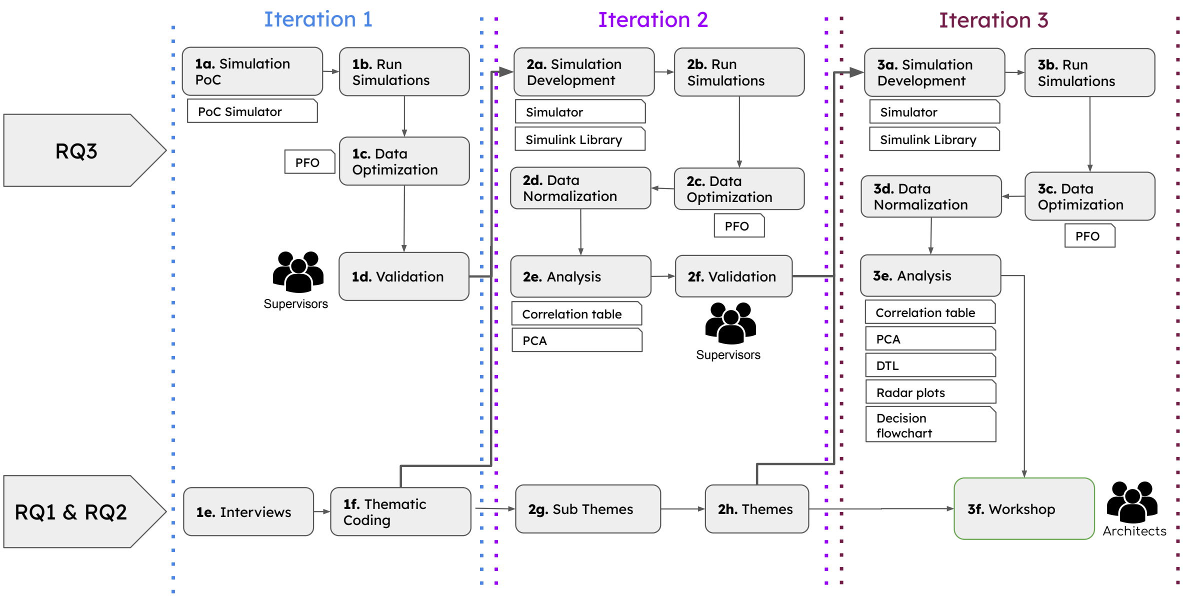

Figure 4.1: *Overview of the workflow for answering the research questions. The white boxes depict the entities that make up the artifact in each iteration*

Iteration 1

The focus in iteration 1 was to get answers on RQ1: *What are the challenges with understanding architectural tradeoffs in a cloud-based vehicular software system?* and RQ2: *What are current practices when making decisions in design spaces for a cloud-based vehicular software system?*. This was done through the interview study and following thematic coding as shown in steps **1e** and **1f** in Figure 4.1. The information from the thematic coding served as input to the simulation development in **2a** and further thematic analysis in stages **2g** and **2h**.

As a first step of answering RQ3: *How can architectural tradeoffs be elicited and characterized to support decision-making in a cloud-based context?* a simulator was developed in step **2a** with which some initial simulations were run. The data from said simulations were optimized using Pareto Front Optimization (PFO) to be later validated by both the academic and industrial supervisor in stage **1d**. The proof of concept simulator and PFO approach were the addition to the artifact for this iteration.

Iteration 2

Iteration 2 marked the beginning of a shift in focus towards RQ3. The thematic coding findings fueled the simulator's continued development in stage **2a** from a basic proof of concept to a complex simulator and Simulink library containing blocks of modeled architectural components for multiple simulations. The development of a library of modular Simulink blocks that model architectural components and use common interfaces for communicating with each other made it possible to simulate many different architectural topologies without specifically creating new models each time the models were simulated, which was performed in stage **2b**.

The simulation data were optimized using PFO and normalized in stages **2c** and **2d**. The data was then analyzed using a correlation table between variables in the data and Principal Component Analysis (PCA), both techniques used in previous research [5]. The results of the analysis were validated with the supervisor group to identify what could be developed further and improved and served as an input for stage **3a** in iteration 3.

The codes that were found in the thematic coding in step **1f** of iteration 1 continued to be processed in stages **2g** and **2h** as per the thematic analysis approach to condense the interview findings down to sub-themes and themes. Extracting themes gave new insights into how the simulations and analysis could be improved, considered in stage **3a** in iteration 3. The insights mainly influenced the selection of quality attributes that architects take into account when making design choices which then got added to the simulator in iteration 3.

Iteration 3

The main focus for iteration 3 was to tie all the results together and get concluding answers for all research questions. This meant final additions to the simulator based on the validation step in iteration 2 and the thematic analysis in the interview study to be able to run the last simulations. The data was optimized and normalized in stage **3c** and **3d** as in iteration 2 before reaching the analysis step. The analysis in stage **3e** was further developed to include more methods and visualizations such as Decision Tree Learning (DTL), Radar Plots, and Decision Flow Charts to accommodate the needs found in the interview study and validation steps in iteration 1 and 2.

A Workshop with architects was conducted in stage **3f** for a final validation and evaluation step. All analysis and interview study results were shared and discussed during the workshop to record working professionals' opinions and thoughts.

4.2 Challenges and Current Practices

To understand how to help architects with decision-making around tradeoffs, it is important first to understand the current practices at VCC and their challenges. These questions can be answered by conducting an interview study and subsequent thematic analysis. The interview study also works as a guiding factor for the tradeoff analysis performed in Section 4.3 in order to gain insight into what aspects are most important to focus on for the tradeoff analysis part of this thesis.

The following sections go into detail about the interview study aimed at answering RQ1: *What are the challenges with understanding architectural tradeoffs in a cloud-based vehicular software system?* and RQ2: *What are current practices when making decisions in design spaces for a cloud-based vehicular software system?*. The interview study also supports the tradeoff analysis, which aims to answer RQ3: *How can architectural tradeoffs be elicited and characterized to support decision-making in a cloud-based context?*. The results are presented as themes and sub-themes, which are then dissected.

4.2.1 Interview Study Approach

As described in Section 3.2, semi-structured interviews were conducted. Questions were asked based on a pre-written interview guide (see Appendix A) with follow-up questions based on the answers given. Each interview was between 20-30 minutes long. The interview guide was also updated somewhat as the interviews progressed. This was performed to improve the flow of the interviews and the likelihood of receiving answers in line with what was of importance to the study.

Role	Years of Specific Experience	Years of Relevant experience
Software Architect	3	20+
Software Architect	1	10
Cloud Architect	3	20+
Advanced Engineering Lead	1	3
Solution Architect	5	25+

Table 4.1: *Overview of interview subjects*

In Table 4.1, the different interview subjects and their experience can be seen. Specific experience in this context would mean experience in the current role, even if that work was conducted outside of VCC. The relevant experience involves everything that could be directly linked to the experience needed to fulfill the tasks faced in the current role. For example, working as a software architect without having several years of experience writing and working with software is not feasible.

The difference between a software architect and a cloud architect mostly depends on which problems they are tasked with. A **cloud architect** is responsible for sending, retrieving, and storing solutions in the cloud and discerning such systems' possibilities and limitations. Whereas **software architects** are generally more concerned with software-specific functions and modules and their interaction with other systems.

In relation to software and cloud architects, a **solution architect** is tasked with more broad solutions comprising many different parts. Abstracting away from specific implementations and having a more abstract overview of how to develop a product. In a vehicle setting, this could include questions such as where different parts should be located in the vehicle to improve data flow and redundancy.

An **advanced engineering lead** is a software architect tasked with researching new methods and ways of solving different problems. Having someone focused on research can be a way to ensure that the team stays informed on the latest methods and stays at the forefront of development practices.

4.2.1.1 Coding

The first step in the thematic analysis was to encode different views, opinions, and facts. Each quote or sentence of interest to this study is allocated to a code. If the sentence/quote fits well into an already created code, then it was added to that code. If the sentence/quote covered something previously not addressed, then a new code was created to cater to this.

Each code was placed under the umbrella of the thematic pillar (as seen in Figure 3.1) that they corresponded to. An example would be the code "Lack of technical aids" that falls under Decision Making.

Area	No. of Included Codes	Total References
Decision Making	33	98
Problems	15	39
Tool Requirements	8	18
Tradeoffs	10	21

Table 4.2: *Overview of areas of talking points and included codes from interview study*

In Table 4.2, the total number of codes can be seen that were included in each area (thematic pillar). No. of included codes are the number of codes that fell into those categories. The Total References are the total number of quotes/sentences coded into that area since each code could be used more than once, both from the same interview and different interviews. An example of this would be the code "Experience-based" which was referenced six times in all five conducted interviews. This code falls under Decision Making and would thus add 1 to the No. of included codes, and 6 to the Total References for that area.

4.2.1.2 Theme Selection

After each interview had been properly encoded, it was time to map these onto a graph. Each parent node was one thematic pillar, and the child nodes constituted each code.

At this point, similar codes were combined into sub-themes, and connections between nodes were established with arrows. After iteratively working with creating sub-themes and relationships, Figure 4.2 was developed.

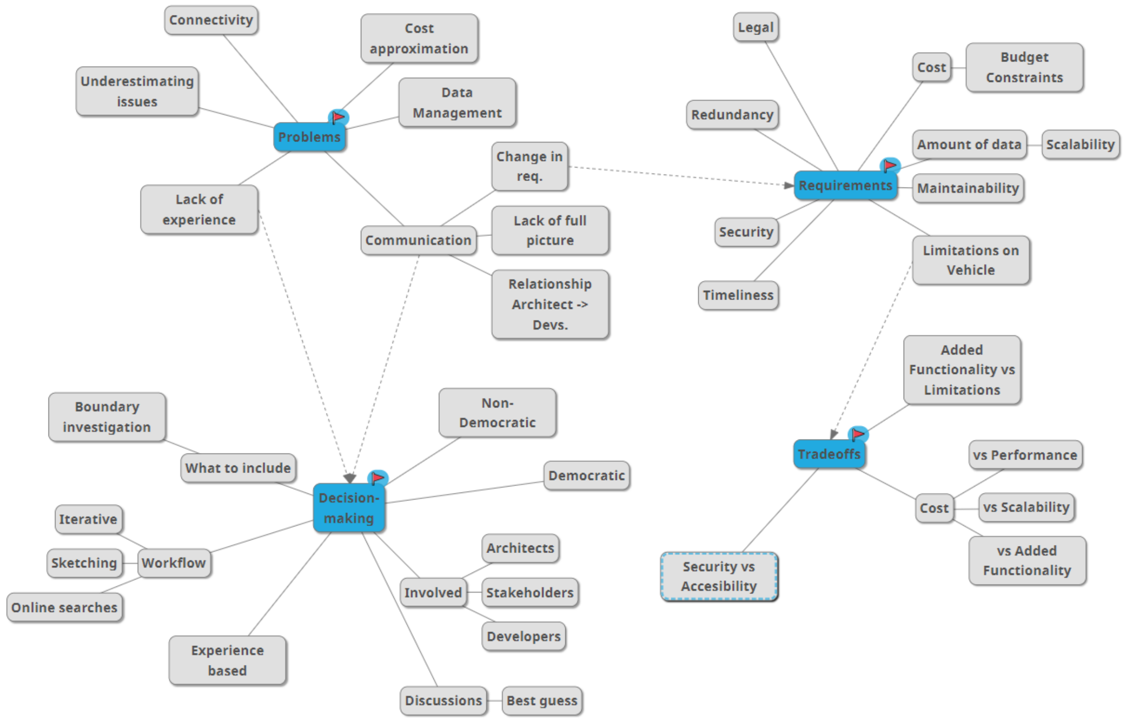

Figure 4.2: *Sub themes acquired from the thematic analysis*

4.2.2 Themes

From the sub-themes in Figure 4.2, time was spent analyzing the root cause for what these hinged on with regard to the importance established in the interviews. This resulted in 5 main themes that are mutually exclusive but ultimately capture a large majority of all codes initially created. These themes are presented in Table 4.3

Theme	Subtheme
Everything stems from cost	Development & Hardware
	System
	Added benefit analysis
Need for new and improved ways of analyzing tradeoffs	Underestimating issues
	Discussion based approaches
	Workflow
Communication is key	Changing requirements
	Involved people
	What to include
Experience is a deciding factor	Cost approximation
	Iterative workflow
	Limitations of environment

Table 4.3: *The main themes and their corresponding sub-themes*

The following sections will dissect these themes, what they mean from a pragmatic point of view, and the sub-themes included in each. All quotes are taken unedited from the conducted interviews.

4.2.3 Everything Stems From Cost

> "Cost is always a very important factor. Is the performance gain worth the increased cost?"

What was abundantly clear from the interviews is that cost minimization at all stages of product development is crucial for a corporate setting (in contrast to academic or research settings). For an R&D department, this might seem controversial, however. After all, the hope is that product improvements and advances in new technology will drive the company's long-term profit. However, some strict guiding factors are especially prevalent in a vehicular context. These are the foundations of decision-making around software architectures and therefore are driving factors for RQ2.

Firstly, the developers of the systems need to work effectively to come up with resolutions to the problems faced, as well as the development of new features. As discussed in this paper, for large software systems, the early stages of ideation and design of such systems are the responsibility of software architects. Since time-to-develop and development costs are important to keep down, the architects cannot evaluate every possible topological setup and idea. This means that, more often than not, the cost is the driving factor in the decisions on how and what to implement.

> "The best software is the one you don't have to write at all"

Secondly, the chosen solution would not be considered feasible without first being acknowledged as a cost-effective/safe way of solving the problem. This is true to varying degrees depending on the level of abstraction the architect work with. For a solution architect, this is among the core principles of which they conduct their work. The cost might be less prevalent for a software architect, depending on the problem. For a smaller-scale software issue, other quality attributes might be of more importance. But the only quality attribute found throughout the conducted interviews to be necessary to essentially all problems is the cost attribute.

> "Usually you realize after a while that this will cost a lot. And then you start thinking about why it costs a lot"

Because vehicle systems are heavily coupled with hardware components, it is not possible to simply add computational units or other hardware components that, at first glance, might seem like the best solution for a particular problem. There are two factors of importance here. Firstly, vehicle platforms are decided on very early, sometimes years before the implementation of software-specific parts. And secondly, each hardware component houses software for many different parts of the system (except for maybe safety-critical systems such as steering and braking). Adding hardware to solve a specific problem might prove impossible and costly since this component must be added to hundreds of thousands of vehicles. It also means that software on these components shares usage with other software. The cost for using the software, such as ram-usage or storage usage, can therefore be deal-breakers since they might break other parts of the system. This also shows one of the challenges software architects face. They need to understand not only the system they themselves are working on but also how their system behaves in relation to, and influences, other systems that share hardware components. This is an important aspect of RQ1, which deals with challenges surrounding understanding tradeoffs in a vehicular setting.

> "The main concern when developing for the cloud is cost aspects. Since depending on the solution, it might cost an enormous amount of money because you are using the tools wrong"

Since this thesis deal with emulating a cloud environment, particular interest was how different topologies differentiate themselves as more probable or viable solutions to a problem in a cloud setting. As discussed, a cloud environment brings with it many positive aspects. However, many of these benefits to other quality attributes (especially redundancy and scalability) quickly falter when faced with the reality that is the resulting cost of the system. Discussing this matter further and gaining increased knowledge of specific cloud components strengthens the notion that many specific tools are intended to work together with other components to some extent. This creates an intricate system of components that one on one might seem cost-effective but prove too costly in the end. Since cost is the primary concern and thus the overall most important quality attribute, as found in this interview study, cost should be one of the quality attributes accounted for in a tradeoff analysis aimed at answering RQ3.

> "The best software is the one you don't have to write at all. Sometimes we need to do something to add functionality or comply with legal requirements, which is very common when developing for vehicles"

Lastly, architects are tasked with conducting what could be considered an added benefit analysis. Whether explicit or not, such an analysis is the basis for everything that gets developed and later implemented. Suppose a solution is deemed too costly, performance-lacking, or needing to be more scalable. In that case, a consensus will be met to exclude that solution from the possible candidates of solutions in the design space. Therefore, an added benefit analysis is also conducted on the proposed functionality (added feature) using analyzing different solutions. If all solutions are infeasible in one way or another, then that functionality will probably be ruled out. The exception to this rule is legal requirements that have to be met, even if the eventual implementation is lackluster in some regard. This is an important insight for RQ2.

4.2.4 Need For New and Improved Ways of Analyzing Trade-offs

> "Usually you underestimate an issue and it turns out worse than you thought it would"

When conducting the interviews, an emphasis was put on trying to understand what is lacking for the architects to make well-founded decisions as a basis for RQ1. To improve confidence that the solution will have any previously unidentified side-effects down the line (during development or after implementation). One of the most discussed points during the interviews was the fact that underestimating issues is tricky. On the one hand, there are too many parameters to be able to control for all possible scenarios, and this would be too costly. On the other hand, architects need to have some level of confidence that the proposed solution will hold and that the risk of negative side effects is mitigated to the best of their ability.

> "I don't think we have a standardized way of working with an architecture"

One finding throughout the interviews which answers RQ2 is that there is no standardized way of creating software architectures and analyzing different pros and cons of systems. Sketching software is the only denominator for all interviewees. Other than that, different approaches are employed. Some architects prefer working iteratively by writing some code and seeing the feasibility of things, then going back to the topology and trying new things until a satisfactory level regarding the quality attributes of importance is met. Other approaches included talking with stakeholders to understand what is most important for the system and which components and features to focus on. Brainstorming is yet another approach that was mentioned in the interviews.

Another finding is the fact that collaboration is a large aspect when it comes to designing systems. Software architects will work together to devise solutions and sanity-check ideas. One question asked was who gets the final say in what eventually gets implemented. The interviews show that this is a team effort; the best idea usually wins and gets implemented. However, there is no standardized approach to evaluating different topologies and tradeoffs that come with different decisions.

> "You certainly want to find any deal-breakers early on so that you can go back or change concept"

When asking questions about how the interviewees' workflow could be improved and how to make decision-making easier, many opinions were lifted. As the quote above mentions, having a more substantial foundation for understanding different options and their underlying constraints early on in the development stage can be beneficial, as this is not currently available to software architects.

> "We don't have the time to try the concept out first before putting it into production. You do it sort of entangled"

As described in Section 4.2.3, since the cost of development is an important aspect during development, any addition to the workflow or decision-making needs to be fast to set up, easy to use, and give meaningful insights at a rapid pace. When discussing our approach and intended solution with the interviewees, it came to our attention that any addition to the workflow would not be feasible if it would work as a bottleneck for the development process.

> "You need to convince yourself and others that the new technology is worth the effort. And I think this can be very hard, to show this"

One of the perks of having more data-driven support for any claims on the system is that it can show other people why one approach should be chosen over another. One of the difficulties software architects face that was discussed in the interviews is that stakeholders and other architects need to be convinced that a decision is the right one before being handed off to development.

> "It's easy to like the new solution"

It is important to consider that a new implementation is not always optimal or wanted; this was explained in the interviews. It was brought to our attention that it can be difficult to evaluate new implementations (new in the sense of topological setup or specific components) in relation to existing solutions. Without dwelling too much on action bias (which is the act of favoring doing something or anything without any prior analysis, even if it is counterproductive), it lies in the interest of a software architect to be able to analyze a new solution and the old one with a side-by-side comparison. Since it was described during the interviews that the software architect also has to convince other stakeholders at the company, examining tradeoffs with different solutions (both existing and new) would be beneficial to answer RQ3.

> "Trying to see how the consequences would be in different steps of the process. What will happen in each of these steps"

Another consequence of improved ways of analyzing tradeoffs in the design phase explained is the need to evaluate an architecture at different stages in the process. Several architects explained that it would be interesting to see what sort of insight a simulation of the system could gain them to understand the different topologies and their components. Today, it is difficult to discern the effects of specific parameters on the system and the cascading impact that an unfortunate decision on these might have on the resulting system. Thus giving further evidence for the need for improved ways to support decision-making around tradeoffs which is an essential insight for RQ3.

4.2.5 Communication Is Key

> "I don't think anyone at the company has a large-scale view of the system. Meaning that we can make little decision here, which adds a lot of complexity for someone else without us noticing, and vice-versa"

Vehicle platforms are the underlying structural topologies for how to build a vehicle. They explain which components should go where and what they are connected to. These platforms have been developed over many years, with hundreds and sometimes thousands of developers. Since there is a limited amount of hardware, several teams must share the hardware for which their software is put on. The interviews explained that each development team would try to look after their own interests and get as many resources as possible for their software. Collaboration can therefore be seen as a key aspect when developing a vehicular platform that influences decisions that need to be made, which is important to RQ2.

> "You start doing something and then you realize that it is an incredible amount of details to everything"

One of the more difficult notions to discern early in the development of a system is the resulting impact on quality attributes that the selected system will have, which serves as a basis for answering RQ1. From the interviews, it was clear that the level of detail needed to fully understand the system is far too complex for any one architect to grasp fully. Instead, they rely on abstractions to some level to conduct their work. The interviewees expressed that it can be beneficial to have someone who is a technical expert on the system to talk to and discuss the feasibility of different solutions. This can then give direction and inspiration on what architecture best fits the system and its intended purpose.

> "The challenge was with explaining to the stakeholders and teams to get the correct requirements. The only tool we had was sketching"

From the interviews, an overall idea of the workflow was elicited to answer RQ2. From the start of a project, when an architect first gets involved, to hand off to the developers. From the interviews, insights were gained into what tools are currently utilized and how these are used. As expressed by the interviewees, one of the most important aspects is getting the correct requirements for the system intended for the new architecture. As the quote above mentions, sometimes explaining clearly what information the architects are after to create a good solution can be difficult. This can lead to requirements that are poorly defined or, worst case, incorrect.

> "It's really difficult to have a grasp on all the details. In reality, it's always more complicated"

The interviews show that software architects work somewhat iteratively to reach a consensus on what system to develop. However, after a hand-off to the developers in charge of development, architects more or less leave the "workflow" and are on to new projects. Thus, any issues with the resulting system performance might not reach back to the architects. From the interviews, it is still unclear how, if, and to what extent architects get informed about how the system performs and behaves (with regard to quality attributes) after implementation.

4.2.6 Experience Is a Deciding Factor

One of the largest realizations when conducting this thesis is that being a software architect is a technically advanced role that requires a lot of experience in software, both writing and reading, as well as deep foundational domain knowledge. Since the experience level varied among the interviewees, several perspectives on experience were explored.

> "When you are not so experienced with a concept, it is easy to focus too much on the advantages but not maybe appreciate the challenges"

Firstly, one area discussed during the interviews is that domain knowledge and previous concept knowledge can be valuable factors in understanding tradeoffs. As explained by one of the architects, if you do not have any previous experience with a concept, it can be difficult to anticipate the drawbacks of such a system since the benefits are often addressed in the documentation of components. This is of interest for RQ1, as the possibility of anticipating system flaws can be attributed to the experience of the software architect. It also gives further evidence of how supporting decision-making can increase confidence for less experienced software architects.

> "To fully understand the impact, whether it's good or bad, is hard"

Secondly, the experience can be a way for software architects to discern influential parameters and components that they anticipate will greatly impact the resulting system. If the software architect is less experienced, utilizing a more data-driven approach to discern these parameters and components could be interesting.

> "You need to have the knowledge of doing failures in order to see if a solution is good or bad. But a tool would help you realize if you have missed anything"

Lastly, architects can use their prior project experience to guide their judgments while developing new systems. They can apply lessons learned from previous initiatives to the current project, identifying trends and best practices. This can assist them in making educated architectural decisions, decreasing the possibility of unexpected results and limiting potential dangers. Utilizing previous experience as a basis for decision-making partly answers RQ2.

Summary

This chapter has presented the interview study results, which serve as a way to understand current practices and challenges with understanding tradeoffs and making software architectural decisions. This was conducted to further understand how different data visualization techniques can help practitioners make said decisions. The following chapter provides insight into how a simulator can elicit and characterize software architectures, which can be used to create data visualizations.

4.3 Architectural Elicitation and Characterization Using Simulations

The following subsections will first, in Section 4.3.1, present all modeled components and their parameters and possible configuration. Secondly, Section 4.3.2 will present how the topologies proposed in the use case of the thesis have been modeled and how the components within the topologies communicate. Section 4.3.3 will explain how the simulations were conducted and what quality metrics were recorded. Lastly, in Section 4.3.4, the simulation results will be shared, how parameters were chosen and set, how the simulations were performed, and the data recorded.

4.3.1 Architectural Blocks

The scenario for this thesis is introduced in Section 2.2, and the possible systems that realize a solution to the scenario contain many different components. Most of the components are AWS cloud services, but there is also one important component in the car. The components are divided into five groups: *Source, Transport, Processing, Intermediary* and *Storage*. These groups will be recurring throughout the results as a high-level grouping.

The first step of the process is where the data is created, the source.

Source Component

The data in this scenario gets created in the car by sensors that collect metrics and diagnostic data from the car's battery. The car then transmits this data to the cloud. This was modeled as one block in Simulink, simulating a specified number of cars producing data packets of a specified size in a specified frequency; for this scenario, as explained in Section 2.2, the number of cars was set to 1000, the data size to 8 kB, and the frequency to 1 Hz meaning one 8 kB piece of data was sent out every second. The source simulates 15 minutes of constant data creation, meaning 15 minutes of constant use of all 1000 cars.

The local cache, which is an option to use in the car to effectively batch the data and send data less frequently, was modeled in the source component as a parameter that could vary as a part of the configuration. In effect, a bigger value of said parameter leads to the source component transmitting data in such a way that it simulates each car sending bigger pieces of data less frequently.

Transport Components

The two data transport options in the chosen solutions for the scenario are AWS IoT core using MQTT as its transport protocol or AWS Kinesis using HTTP. These two components were modeled as separate blocks functioning in a very similar way.

The blocks take the source block output as its input and simulate transport to the AWS cloud. From the input, the blocks get information on the total data amount to send, the size of each packet, and how many cars the data is coming from. This is all taken into account for calculating the quality implications that for both the IoT core block and Kinesis is completely based on the data and the parameter values set for the block.

Processing Components

The scenario dictates that the data from the cars, as its penultimate step once it has arrived in the cloud, gets calculated/processed. There are multiple services that can do calculations on the data. However, three AWS services can be found within the topologies proposed for the solutions: Lambda, Fargate, and EC2. These three services were modeled as three separate blocks in Simulink. From the input, the blocks get the same data as the transport blocks, meaning it is aware of the total data size, the size of each packet, and the number of cars. Together with this data, its parameter values, and predetermined values within the code of the block, the block simulates the calculations being performed and its effects on the quality attributes. Lastly, it outputs the data for the final step, storage.

Storage Component

The proposed solutions for the scenario only have one service to take care of the final storage of the data: AWS S3. Similarly to the other services, this was modeled as its own block that takes the output of the processing component as its input, simulates the storage of the results, and records its implication on the quality attributes. The S3 block marks the end of the data chain in Simulink and thus outputs nothing.

Intermediary Components

The processing services modeled are replicas and, much like their real-world counterparts, rely on intermediary steps, such as managing the processing service, load balancing, or batching the data, to work properly and solve the issue in the scenario. The intermediary steps, as explained in Section 2.2, have been divided up into the AWS services that they utilize, with each service modeled into its own block in Simulink.

Four services have been modeled: Lambda, SQS, Step Functions, and DynamoDB. The Lambda service is essentially the same as the one used for processing but has been slightly modified into two new Simulink blocks, VinBatcher and VinChecker. The reason that they are modeled into their own blocks is that they are very specialized and would be used differently than the processing Lambda service if they were to be deployed in a live architecture. The SQS service similarly has been modeled into two different versions: one that is a generic SQS replica and JobQueue, which is a specialized and tuned SQS service instance.

All intermediary blocks take input from either the transport block or another intermediary block and perform its alterations on the data or trigger other blocks in the same way their real-world counterparts would, then output the data to another intermediary block or a processing block.

Parameters and Block Setup

In total, described above are 13 blocks in Simulink. Table 4.4 summarizes the blocks and what service/component they're modeled after.

Group	Modeled Service/Component	Block
Source	In-Car Cache	Source
Transport	AWS IoT Core	IoT Core
	AWS Kinesis	Kinesis
Processing	AWS Lambda	Lambda
	AWS Fargate	Fargate
	AWS EC2	EC2
Storage	AWS S3	S3
Intermediary	AWS SQS	SQS
		JobQueue
	AWS Lambda	VinBatcher
		VinChecker
	AWS StepFunctions	StepFunctions
	AWS DynamoDB	DynamoDB

Table 4.4: *All blocks modeled in Simulink and what real-world service/component they mimic*

The blocks have been set up so that their parameters, performance, and implications on the quality attributes are similar to their real-world counterparts. Table 4.5 lists all possible parameters, their value ranges, and a brief description of their effect.

Block	Parameter	Min. Value	Max. Value	Value Step	Description
Source	Cache Size	1	10	1	Number of data packets gets cached in the car before transmission
Kinesis	On Demand	0	1	1	Sets pricing scheme (0 = provisioned, 1 = 'pay for what you use')
	Enhanced Fan-Out Consumers (EFOs)	0	1	1	Individual reading channel/consumer (0 = All share, 1 = Each consumer has one)
	Records	200	4000	50	No. of records per second that Kinesis can read (record size = data packet size from the car) (N/A with On Demand = 1)
	Peak Capacity	0	1	0.1	Percentage of capacity to reserve for peak loads (0 = no extra capacity, 1 = 100% of base capacity) (N/A with On Demand = 1)
Lambda	Parallel Instances	1	100	1	No. of instances that can be run in parallel
	Allocated Memory	128	10000	64	Amount of memory allocated to each instance (In MB)
Fargate	vCPUs	2	16	2	No. of virtual CPUs allocated to computing
	Memory Level	1	3	1	Size of memory allocated to each instance. Varies based on the number of vCPUs (1 = Low, 2 = Medium, 3 = High)
EC2	vCPUs	2	16	2	No. of virtual CPUs allocated to computing
	Memory Level	1	3	1	Size of memory allocated to each instance. Varies based on the number of vCPUs (1 = Low, 2 = Medium , 3 = High)
	Instances	1	100	1	No. of instances that can be run at a maximum
SQS	Queue Size	8	1000	8	Memory size of queue (MB)
	Timeout	1	100	1	No. of seconds before queue is flushed (if not filled before that)
DynamoDB	Stream	0	1	1	Stream data or write and read through PUT and GET
	On Demand	0	1	1	Sets pricing scheme (0 = provisioned, 1 = 'pay for what you use'
	Write Rate	500	10000	50	No. of write operations/sec that can be managed (N/A with On Demand = 1)
	Peak Capacity	0	1	0.1	% of capacity to reserve for peak loads (0 = no extra capacity, 1 = 100% of base capacity) (N/A with On Demand = 1)
	Peak Time	0	1	0.1	% of operating time that peaks can be handled (0 = No time, 1 = constant peak can be handled) (N/A with On Demand = 1)
	Read Rate	500	10000	50	No. of read operations/sec that can be managed (N/A with On Demand = 1)

Table 4.5: *All parameters in the Simulink blocks and their possible values*

4.3.2 Modeled Topologies

As described in Section 2.2, three groups of topologies were proposed to solve the scenario: `Simple`, `Stream`, and `Sophisticated`. Within each group, there are some variations of services to use, further dividing the groups. The modeled topologies are built very similarly inside Simulink to how the services would be set up in a live architecture. Figure 4.3 shows how `Stream_1` is modeled and set up within Simulink. The topologies are connected in the following way in Simulink:

- `Simple_1`:
  $Source \Rightarrow IoTCore \Rightarrow Fargate \Rightarrow S3$
- `Simple_2`:
  $Source \Rightarrow IoTCore \Rightarrow SQS \Rightarrow Lambda \Rightarrow S3$
- `Simple_3`:
  $Source \Rightarrow Kinesis \Rightarrow Fargate \Rightarrow S3$
- `Simple_4`:
  $Source \Rightarrow Kinesis \Rightarrow SQS \Rightarrow Lambda \Rightarrow S3$
- `Stream_1`:
  $Source \Rightarrow IoTCore \Rightarrow VinBatcher \Rightarrow DynamoDB \Rightarrow SQS \Rightarrow Fargate \Rightarrow S3$
- `Stream_2`:
  $Source \Rightarrow IoTCore \Rightarrow VinBatcher \Rightarrow DynamoDB \Rightarrow SQS \Rightarrow Lambda \Rightarrow S3$
- `Sophisticated_1`:
  $Source \Rightarrow IoTCore \Rightarrow VinChecker \Rightarrow DynamoDB \Rightarrow StepFunctions \Rightarrow JobQueue \Rightarrow EC2 \Rightarrow S3$

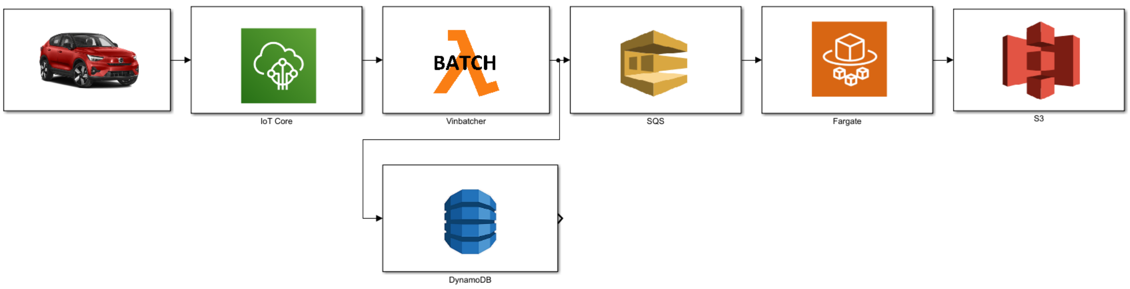

Figure 4.3: *The* `Stream_1` *topology in Simulink*

4.3.3 Modelling and Recording Quality Metrics

Through discussions with the architects behind the scenario and the proposed solutions, as well as through the interviews, the relevant quality attributes were decided to be Latency, Cost, Complexity, and Load Sensitivity. All the blocks' effects on the quality attributes are modeled in each block in Simulink.

Latency

Latency in the case of the simulations is the sum of all latency caused by the blocks divided over the number of data packets from the cars that have been computed. An example of where latency is created is if the Lambda block has been configured with insufficient memory or instances to keep up with the incoming data flow.

Cost

The cost is the price of running the system for the simulation time, which in this scenario is 15 minutes. The cost has been directly modeled from the pricing for each AWS service, which Amazon declares in a detailed manner on their website. What affects the cost of each block can be the amount of data it processes and/or stores, the number of invocations of the service, the time running, and the parameters' values.

Complexity

Complexity measures how many parameters and blocks a setup contains (not counting *Source*). For example: `Simple_1` has three components, and *Fargate* is the only block with parameters (two), giving it a total complexity of five.

Load Sensitivity

Load Sensitivity is the measure of how sensitive the Cost and Latency of the system are when the load increases. It is computed by running a second simulation, increasing the number of cars tenfold, and then comparing how much the Cost and Latency per car increase. Load Sensitivity value of 10 means that the Cost and Latency increase tenfold which means it scales linearly. If the Load Sensitivity is 0, the Cost and Latency are constant when increasing the load.

4.3.4 Simulating the Modeled Topologies

The simulations were run by loading the Simulink models with parameter values and recording their quality attributes after simulating the scenario. The parameter values were randomly chosen for 2000 simulations, and duplicate runs were excluded. Each simulation gives one row of data containing all its parameters and quality attribute measurements. Upon completion of all simulations, the total data frame consists of 12136 data rows, each with a unique architectural setup.

Summary

This chapter has presented the approach taken with simulating different topologies made up of components and parameters that solve the scenario posed in 2.2. By conducting simulations, a dataset could be created to be used as a basis for different data visualization approaches. The next chapter introduces these data visualizations and how practitioners can utilize them to support decision-making.

4.4 Supporting Decision Making

This section will present all analysis approaches and methods developed and how they can be applied in the use case on the data from the simulations. Subsection 4.4.1 explains how optimal samples from the simulation are chosen throughout the analysis. Subsections 4.4.2, 4.4.3, 4.4.4, and 4.4.5 will show visualization and analysis methods applied to the simulation data to discern between topologies. Subsections 4.4.6 and 4.4.7 will further deepen the analysis and move past the topologies slightly to look at the effect of parameter values in the quality attributes. Each subsection will also have a paragraph where the findings are summarized, and the values this brings to a practitioner are discussed.

4.4.1 Finding Optimized Samples with Pareto Front Optimization

As proposed in this thesis, the analysis of architectural tradeoffs requires a substantial amount of multivariate data. From the simulation data of the use case, there are over 12000 rows of data from seven different topologies with 20 different parameters. All simulation runs have been performed with random parameter values, and even though the parameter values are chosen from a suitable range of values, there is no guarantee that they will perform well. In the data from the simulations, the simulations are very spread out in terms of the recorded quality attributes, which creates the need for filtering the data.

For the use case, the architects at Volvo were interested in getting the architectural setups with good results and analyzing these. By applying Pareto Front Optimization with one or multiple quality attributes as targets and the objective of minimizing them it is possible to pick out the optimal samples from the data. Additionally, applying PFO continuously until a certain number of samples have been picked out can create data sets containing a certain percentage of optimal samples. For example, by running PFO until it has picked out 50 samples from 1000 samples in total yields a new data set containing the top 5% of the data samples.

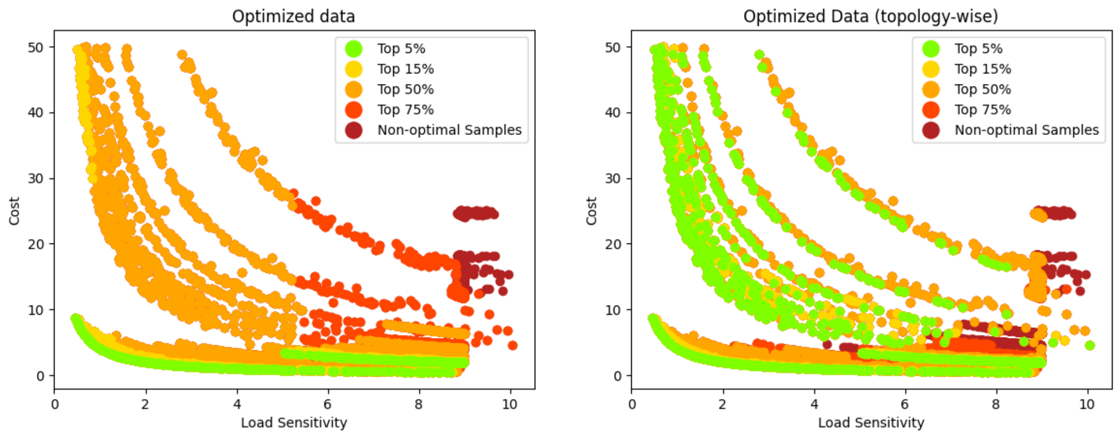

Figure 4.4: *Data from running PFO with minimizing Cost and Load Sensitivity as the objective on the simulation data with the colors of the samples showing what data set they are in*

Figure 4.4 shows PFO applied on the simulation data with *minimizing Cost and Load Sensitivity* as objective. The colors depict what data set the samples are in. For example, a green sample means they are in the top 5% of samples. The left figure shows the PFO run on all topologies simultaneously to get the optimal samples. However, in some of the analyses in this thesis, it is necessary to get the optimal samples per topology. This is done by separating the data based on topology, running the PFO for each topology, and then merging the different data frames. This means the top 5% optimal samples will contain the top 5% of all the seven topologies. The data from doing this using the same data and objective is shown on the right side in Figure 4.4.

Practical Implications

Pareto Front Optimization is crucial in this thesis's analysis and visualization techniques for two main reasons. It allows a practitioner to remove architectural setups that produce unwanted results from the data. By keeping for example 95% most optimal sample outliers can be removed ahead of subsequent analysis. Secondly, and most importantly, if a low percentage of optimal samples is kept, then a practitioner can rest assured that all samples are from satisfactory architectural setups. Using only good samples gives more useful information from the subsequent analysis and visualization methods, as seen in upcoming subsections.

4.4.2 Visualizing and Analyzing Tradeoffs Between Topologies using PCA

As previously explained, the data in this thesis is multivariate with many dimensions. In the simulation data, there is one dimension for what topology the sample is from, twenty different parameters, and four quality attributes per sample. To better understand, analyze and visualize this data, dimensionality reduction techniques can be applied. One technique used in previous research for this cause is PCA [6].

The PCA can explain trends in the data and tries to explain as much of the data as possible in fewer dimensions. From the data, a correlation matrix is calculated on which the PCA is performed. From plotting the loading scores of the PCA results, often called PCA plots, the correlations in the data are easier to discern.

When applied to the simulation data, the PCA and subsequent plots allow the possibility to see how the different topologies relate to the quality attributes. The left plot in Figure 4.5 shows the PCA plot of all samples, and the right plot shows the PCA results of the samples where only the 15% most optimal samples for each topology have been kept. The optimizations were performed with Pareto Front Optimization with the objective of minimizing all four quality attributes (*Latency*, *cost*, *complexity*, and *Load Sensitivity*).

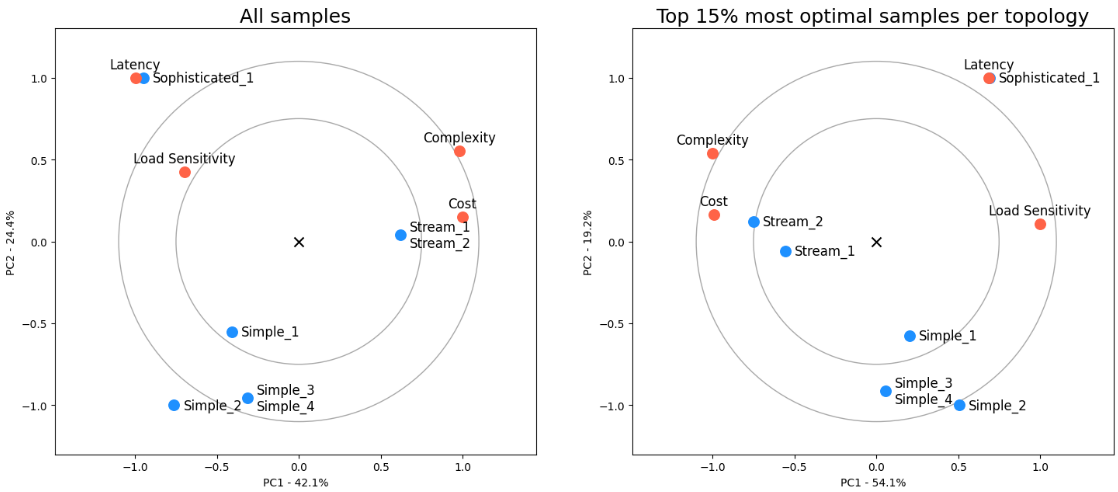

Figure 4.5: *PCA plots of the quality metrics and topologies showing the correlations between them in all of the samples (left) and in the data with only the top 15% of samples from each topology. The left plot describes 65.5% of the variance in the data, and the right plot explains 73.3%*

What can be seen in the PCA plots is that the topologies have the same general correlation and effect on the quality metrics regardless if the PCA is performed on the full data or only the optimal samples. This is shown by the two plots being almost mirrored on the x-axis. This means that the same correlations hold true for both PCA plots. However, the optimized data PCA has an easier time explaining the variance in the data (73.3% explained in the optimized samples and 65.5% in all samples using the first two PCs), pointing to the fact that the optimization creates a more uniform dataset. Furthermore, it can be observed that topologies in the same group create clusters, meaning that topologies from different groups of topologies perform dissimilarly from each other.

Furthermore, what can be observed in the plots is also that the `Simple` group has a negative correlation with *Cost* and *Complexity*, and a small positive correlation with *Load Sensitivity* and almost no correlation with *Latency*. The `Stream` group has a negative correlation with *Load Sensitivity* and *Latency*, and a positive correlation with *Cost* and *Complexity*. The `Sophisticated` group is negatively correlated with *Complexity* and *Cost* while having a positive correlation *Load Sensitivity* and a very strong positive correlation with *Latency*.

Practical Implications

PCA and loading score plots have been applied in related research on understanding tradeoff design spaces [5][6]. The plots in Figure 4.5 show how much information a practitioner can get from the data. On this abstraction level, the implications of the topologies on the quality attributes and how they differ can be seen. If all the practitioner were interested in was what topology is the best option, these PCA plots could be enough to find a factual decision basis. However, the PCA plots can be hard to interpret, a problem that will be addressed in the next subsection.

4.4.3 Visualizing Tradeoffs Between Topologies using Radar plots

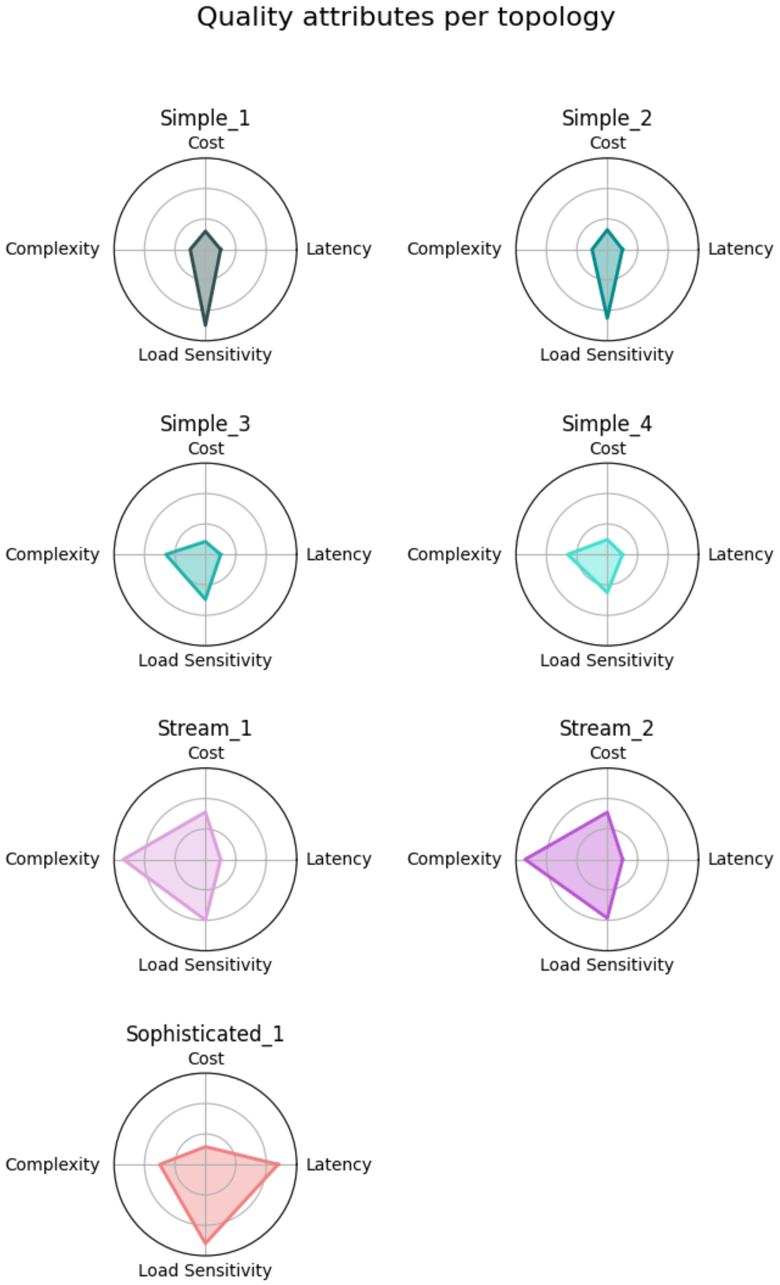

Figure 4.6: *Radar plots showing how each topology affects the quality attributes, with high values (meaning poor quality metric for the attribute) closer to the outer rim of the circle. The scale of each axis has been normalized to be comparable between the topologies*

PCA plots have a big disadvantage in being quite hard to read and analyze, especially to a less experienced eye. Another visualization tool that also has the capability of visualizing tradeoffs while also making it possible to compare different samples are radar plots.

Computing the mean values of the quality attributes for each topology respectively in the simulation data, making them all lie on the same scale, makes it possible to create radar plots which can be seen in Figure 4.6.

What can be seen in Figure 4.6 is that there exist two general, somewhat skewed, shapes: a kite as can be seen for `Simple_1, Simple_2, Stream_1,` and `Stream_2`. Or a trapezoid, as can be seen for `Simple_3, Simple_4,` and `Sophisticated_1`. The existence of such shapes shows that no topology minimizes all qualities and tradeoffs therefore exist, which supports the PCA. The kite shape indicates that the loss in quality is focused mainly on one quality, *Load Sensitivity* or *Complexity*. At the same time, the trapezoid indicates that there are two quality attributes where there are losses.

Practical Implications

The radar plots show the same results as the PCA plots in Figure 4.5 in the previous subsection but in a different way. Radar plots can be helpful for a practitioner who is not experienced in interpreting PCA plots or wants to condense the information in a more digestible visualization. Both the PCA plots and the radar plots do, by design, display general trends of the data. Other methods need to be explored for more specifics, which will be done in the next couple of subsections.

4.4.4 Visualizing Strengths in Topologies using Stacked Bar Charts

Both the PCA and radar plots make it possible to see strengths and weaknesses and show evidence that all topologies have some quality tradeoffs. However, both of these plots show general trends and do not rule out the possibility of a small number of samples still being optimal or non-optimal in some cases, even though its topology, in general, is not.

By turning the problem of choosing a topology on its head and instead looking at what topologies are good options for minimizing different sets of qualities, it is possible to see these types of cases. Figure 4.7 shows a stacked bar chart to visualize this. Each bar shows what topologies are found in the samples when selecting the top 15% most optimal samples in the data for different PFO targets. The height of each color shows how big a portion of the samples are from each topology. What target or targets the PFO has been performed with can be seen above each bar.

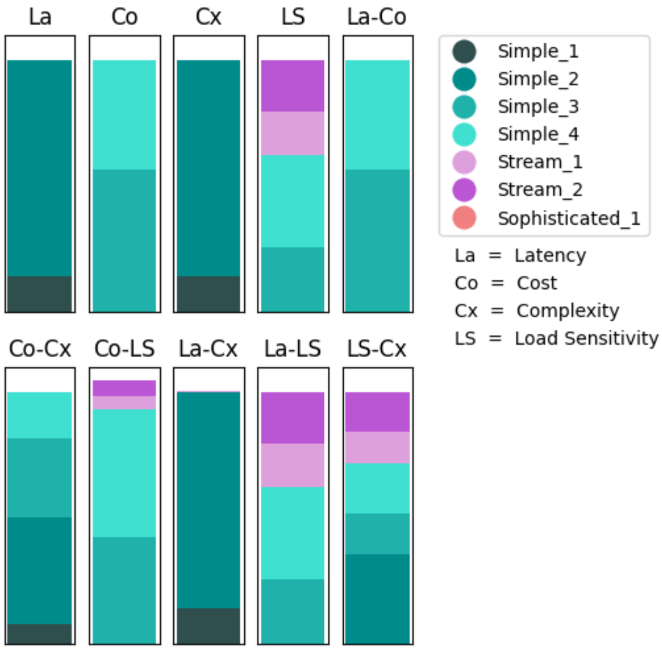

Figure 4.7: *Bar plots describing the occurrences of different topologies that are among the top 15% of optimal values when using the specified objectives in Pareto Front optimization (Objectives used are shown over each bar)*

What can be seen is that `Sophisticated` misses all targets as there are no samples from that topology in any of the optimizations. From the radar plots, it seems as `Sophisticated` is quite cost-effective, but it can be seen that in comparison to `Simple_3` and `Simple_4`, it is worse as the cost-optimized samples only consist of samples from these topologies. The plot also gives further evidence of the `Simple` group being superior in most cases. The `Stream` group of topologies also seems to be a good option when *Load Sensitivity* is an important concern.

Practical Implications

The stacked bar charts in Figure 4.7 complement the PCA and radar plots. In the data for this thesis, however, the bar charts do not provide any major revelations, except that `Sophisticated` performs badly. This is mainly caused by the fact that samples from the same topology perform very similarly and have no exceptional samples that differ greatly from the general case of the topology. This method does provide practitioners the possibility to find architectural setups that, in a good way, are performing very differently from their peers from the same topology.

4.4.5 Analyzing Driving Factors of Quality Metrics with Decision Flow Charts

Up to this point, the topologies have been the main dividing factor of the data. However, many different possible topologies exist, even though only seven are used in this study. Each topology can be seen as taking a decision in five steps: *Source*, *Transport*, *Intermediary*, *Processing*, and *Storage*. Therefore, the components used within a topology can be described as choosing one or multiple features per step. Table 4.6 breaks down each topology into its features. The scenario's steps and possible features can be seen in Figure 4.8, which also shows how the decisions correlate with the Cost and Latency metrics in the data.

	Source	Transport	Intermediary	Processing	Storage
Simple_1	Caching / No Caching	MQTT	No Intermediary	Auto-Scaling Containers	S3
Simple_2	Caching / No Caching	MQTT	Load Optimization	Serverless	S3
Simple_3	Caching / No Caching	HTTP	No Intermediary	Auto-Scaling Containers	S3
Simple_4	Caching / No Caching	HTTP	Load Optimization	Serverless	S3
Stream_1	Caching / No Caching	MQTT	Load Optimization, Job Batching	Auto-Scaling Containers	S3
Stream_2	Caching / No Caching	MQTT	Load Optimization, Job Batching	Serverless	S3
Sophisticated_1	Caching / No Caching	MQTT	Container Management	Containers	S3

Table 4.6: *Each topology broken down into its features*

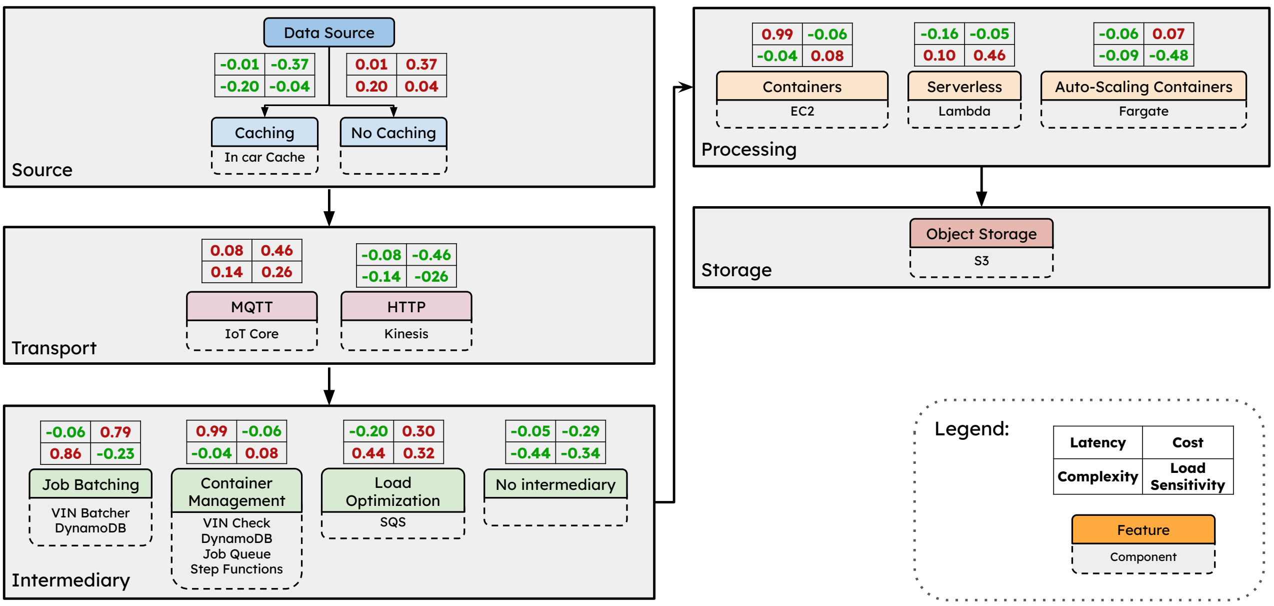

Figure 4.8: *Decision Flow chart describing what design decisions there are for each part of the architecture and their correlation with the quality attributes. A higher (red) number means that there is a positive correlation with the quality attribute, therefore increasing said attribute in the model. A lower (green) number means a negative correlation with the quality attribute, therefore decreasing the attribute in the model*

In Figure 4.8, it can be seen that the biggest culprits in the data in terms of *Cost* and *Latency* is the decision not to cache data in the car and the use of MQTT as the transport protocol. What can also be seen is that all *Intermediaries* entail a trade-off either through lowering the cost but increasing the latency or vice-versa. This gives further insight into the reasons behind why `Simple` and `Stream` outperforms

`Sophisticated` - they omit the *Container Management* and *Containers* features, which are the features that have the highest correlation with high latency.

Two topologies that stand out are `Simple_3` and `Simple_4`. This can partly be explained by the fact that they are combinations of features that minimize the positive correlation with the quality attributes overall. This is further proven in Figure 4.6 where they can be seen minimizing many of the qualities, which also is seen in Figure 4.7 where they appear in all optimizations but those of *Latency*, *Complexity* and *Latency-Complexity*. However, both topologies have some features that positively correlate with complexity. For example, `Simple_4` include the *Load Optimization* with a 0.30 correlation with cost but still appears in the cost-optimized solutions. This indicates that further analysis is needed to understand the underlying factors.

Practical Implications

The division of the topologies into steps and features, thus expressing the design space as the decision of different features for each step, analyzes quality implications a bit more granular. It makes it possible for practitioners to identify what causes the most significant effects on the quality attributes. If the data would include even more different features divided over a low amount of topologies, a practitioner might be able to identify new possible topologies that would minimize qualities even more. However, this division into features still can not fully explain the data and tradeoffs; an even more granular analysis is needed, which will be done in the upcoming section.

4.4.6 Visualizing and Analyzing Parameter Values using PCA

From the PCA plots, radar plots, and stacked bar chart presented in the previous sections, the data have largely been explained by its topologies and quality attributes. However, it is not only the topology that affects the quality attributes. The parameters of the components and their values greatly influence the performance and operation of the components.

Once again applying PCA to the simulation data but focusing on the parameters, the correlations between individual parameters and the correlations between parameters and quality attributes can be visualized. Figure 4.9 shows the results of a PCA performed on the top 15% samples from each topology in the simulation data.

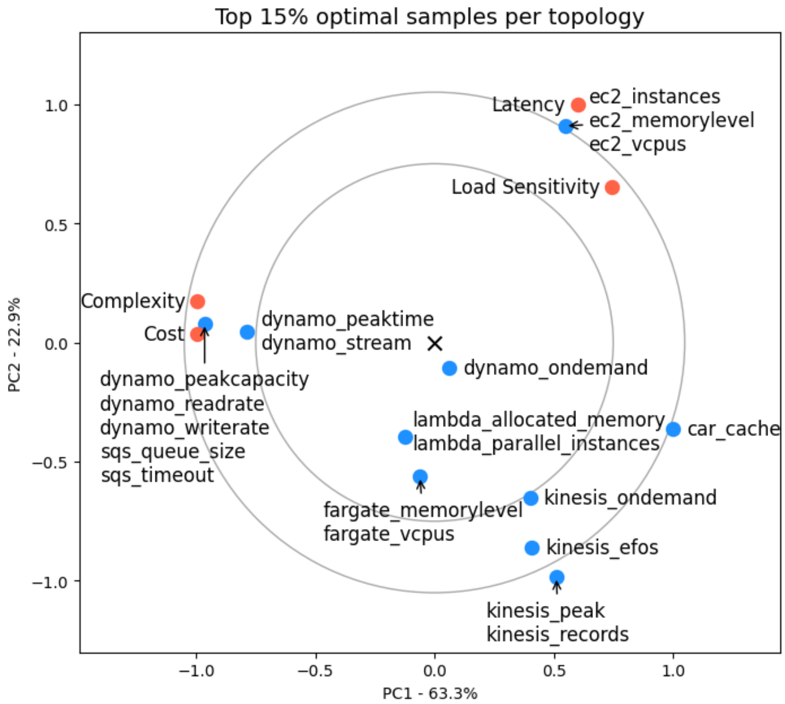

Figure 4.9: *PCA plot describing the correlation between parameters and quality attributes in the data*

What can be seen in Figure 4.9 is first that all quality variables lie far from the origin (marked by an X in the figure) which means that they're important for explaining the variance in the data. Secondly, all parameters that stem from the same component are tightly clustered together, which is quite explainable by the fact that if a component does not exist in a topology, all the parameter values will be zero. If the components exist, they will, in most cases, be non-zero. Therefore this PCA plot is better at explaining how the existence of components affects the quality attributes. It can be seen that EC2-parameters have a significant positive correlation with both load sensitivity and latency. Similarly, Dynamo- and SQS parameters strongly correlate with cost and complexity. On the contrary, all parameters in the lower right quadrant, car cache, and Kinesis-parameters negatively correlate with cost and complexity.

As shown previously in Figure 4.9, it is not very accurate in showing correlations for each parameter within the same component. To visualize these correlations, the scope of interest must be narrowed to a single topology. The PCA plot of doing the same PCA with the same optimization of top 15% optimal samples but this time on only `Simple_4` can be seen in Figure 4.10. What can be seen is that parameters in the same component are no longer as tightly clustered, and thus conclusions on the individual parameters can be drawn.

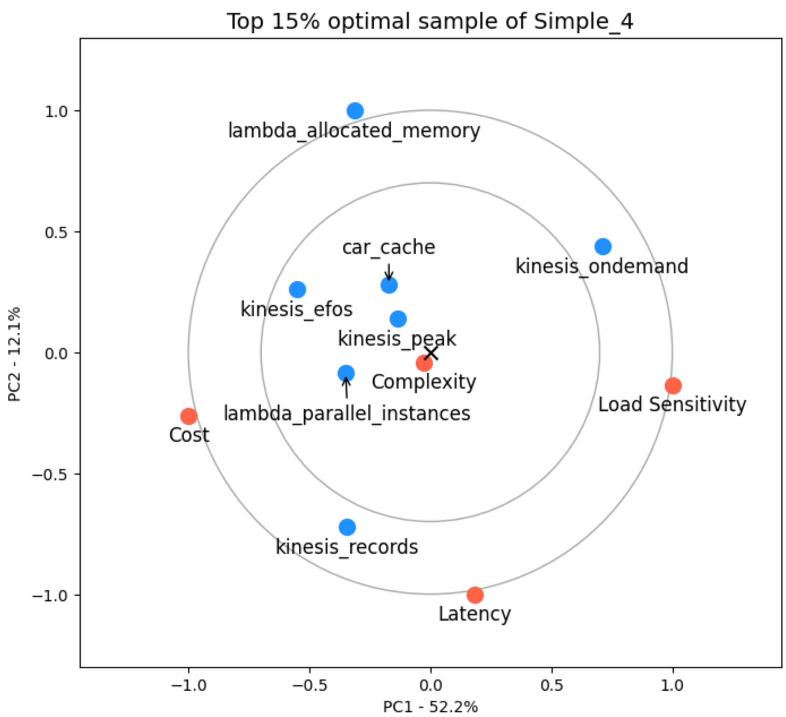

Figure 4.10: *PCA plot showing how the parameters in Topology Simple_4 affect the quality attributes*

The quality attributes of great importance for explaining the variance are cost, Latency, and Load Sensitivity. Recall that Complexity is the measure of the number of components and parameters which will stay the same overall samples from the same topology. Thus Complexity is at the origin of the plot. The parameters that have greater importance for explaining the variance are *lambda_allocated_memory*, *kinesis_ondemand*, and *kinesis_records* as they are placed farther from the origin than the other parameters. *kinesis_records* have a strong positive correlation with cost and latency while having close to zero correlation with load sensitivity. *kinesis_ondemand* on the contrary, has a strong negative correlation with cost, a strong positive correlation with load sensitivity, and almost a non-existent correlation with latency. One parameter stands out as being free of tradeoffs: *lambda_allocated_memory*. It only has a strong negative correlation with latency while having an almost non-existent correlation with load sensitivity and cost.

Translating the correlation means that increasing allocated memory to the Lambda service seems only to increase latency while having no other drawback. Thus, a low allocated memory seems to be the best option. Whether Kinesis is set to be on-demand or not seems to shift the topology to either being expensive but better at handling the load or less expensive but more sensitive to an increased load. Additionally, it seems like the number of records for Kinesis should be relatively low since it otherwise increases both cost and latency.

Practical Implications

The results shown in both Figure 4.9 and 4.10 view the data from a whole other direction than what has been done with the methods presented in the previous subsections. What a practitioner can get from analysis on this granular level is to get a good look at how all components of the architectures affect the quality and what tradeoffs they entail. These methods give the most information when a practitioner has narrowed down on a smaller subset of the data, for example, only top performers as in Figure 4.9 or a single topology as done in Figure 4.10. This analysis can serve many purposes, such as giving insight into what tradeoffs the components entail or what parameters of the components are most important for good quality.

The PCA plots correlate values based on whether they are high or low, which gives crucial indications of suitable parameter values. However, a practitioner wanting a concrete answer on good values can be left wanting more. This is addressed in the next subsection.

4.4.7 Analyzing Important Design Decisions and Threshold Values using DTL

The presented results thus far all explain general trends in the data. However, if a topology is being analyzed and the architects want more information on the driving factors, such as the most important parameters, and if there are any threshold values for parameters where the quality attributes change drastically, other analysis methods are needed.

By utilizing the learning part of decision tree learning (DTL), information on what threshold values for parameters there are can be retrieved from the data. The model will train on the complete data set, attempting to build a model that would classify a quality attribute based on all parameters and their values. After the training, the hyper-parameters can be retrieved as they will convey the information of interest.

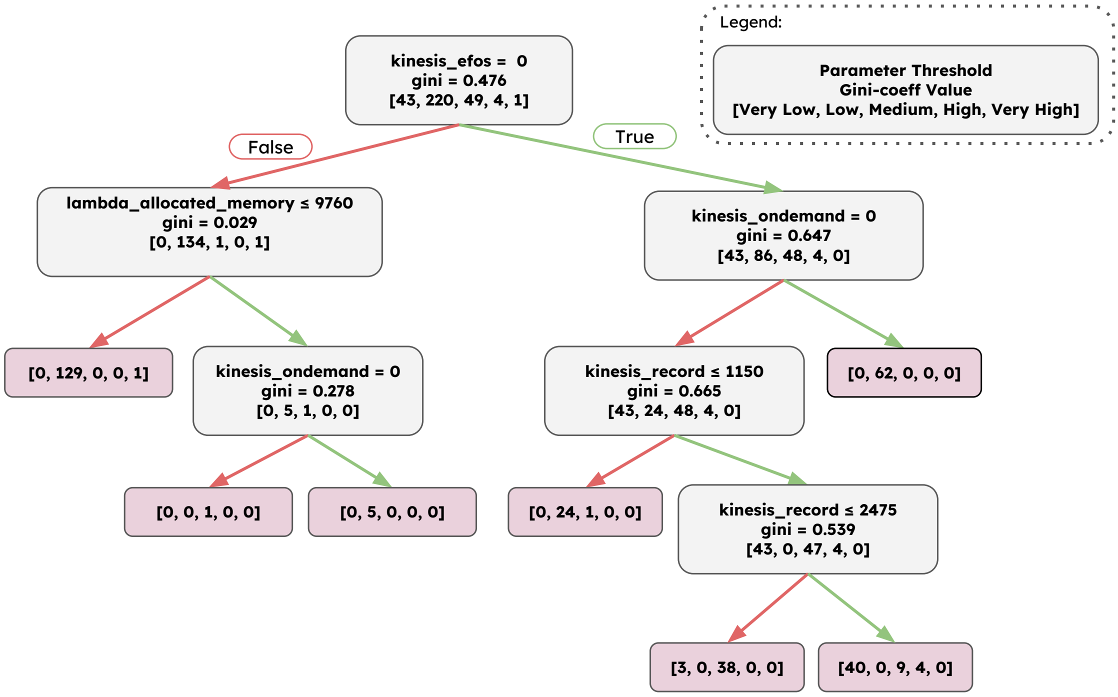

Figure 4.11: *Decision Tree for Topology Simple_4 showing the most important parameters and threshold values for classifying the samples into five Cost categories: Very Low, Low, Medium, High, Very High*

Figure 4.11 shows a DTL model that classifies the cost of samples into five classifications: Very Low, Low, Medium, High, and Very High, which are simply even sub-ranges of the range of cost in the data. The model has been trained on the same top 15% most optimal samples of topology `Simple_4`. The root node shows the classifications in the data and how each decision in the tree splits up the data until almost all samples have ended up together with samples from the same cost classification in the pink leaf nodes.

What can be seen in Figure 4.11 is that *kinesis_efos*, *lambda_allocated_memory*, *kinesis_ondemand*, and *kinesis_records* are the important parameters to classify the cost for `Simple_4`. The important values for getting Very Low cost are to have *kinesis_efos = 0*, *kinesis_ondemand = 1*, and *kinesis_records >= 2475*.

Practical Implications

As in Figure 4.11, the decision tree gives a practitioner tangible evidence of the important parameters. The same parameters found in the decision tree can be seen in the PCA plot in Figure 4.10 in Section 4.4.6 which shows that the analysis methods in large tell the same story but on different granularity levels. The decision tree not only conveys the most important parameters but also indicates what parameter values should be chosen for different quality levels. This can be used in a later stage of design when parameter values must be decided on. The practitioner can then get a factual basis for the decisions they make.

Summary

This section has introduced a plethora of different visualization techniques to be utilized by software architects to support decision-making. These visualization techniques can be used for various purposes, such as deciding between different topologies, understanding the strengths and weaknesses of each topology, driving factors of the behavior of topologies in the form of standout parameters, and exploring threshold values for a given topology. A workshop was conducted to evaluate the usefulness and understandability of each of these approaches, and the results are presented in the following chapter.

4.5 Workshop Findings

A final step for this thesis was to validate our results with architects to conclude the simulations, data analysis, and method of analyzing tradeoffs in a more data-driven approach. Since data visualization techniques have been proposed in research to support decision-making for software architects and additions to these techniques, this workshop intended to understand the usefulness and understandability for practitioners in the industry.

The plots and graphs in this paper were presented to three architects in a workshop. The architects were asked to speak their minds on what was presented, the usefulness of such an approach, and if the data analysis was insightful.

There were a few findings from this workshop. Firstly, PCA plots can be difficult to discern for an untrained eye and require further assistance from someone more trained in data analysis to guide them. As one of the architects put it:

> "When we come across something that looks like this, we usually try to avoid it."

However, after some discussion and explanation, the architects could grasp the PCA, which sparked discussions on the underlying causes of certain components and how they interacted with quality attributes.

In contrast to the PCA, the Radar plots were very well perceived by the practitioners for their ease of understanding. Discussions were sparked on the different topologies and how the results might be interpreted. One of the architects asked if it is possible to conclude that *Simple_3* and *Simple_4* of the topologies are the best and should therefore be chosen as the solution, based on the radar plots. This is the correct intuition (barring the fact that our simulations might be lacking details in the simpler models that might skew the results). This is a conclusion the architects could not easily draw from the PCA plots.

Furthermore, the decision tree was shown to the architects. Here, the practitioners would have liked to see more defining characteristics of the topology and what exact threshold they should choose for specific parameters. However, with our simulation data, it is not possible to draw such conclusions. As seen in Figure 4.11, the decisions would have to filter down into different cost categories to draw such conclusions. However, as can be seen, several decisions lead to the same categories, meaning that even if you follow the path of a less optimal solution (as supported by the decision tree), you might still end up in a low or very-low category. Since this is an artifact of our specific simulations, performing DTL to find threshold values seems lucrative for the practitioners to understand threshold values and important parameters for a topology.

When discussing the feasibility of the data analysis and how it can aid in decision-making, the architects were optimistic. One of the architects said the following:

> "There are so many parameters that you don't know how they will influence the system. When you have made four mistakes on different systems, then you might have learned to not do things that way. This approach could help to avoid those first four mistakes."

The architects also discussed how getting the simulated components as close to their real-world counterparts is important. We discussed our approach and explained that the cost models are basically real-world since this information is easily retrievable. Regarding other quality attributes, such as complexity, we explained that it would be up to the architects to evaluate the different topologies and their impact on the complexity metric. This was perceived as valuable, and it is possible to add more quality attributes, such as design cost, to the system and see how different topologies would affect such an attribute.

Lastly, the different topologies were discussed, and how they might be better comparable if redundancy (or some form of cost-penalty for data loss) gets included in the system. The current system shows that the *Simple* models are the cheapest (and generally best in terms of quality attributes). The architects validated this and acknowledged that this would also be true in a real-world implementation. However, the *Simple* solutions would have a large cost-penalty if redundancy was taken into account, and the amount of data loss (if the system goes down, for example) would be considered high by the architects. Whereas the more advanced model you choose, the more redundancy gets introduced into the system, and the risk of data loss gets mitigated at increasing levels.

5
Discussion and Summary

This chapter is dedicated to concretely answering each of the research questions posed in this thesis, and the conclusions are based on the results in Section 4. This thesis has introduced new approaches to analyzing tradeoffs and how practitioners in the industry can utilize these to make well-founded architectural decisions. In the results, there are several aspects of interest to discuss further, so this chapter also intends to discuss these findings and related topics. Furthermore, this chapter also gives ideas on what topics to explore further in future research.

5.1 Realism of Simulator and Applicability

Firstly, this thesis' results are based on the simulator that was created. As stated, the simulator was never intended to work as a guide in making any real-world decisions for any system. Rather, the resulting data could be used to explore different data analysis and visualization approaches. Practitioners have evaluated these methods, understanding that the exact data cannot be trusted. Rather, the methods themselves are what is of importance. It should be noted that care was given to the simulator to reach a good resemblance to the real-world counterpart, and certain aspects, such as the relationship between quality attributes, were verified manually against knowledge from the architects of how similar systems behave.

If this project was to be reproduced, some aspects that could be made differently were identified. Firstly, a more modular simulation would possibly be more beneficial for the software architects. Since the topologies analyzed in this thesis were given by software architects that had been pre-identified, many of the interesting aspects had already been identified. As such, there were not many surprises when presenting the topological results to the architects. Conducting simulations when the topologies had been pre-identified meant that it was not possible to let the data analysis results give a lot of insight into possible topologies. Instead, it served as a means to support the software architects' preconceptions. It might have been more interesting not to lock onto such topologies and instead perform permutations, including every possible topology and identify feasible solutions regarding preferences and legal requirements afterward.

Not everything of interest to the software architects, as explored in the interview study, could be modeled in this thesis. In some cases, it's due to the limitations stated for this thesis, and in some cases, modeling some qualities or factors of the architectures is impossible or infeasible. This means that no matter how thorough the analysis is, architects still need to weigh the findings against previous experience and knowledge. For example, they can weigh in on the features that `Sophisticated` brings against the increased cost and complexity and, from there, make decisions. This indicates that a tool for analyzing tradeoffs complements the workflow. This became apparent in the workshop, where architects mentioned that the tool likely would serve as further proof if it confirmed suspicions about a tradeoff. However, it only raises a question for further discussion and examination if it contradicts the practitioners' beliefs. This can indicate the approaches lifted in this thesis not being reliable enough, but it still offers a useful insight into the usefulness and its justification as a useful tool. Continued work on evaluation analysis could make practitioners more susceptible to such approaches.

A more strict evaluation of the models could be conducted by implementing a real version. This could also help gather more true-to-life data supporting more accurate simulations. The largest benefit would be to see performance and latency. In a cloud context, unexpected behavior is sometimes not explicitly stated in the documentation but can show itself to practitioners when implementing a system. An example of this can be how latency could be very high for a solution with a single virtual machine. It might be the case that the threshold for how much data is allocated to a container before starting up a new container is very close to being reached. Starting up another container manually would increase the cost a bit but significantly reduce latency. Such behaviors are absent in our simulations, but further improvements could be made.

5.2 Future Research

Currently, the simulator only includes especially interesting quality attributes, as supported by the interview study. However, several other quality attributes have been discovered to be of importance to different software architectures. One such example that was lifted is that the current simulator results do not consider what functionality each of the topologies entails. If one was to take any takeaway from the results currently, it is that the `Simple` topologies are superior in almost every aspect. However, for their real world-counterpart, the more advanced models `Stream` and `Sophisticated` also bring additional features that are simply not available in the simpler models. The `Sophisticated` topology makes it possible to batch data that originates from the same vehicle and decide how the calculations on that specific data are performed. And so if that is a desirable trait, maybe due to specific circumstances or legal requirements, then this is not conveyed in the results.

One important aspect brought forward in the interview study is that any tool or approach can not be disruptive to the current workflow. Since the time spent developing and analyzing architectures equates to higher development costs, it is important to efficiently support decision-making without taking up too much time or effort for the architects. The results of this thesis show the feasibility of utilizing visualization techniques such as radar plots and decision trees as preferred over PCA plots.

However, since a simulator had to be created to yield these results, creating these visualizations would not be feasible in a real-world industry setting. Luckily, there is still hope since this thesis also introduces a library for components in Simulink that can be assembled in many ways. Continued work on improving and extending this library could lead to a path where topologies could be tested quickly and efficiently to a satisfactory level to support some sort of decision. Whether or not this decision will guide which topology to choose, as a way to tinker with optimal parameter setup or any other way, remains to be seen.

This thesis has been conducted using cloud components, specifically from AWS. However, most of the interview subjects in the study were not used to developing in and for the cloud. In practice, most software architects are more concerned with less abstract issues, such as code complexity, function calls, micro-services, or API calls. This thesis does not currently cover an interesting aspect: how these would influence quality attributes. To discuss the concerns regarding tradeoffs, the interviewees were instructed to imagine issues commonly found within their domain. In future research, it could be of interest to use static code analysis tools to gain more specific data to support the simulator. One of the largest issues with cloud components is that the only available model that can be accurately depicted from documentation currently is the cost model. Analyzing code components on a lower level, it might be possible to invoke scenarios such as different bandwidth levels to simulate latency more accurately. This would also allow the addition of other quality metrics such as packet loss, number of retries, and reliability overall.

Hopefully, the main findings of this research will spark the interest of new, similar-style projects, which could eventually lead to tradeoff analysis approaches that are more generalizable and that can one day be part of the design process of major software architectural projects.

5.3 RQ1 - What are the challenges with understanding architectural tradeoffs in a cloud-based vehicular software system?

This thesis included an interview study to acknowledge practitioners' challenges when making tradeoff decisions. The main takeaways are as follows:

- Complex systems include too many parts to evaluate optimal solutions and setups efficiently
- The time to develop is required to remain low to decrease cost. Thus making it impractical to analyze details in systems
- Understanding the impact of tradeoffs is difficult to anticipate and is easily underestimated
- Constantly changing requirements from stakeholders and systems makes it difficult to get a grasp on what to prioritize and to predict fallacies in the architecture under design

- Software architects need to align with hardware constraints that are not always foreseeable and posed by other teams' software architectural decisions
- Currently, in practice, there is no way to visualize tradeoff design spaces

These findings confirm previous studies' findings and have been covered as the basis for why analysis methods such as ATAM are valuable to practitioners [8]. However, in relation to the visualization techniques that have been previously identified and used in research, these results serve as a basis for what practitioners would value in such analysis methods, which has not been studied previously [5]. Grasping the foundations as to why particular techniques would be valuable and to what extent can encompass future research as to what objectives to focus on and what to avoid.

5.4 RQ2 - What are current practices when making decisions in design spaces for a cloud-based vehicular software system?

To grasp the current practices employed in the industry, workflow-specific questions were posed to practitioners during the interview study. And can be summarized as follows:

- Software architects work together as a team to develop a solution. The best-proposed solution, as determined by the team, eventually gets implemented
- Visualization tools for sketching are utilized
- Decision-making relies heavily on experiences and previous knowledge
- Architectural design is mostly an iterative process where initial decisions are often reconsidered. However, not based on data-driven approaches
- No major run time data gets collected during the design phase of software architectures

Since this thesis to the best of our knowledge is the first time visualization techniques have been implemented and evaluated in the industry to deduce tradeoffs and their impact on quality attributes in software systems, it was essential to properly understand how such approaches could be valuable in the context of the current workflow. These results are not surprising, barring perhaps the fact that experience is such a key deciding factor on what architecture gets proposed and implemented, which is something that, to our knowledge, has not been studied in research. Since the current software architects appear to utilize very few tools, including data visualization techniques could interest both experienced and less experienced practitioners. For experienced software architects, the results could serve as a way to double-check intuition, as well as give guidance on important parameters and setup. For less experienced practitioners, the techniques could be used as a basis for which topologies appear to perform the best for a given objective, such as minimizing cost. Receiving any insight could spark new ideas on topologies and which components to include.

5.5 RQ3 - How can architectural tradeoffs be elicited and characterized to support decision-making in a cloud-based context?

There is a need for a lot of data on different architectures with recorded quality metrics to elicit architectural tradeoffs. These can be recorded using prototypes or live architectures, but this thesis proposes using a simulator if such data is unavailable.

When the data has been acquired, several techniques have been proposed to characterize and support decision-making around tradeoffs. These include:

- PCA plots
- Pareto Front Optimization
- Radar plots
- Decision trees
- Stacked bar charts
- Decision flow chart

These methods have been evaluated and are deemed valuable to software architects to varying degrees. The proposed methods equip the architects with a more granular analysis of the tradeoffs, performance of individual components, and parameter selection than previously available in approaches such as ATAM [8]. Compared to previous research, PCA and PFO appear to be some of the less preferred visualization techniques by practitioners in the industry [5]. This also points to the importance of implementing research in a real-world setting. When something is considered valuable to researchers with a lot of experience in understanding dimension-reduction techniques such as PCA, it does not necessarily translate to practitioners feeling the same way, especially if the understanding of such techniques gets in the way of the workflow for the practitioners. This thesis concludes that the proposed methods are most valuable if the underlying data is trustworthy and relatively lightweight in time to set up and analyze.

6 Conclusion

In conclusion, this thesis focused on software architecture tradeoff analysis methods in the context of cloud-based vehicular systems. The research aimed to address the challenges faced by software architects in understanding and navigating the tradeoffs associated with design decisions in such systems. The study utilized a real-world use case in the vehicle industry, simulating software architectures developed by architects at Volvo Cars. The data collected through this simulation was then subjected to architectural tradeoff analysis using various visualization techniques, including Principal Component Analysis (PCA), Decision Tree Learning (DTL), PCA loading score plots, and radar plots.

The findings of this research indicate that the analysis and visualizations provided valuable insights for software architects in making well-founded design decisions and gaining a better understanding of the tradeoffs in software architecture. The techniques employed in this study enabled a more granular analysis of tradeoffs, the performance of individual components, and parameter selection compared to existing approaches.

The challenges identified in understanding architectural tradeoffs in cloud-based vehicular software systems include the complexity of systems, the need for efficient decision-making processes, the difficulty in anticipating tradeoff impacts, changing requirements, alignment with hardware constraints, and the lack of visualization tools for tradeoff design spaces. Understanding and addressing these challenges through effective analysis methods is crucial for supporting architects in their decision-making processes.

The current practices in the industry rely heavily on experiences and previous knowledge, iterative design processes, and limited use of visualization tools. The results of this research highlight the potential value of visualization techniques, such as radar plots and decision trees, in supporting decision-making for both experienced and less experienced practitioners.

Several aspects for future research and improvement were identified. The simulation could be made more modular to allow for a wider exploration of topologies and a more comprehensive analysis of feasible solutions. Additionally, incorporating additional quality attributes and considering the impact of functionality in different topologies would enhance the simulation's accuracy and relevance even more.

Bibliography

[1] F. Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif, "Quality characteristics for software architecture." *The Journal of Object Technology*, vol. 2, p. 133, 2003.

[2] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203).* IEEE, 1998, pp. 308–317.

[3] P. Kruchten, R. Capilla, and J. C. Duenas, "The decision view's role in software architecture practice," *IEEE software*, vol. 26, no. 2, pp. 36–42, 2009.

[4] T. Kostelijk, "Misleading architecting tradeoffs [dvd hard-disk architecture tradeoff analysis method]," *Computer*, vol. 38, no. 5, pp. 20–26, 2005.

[5] J. Cámara, M. Silva, D. Garlan, and B. Schmerl, "Explaining architectural design tradeoff spaces: A machine learning approach," in *Software Architecture: 15th European Conference, ECSA 2021, Virtual Event, Sweden, September 13-17, 2021, Proceedings.* Springer, 2021.

[6] R. Wohlrab, J. Cámara, D. Garlan, and B. Schmerl, "Explaining quality attribute tradeoffs in automated planning for self-adaptive systems," *Journal of Systems and Software*, vol. 198, 2023.

[7] D. Sobhy, R. Bahsoon, L. Minku, and R. Kazman, "Evaluation of software architectures under uncertainty: a systematic literature review," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–50, 2021.

[8] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and S. Carriere, "The architecture tradeoff analysis method. in proceedings of the 4th international conference on engineering of complex computer systems. montery," 1998.

[9] Business 2 Community, "How big is aws?" https://www.business2community.com/cloud-computing/how-big-is-aws-02214837, accessed 2023-05-09.

[10] Volvo Car Corporation, "Volvo cars - our story," https://www.volvocars.com/intl/v/our-story, accessed 2023-05-09.

[11] ——, "Future volvo cars to run on volvo operating system as company takes software development in-house," https://www.media.volvocars.com/global/en-gb/media/pressreleases/283545, accessed 2023-05-09.

[12] ——, "Volvo cars to be fully electric by 2030," https://www.media.volvocars.com/global/en-gb/media/pressreleases/277409/volvo-cars-to-be-fully-electric-by-2030, accessed 2023-05-09.

[13] ——, "Volvo cars app," https://www.volvocars.com/se/v/volvo-cars-app, accessed 2023-05-10.

[14] ISO25000, “Iso/iec 25010,” https://iso25000.com/index.php/en/iso-25000-standards/iso-25010, accessed 2023-05-09.

[15] H. Abdi and L. J. Williams, “Principal component analysis,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, pp. 433–459, 7 2010.

[16] A. Tharwat, “Principal component analysis - a tutorial,” *International Journal of Applied Pattern Recognition*, vol. 3, p. 197, 2016.

[17] P. Ngatchou, A. Zarei, and A. El-Sharkawi, “Pareto multi objective optimization,” in *Proceedings of the 13th international conference on, intelligent systems application to power systems.* IEEE, 2005, pp. 84–91.

[18] Z. Liang, G. Zhang, J. X. Huang, and Q. V. Hu, “Deep learning for healthcare decision making with emrs,” in *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM).* IEEE, 2014, pp. 556–559.

[19] Z. Zhang, “Applications of the decision tree in business field,” in *2021 3rd International Conference on Economic Management and Cultural Industry (ICEMCI 2021).* Atlantis Press, 2021, pp. 926–929.

[20] S. Suthaharan and S. Suthaharan, “Decision tree learning,” *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*, pp. 237–269, 2016.

[21] S. B. Kotsiantis, “Decision trees: a recent overview,” *Artificial Intelligence Review*, vol. 39, pp. 261–283, 4 2013.

[22] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice.* Addison-Wesley Professional, 2003.

[23] A. R. Hevner, “A three cycle view of design science research,” *Scandinavian Journal of Information Systems*, vol. 19, 2007.

[24] MathWorks, “Simulink - simulation and model-based design,” https://se.mathworks.com/help/simulink/, accessed 2023-05-09.

[25] W. McKinney, *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython.* " O’Reilly Media, Inc.", 2012.

[26] pypi.org, “paretoset,” https://pypi.org/project/paretoset, accessed 2023-05-09.

[27] SciKit-Learn, “Decision trees,” https://scikit-learn.org/stable/modules/tree.html, accessed 2023-05-09.

[28] ——, “Principal component analysis (pca),” https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html, accessed 2023-05-09.

[29] matplotlib, “Principal component analysis (pca),” https://matplotlib.org/, accessed 2023-05-09.

[30] Plotly, “Plotly open source graphing library for python,” https://plotly.com/python/, accessed 2023-05-09.

[31] R. Patel and B. Davidsson, *Forskningsmetodikens grunder.* Studentlitteratur, 2011.

A

Appendix 1

Interview Guide

- **Opening Remarks and Introduction**
  - Tradeoffs in our case means that focusing on one aspect (quality attribute/aspect), will affect the outcome of another aspect or attribute.
  - Design choices means different design setups, choices within and between architectures
  - We are looking to gather information regarding your work in the design space, which means an initial design where you have many different choices available and you have to navigate in order to reach an architecture that solves the problem.
  - ☐ Who are you, what's your role at Volvo - How long worked with SE and Architecture
- **Part 1 - Tradeoffs**
  - ☐ Explain a recent architecture you worked on that has been implemented?
  - ☐ Can you recall a specific architectural problem you overcame?
    - ☐ Do you have more examples?
  - ☐ Can you name some common tradeoffs you encounter often? (Explain tradeoff in our terms)
    - ☐ When did you encounter them?
    - ☐ Explain the architectural problem briefly?
    - ☐ Did you account for the tradeoff initially, or were they discovered?
      - ☐ Did: Exaggerated? Downplayed? Surprised?
      - ☐ Did not: How were they discovered? Why wasn't it thought of before?
    - ☐ Was the tradeoff found during design or implementation?
  - ☐ Generally, what are the some biggest challenges regarding understanding tradeoffs early on in architectural design in your opinion?
  - ☐ What are the largest factors in designing good architecture in regards to tradeoffs in your opinion?
  - ☐ Can you imagine a workflow or tool that can make the decisions made around trade offs easier?
    - ☐ What characteristics should a workflow contain/enact to make decision making easier?
  - ☐ Propose our idea - Cater this to your area. Looking into a way to use data that you have on multiple architectures or setups of solving the same

issue for the same kind of scenario. And applying analytical approaches to find a way to visualize tradeoffs to developers or architects in a way that's digestible. Hopefully this visualization will show how different components relate to each other and point towards the most important things to consider in regards to the tradeoffs.
  - ☐ Do you have any thoughts?
  - ☐ More data-driven decisions beneficial?

- **Part 2 - Decision-making and Workflow**
  - ☐ How do you start when trying to create an architecture?
  - ☐ Who is involved, what are the roles?
  - ☐ Technical aids- software, analysis, modeling?
  - ☐ Can you visualize the design space in the design phase?
    - ☐ What are the difficulties?
      - ☐ Any others?
  - ☐ There are often several ways of implementing an idea - Who gets the final say? (Data-driven, democratic or hierarchical?)
  - ☐ Is something lacking for you to make a good decision? (Comfortable in decisions made/Something to lean on?)
  - ☐ Is there some iterative factor? Do you ever go back to the initial design phase?
  - ☐ What's the factor for making certain decisions infeasible - Cost? Quality? Performance? Time-to-implement?
  - ☐ Are there any factor that are not important at all?

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY