Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

How to Strangle Systematically: An Approach and Case Study for the Continuous Evolution of Monoliths to Microservices

Valentin Seifermann

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Dr. Justus Bogner, Jonas Fritzsch, Daniel Weidle (Adesso SE)
Commenced:	April 29, 2021
Completed:	October 29, 2021

Abstract

In today's enterprise IT, a growing number of applications are being developed based on microservice architectures to meet enterprise requirements for highly scaled, highly available and resilient systems running in cloud environments. However, monolithic architectures have been the traditional way of developing applications for many years, which results in IT environments still consisting of applications based on these types of architectures. To overcome the limitations of these monolithic architectures, microservice migrations are performed with the goal of gradually decomposing the monolith into independent services at granular levels. In order to achieve this process, different decomposition approaches are presented in various publications, often focusing only on the technical aspects of a microservice migration. However, during this long-term process, various factors occur on different dimensions that are not only technical in nature. These non-functional factors are often not considered in existing publications. This leaves software engineers in need of a systematic migration guide on how to proceed with a continuous migration that also takes into account the decision-making processes based on the factors that influence the migration process.

To address this lack on methodological guidance, this work provides a systematic approach for a continuous and gradual evolution of monolith to microservices based on the strangler pattern. The approach is based on interviews with software engineers and project managers who have participated in microservice migrations. Based on these experiences, the influencing factors that occur during the various steps of a migration are identified. Moreover, the decisions made based on these factors are considered in order to provide a decision-making guidance.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

API Application Programming Interface. 20

DDD Domain-Driven Design. 23

DP Decision Points. 7

MSA microservice architecture. 15

SOA service-oriented architecture. 15

1 Introduction

1.1 Motivation

Over the last decade, the design of enterprise applications based on microservices has been established as an architectural style for developing modern applications. In addition, the pace and dynamism in the evolution of container technologies and orchestration platforms has also contributed to the growing popularity of the microservice architecture (MSA). As a type of service-oriented architecture (SOA), microservices are independently deployable and loosely coupled services running in an isolated environment and communicating over lightweight mechanisms [New15]. These characteristics enable better scalability, maintainability and increased productivity, as microservices can be developed and deployed independently, also making them a perfect choice for elastic cloud environments and continuous delivery [NMMA16]. But also for applications where scalability is not a priority, observations have shown that microservices are proving to be a promising architectural style.

However, a large majority of industrial software systems are long-lived applications, typically based on a monolithic architecture, which are often complex by nature and thus require extensive maintenance as the code base has grown over the years. Also with the shift from on-premise environments to elastic cloud infrastructures, the traditional monoliths are not able to fully benefit from the cloud capabilities. They can not be scaled on the module level and therefore have to be duplicated as a whole [Luk18]. In contrast, MSA is considered native to the cloud by making efficient use of provisioned resources. Moreover, monoliths are partially difficult to organize around business capabilities. The high complexity of the application requires more effort implementing new features on time and fundamental changes involving larger parts of the application leading to a larger time-to-market and a low evolvability. Under these circumstances, monolithic applications pose serious challenges in the IT world.

To overcome these shortcomings of the monolithic architecture, migrating towards MSA is often a viable option. Nevertheless, Migrating existing legacy software environments to microservices is considered a difficult task, as various challenges have to be addressed. They range from technical to non-technical factors, with organizations needing to adapt accordingly to develop and operate the new architecture. Despite the many challenges, various companies in the software engineering industry have succeeded in a complete rewrite of their applications which is considered very expensive and time-consuming [HS17]. To avoid a complete re-building of the application a refactoring of the monolithic application is another option. This also poses various challenges and is therefore considered as a non-trivial and long-lasting process. To accomplish this process of incrementally decomposing the monolith into independent services on granular levels, software architects are in need of selecting a suitable strategy and refactoring approach [FBZW19]. Therefore, various publications propose different migration approaches and techniques, which also have potentials

for industrial adoption [FML17]. This includes the use of Domain Driven Design with bounded context-based decomposition, static code and dynamic runtime analysis as well as refactoring based on non-functional requirements.

In this context, the strangler fig pattern has been proposed as an useful approach for a continuous migration where the monolith is incrementally broken down into individual microservices [LML20] [New19]. Most of the academic literature provides migration processes using different identification and decomposition techniques to implement an one-time service cut. Despite these often technical approaches, the various influencing factors and challenges during a long-term migration are not fully considered. As a result, the approaches are not sufficiently satisfying in supporting the involved developers and architects. This lack on methodological guidance on the refactoring process has already been revealed as the existing approaches still give engineers very little actionable guidance [KH18]. Hence, there is a need for a systematic migration guideline with methodological guidance on how to proceed in a continuous migration also considering factors that influence the migration process.

1.2 Goals

To overcome the lack on methodological guidance, this work aims to provide a systematic approach for a continuous evolution of monolith to microservices based on the stranger pattern. By providing an actionable guide for addressing the challenges encountered during the evolution, the approach is designed to assist stakeholders in the decision-making process during the various phases of the migration.Therefore, the influencing factors that occur during the various steps of a migration are identified. In addition, the decisions made on the basis of these factors are considered in order to provide a decision-making guidance. The information required to develop this approach is derived from existing literature and experience from migration projects in the software development industry. Therefore, interviews have been conducted with various stakeholders who have already participated in microservice migrations.

1.3 Industry partner

The thesis will be conducted in collaboration with the German IT service provider Adesso SE. The company provides consulting and IT services for various business sectors. This includes insurance, financial, healthcare sectors as well as lottery companies and the automotive industry. The core competencies range from the typical Java tech stack to modern microservice and cloud technologies. In addition to greenfield projects, Adesso also carried out various legacy software modernisation projects to make systems cloud-ready using microservices. The software engineers at Adesso also encountered problems where they did not know how to proceed at the respective steps and which decision should be made when various factors occurred at different levels which were not always of a technical nature. This often led to a lot of trial and error, which had a negative impact on the course of the migration. In this context, academic research in the field of microservice migrations has already been successfully conducted together with Adesso, which, also addresses only technical aspects of the migration. [KZH+20].

1.4 Thesis structure

The structure of the thesis will be presented as follows:

Chapter 2 (Fundamentals) provides basic information to understand the topic of this research. This includes approaches used for microservice migrations as well as the concepts and technologies that are relevant to the case study. Moreover, existing researches related to the topic will be discussed.

Chapter 3 (Methodology) describes the overall methodology of this work including the research questions and research methods as part of the study design.

Chapter 4 (Results & Discussion) presents and discusses the results for each research question. This also includes the elaborated migration.

Chapter 5 (Conclusion) summarizes the results of this work and provides directions for future work on this topic.

2 Fundamentals

This chapter provides fundamentals for understanding the concepts and approaches related to microservice architectures and microservice migrations, on which the approach is also built. Moreover, related work regarding the topic of this research is also provided. Section 2.1 provides basic knowledge of the microservices paradigm with respect to how it differs from monoliths. Section 2.2 provides fundamentals on microservice migrations. Furthermore, patterns used for microservice migrations and relevant to the elaborated migration are explained. Section 2.3 discusses the work related to the thesis topic.

2.1 Microservices

Applications with monolithic architectures are built in a single block based on different layer and a single database [DGL+16]. In contrast, microservice architectures which are inspired by service-oriented computing, consist of small loosely coupled and independent services [New15]. Figure 2.1 shows the different architectures and the scope of a change in functionality. Due to the loose coupling and the self-contained nature of microservices, code changes are often limited to a specific microservice. Thus, making changes in a microservice does not affect internals of its peers []. This ability also allows for independent deployability. In contrast, a change in a traditional three-tier architecture, typical of monolithic systems, implies changes in the other parts of the system. These changes across process boundaries are more expensive as changes inside a single service [New19]. Moreover, microservices can be scaled individually while monoliythic systems have to be duplicated as a whole as their modules cannot be executed independently. Thus, microservice architectures are suitable for cloud environments and container orchestration platforms such as Kubernetes, since only the services that require more resources need to be scaled and therefore efficiently utilize the resources provided. Microservices also enable high speed delivery by developing in small, agile cross-functional teams, resulting in new features being delivered more frequently [BHJ16]. As the teams are able to develop independently, the dependencies between the teams are low, which also leads to a faster production releases [NMMA16].

2.2 Microservice migration

Patterns have been around for a long time in the field of software development, providing reusable solutions to common problems in regard to the software architecture and software design. Various Patterns also exist for microservices migrations and monolith refactorings to tackle common problems and challenges.

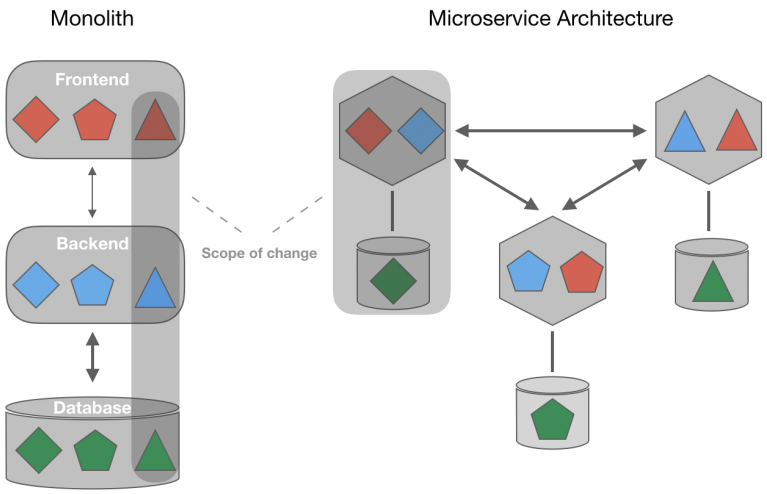

Figure 2.1: Monolithic and microservice architecture [New19] [FL14]

2.2.1 Migration patterns

Strangler fig pattern

One of these patterns is the strangler fig pattern used for incremental refactorings. First introduced by Martin Fowler, the pattern is inspired by a strangler plant that slowly wraps itself around an existing tree trunk until it kills the tree that was its host, leaving only the original form of the tree [Fow04]. A similar approach can also be used for an incremental migration to microservices around the edges of the legacy system where the microservices and the initial system coexist. Individual services are gradually extracted and can also be removed from the monolith after successful completion leaving only a shrunken monolith. In this context, the monolith can even be completely decomposed into microservices. By using this approach, various risks are reduced because the monolith remains functional and the operation is not restricted. In addition, each step in which a functionality is extracted into a microservice is reversible. This provides the possibility to stop the migration or roll back to a certain point. The pattern also supports frequent releases that allow continuous feedback on the effectiveness of the migration. According to Newmann [New19], the process of applying the strangler fig pattern is divided into three different steps which are displayed in Figure 2.2. This example shows a monolith that provides simplified accounting and inventory functionalities. The first step is to identify the possible service candidate, which in this case is the billing functionality. After that, the functionality to be extracted is moved to a microservice. Once the migration process is completed, incoming requests are routed to the microservice instead of hitting the monolith directly. By using this proxy mechanism, the functionality of the microservice can be completed and validated upon time, while also ensuring a uninterrupted, secure and reliable operation of the system including the possibility of a rollback. To ensure proper proxying, an HTTP proxy and Application Programming Interface (API) routing must be set up within the system environment at the beginning to process the incoming HTTP requests. For example, Li et al. used Eureka[1] and Spring Cloud

[1] https://spring.io/projects/spring-cloud-netflix

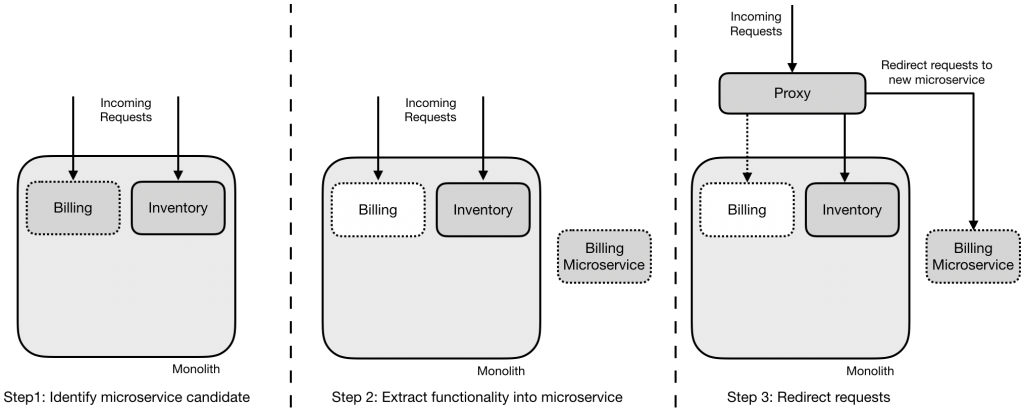

Figure 2.2: Strangler fig pattern [New19]

Gateway[2] in their migration case study for the service registration, service discovery and the API routing [LML20]. In addition to pure HTTP proxying, other asynchronous protocols such as messaging can also be supported. For this purpose, messages can be intercepted and forwarded to the respective microservice based on their content by implementing the content-based router pattern [HW04]. Since functionalities within the monolith are often interdependent and tightly coupled, changes must also be made to the monolith itself when extracting a functionality to enable the new microservice to send requests back to the monolith. For this purpose, the functionality on which the extracted microservice depends has to be exposed from the monolith via an API .

Branch-by-abstraction pattern

Legacy systems are often tightly coupled in practice where the functionality provided by a possible microservice candidate is anchored deep inside the monolith, making it difficult to forward requests from outside the system when using the strangler fig pattern. Therefore, it is necessary to modify the monolith during the migration process. The Branch-By-Abstraction pattern provides a solution to this problem [New19]. It was introduced to handle problems with long-lasting changes on a codebase without disrupting other ongoing development processes on the same system [Fow14]. An abstraction layer is introduced that is used by dependent clients and whose implementations representing the new and old functionality can be exchanged. Figure 2.3 shows the different steps of the pattern. In a first step, an abstraction is created between the functionality to be extracted and the callers of that code. All services calling the original functionality are refactored so that they are finally using the abstraction. In addition, a new implementation of the abstraction is created with the functionality to call out to the microservice containing the extracted functionality. Feature toggles can be used to switch between the coexisting implementations. As the functionality is present in both the microservice and the monolith, any step can be reverted, as in the strangler pattern, reducing potential risks during the migration. However, if the functionality is successfully included in the microservice, it can be removed from the monolith, resulting in a shrinking of the

[2]https://spring.io/projects/spring-cloud-gateway

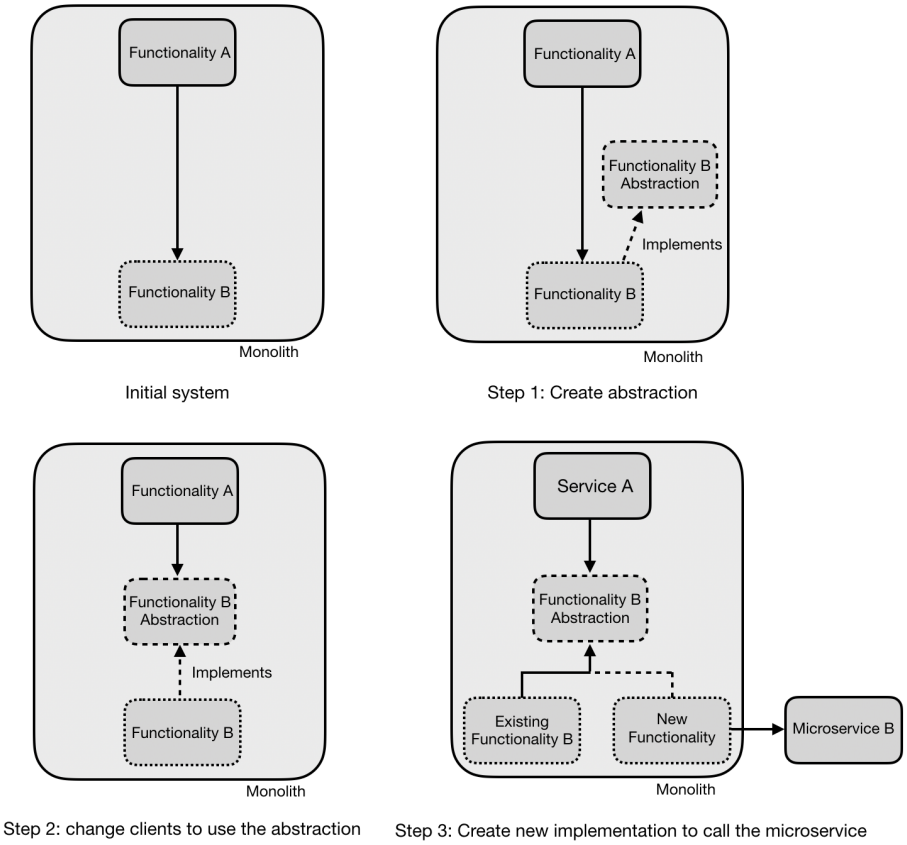

Figure 2.3: Branch-By-Abstraction pattern [New19]

monolith. The pattern can be used in combination with the strangler pattern as part of a migration process when the old system still needs further development without interrupting the migration process by allowing the old and new implementations of the same functionality to coexist.

2.3 Related work

After presenting the migration patterns and the difference between monolithic and microservice architectures, related work on microservice migrations will discussed. The field of microservice migration is a broad research area, which leads to the fact that several researches are partially addressing the research topic of this work and are therefore discussed in the following sections. Section 2.3 discusses the approaches for identifying microservices and decomposing monolithic application which are fundamental steps within migration processes. Section 2.3 contains researches related to the migration strategies.

Microservice identification & decomposition

Microservice decomposition approaches are fundamental processes of a microservice migration and are considered a challenging task. A variety of researches with different orientation and scenarios have addressed the challenge of finding the optimal service boundaries to effectively decompose architectures and extract microservices at granular levels. The approaches vary in terms of orientation, aspects and the level of details as well as the degree of automation. Some of the approaches are based on the key concepts of Domain-Driven Design (DDD), which has proven to be a suitable mean for identifying business capabilities in monoliths. Therefore, Fritzsch et al. [FBZW19] classified several approaches for architectural refactorings and decomposition of monoliths into microservices by means of a literature review. The authors analyzed the approaches and categorized them into the categories *Static Code Analysis aided*, *Meta-Data aided*, *Workload-Data aided* and *Dynamic Microservice Composition*. They have also elaborated a decision guide based on the categories to support engineers in selecting the most appropriate approach depending on their scenario.

Kraus et al. [KZH+20] presented an approach to decompose the Adesso Lottery App *in*|FOCUS by combining static analysis of the source code packages and dynamic analysis of the runtime behavior of the software system. The approach also uses DDD and combines two of the categories defined by Fritzsch et al. The initial domain analysis is conducted to select the bounded contexts used in DDD. The bounded contexts are defined by processing information from various document types and interviews, followed by the creation of use cases that are mapped to defined domain contexts, resulting in a context map. With the help of the the contexts map the target boundaries are defined which results in the definition of the bounded contexts. After the domain analysis the static code analysis and the dynamic analysis of the system behavior is performed. During the static source code analysis source code packages are assigned to the already identified bounded contexts while monitoring and trace visualization tools are used to provide an dynamic overview of the software landscape and the underlying Java architecture including the internal communication between the different parts of the application. This allows to refine the existing bounded contexts or find other potential microservice candidates to decompose the software at a lower granular level. The authors explicitly did not consider any non-functional requirements or quality aspects.

Gysel et al. [GKGZ16] propose a general-purpose approach based on commonly used analysis and design artifacts to overcome the challenges of cutting a system into discrete and autonomous services by enabling loose coupling and high cohesion within services. In contrast to Kraus et al., the authors also focused on architecturally significant requirements and stakeholder concerns by collecting architecturally significant requirements to help software architects in their decision-making. In this context, the authors also concluded that using DDD alone to determine service boundaries for microservices is not sufficient, as more stakeholder concerns need to be considered. Therefore, they introduced a coupling criteria catalog which consists of architecturally significant requirements and decision drivers acting as a semantic model on which their tool framework “Service Cutter“ is built on. Based on the catalog, different user representation also known as “System Specification Artifacts“, such as Entity-Relationship Models or DDD concepts are suggested acting as a user input for the tool to extract the required coupling criteria information. The User-prioritized coupling criteria is then used to transform the input into a weighted graph using scorers to which clustering algorithms are applied leading to candidate service cuts and their dependencies. The authors stress that their method supports greenfield projects as well as iterative approaches for a transition from a monolith to MSA.

Eski et al. [EB18] propose an automatic approach to extract microservices from monoliths using evolutionary and static code coupling information from software classes in combination with graph clustering methods. In their approach the application is represented as graph that includes dependencies and couplings between the classes where clustering technique are applied to suggest microservice candidates. The analysis process consists of two different modules. One module performs evolutionary analysis by extracting and storing software change information using repositories, while the other module performs static analysis by extracting static coupling at the last revision of an existing monolithic application using the parsing of an abstract syntax tree. Both modules serve es an input for the graph clustering module which clusters the graph and identifies microservice candidates. Finally, the suggested microservice candidates are compared with microservices that are created by the developers of related project and acting as authoritative references. Regarding the approach, the authors emphasize that it achieved a success rate up to 89%.

Knoch et al. [KH18] present an approach based on the use of an external service facade to modernize legacy applications, which was also practically applied in a migration process. Since the approach also relies on source code analysis, it can be categorized as *Static Code Analysis aided*. The central components of the approach are service facades, which provide common service operations and are initially created from a domain model. The existing systems will be adapted with implementations of the service operations and the clients are migrated over time to use the existing service facade. These processes serve as a prerequisite for replacing service implementations with microservices. In addition, the authors emphasize that the technique of creating service facades can also be used for internal restructuring of the system. The adapted service implementations are then migrated to microservices while some of the implementations based on proprietary technologies can remain in the monolith.

All these approaches describe the gradual decomposition of the monolith in a technical manner. However, a continuous migration process consists of various steps in which different types of decisions are made and of which the actual technical decomposition is only one part. These decisions depend on various factors that can influence migration in different ways. Therefore, it is also important to investigate these influencing factors.

Strategies for continuous microservice migrations

In addition to the technically oriented approaches, there are also certain methodological approaches to microservice migration that combine various steps as well as technical and non-technical aspects of a microservice migration in an overall concept. For example, Newmann describes in his book "*Monolith to Microservices*" [New19] how to migrate a monolithic application using different patterns and approaches by also focusing on non-technical aspects that need to be considered during the migration. He discusses when microservice architectures can be useful and what should be considered before migrating to a microservice architecture. In addition, he also describes possible pitfalls and challenges that can occur during the migration process itself. Although he presents recommendations on the challenges mentioned, it lacks a coherent decision-making process that also takes into account non-technical aspects of migration, including the concerns and decisions of the stakeholder.

Li et al. [LML20] used the patterns mentioned by Newmann as the basis for their work and proposed a systematic approach based on the strangler fig pattern and DDD. Therefore, many of the steps described are similar, e.g., identification of the service, preparatory tasks in setting up a proxy mechanism, the actual implementation of the service, and removal of the legacy module after successful extraction. The authors also extended the migration steps to include a service prioritisation process, where the order in which microservice candidates are extracted is determined by service value scoring guidelines. Based on the guidelines, the microservice candidate with the highest expected value should be extracted first. Although their approach is similar to that of this work, their focus is different. Since the authors mainly focused on a systematic application of the strangler pattern and the various steps performed when using DDD, factors that can influence the migration as well as decisions based on the factors were not fully considered.

Taibi et al. [TLP17] conducted an empirical study to identify the motivations for migrating towards MSA and the challenges faced during the migration. In addition, they aimed to analyze the migration processes used by the practitioners in order to identify the benefits and issued associated with the specific processes. Their approach for data collection shows similarities to the approach used in this work as they also conducted interviews with experienced practitioners who have recently conducted migrations to microservices-based architectural styles. In regard to the migration process the authors identified three different processes for migrations where the processes have some steps in common but differ in the details. The first two processes focus on the redevelopment of existing functions with the advantage of a complete re-architecture and decomposition of the entire system. In the third process only new functions or existing functionalities that need to be changed are implemented as microservices which leads to the fact that the legacy system is not completely redeveloped but will be gradually eliminated over time. All processes start by analyzing systems using tools or manual approaches followed by defining the architectural structure of the system, which includes tools and frameworks to be used for the architecture as well as the communication protocols and the service APIs. In this context, the first process also contains a high level risk analysis and an architectural plan B. The process differ significantly in regard to the prioritization of feature and service development, where the first process prioritizes based on the customer value and the second process prioritizes according to the dependencies and the amount of bugs. In the third process only new features are developed as microservices where some practitioners used the strangler pattern to add the microservices around the monolith.

Fritzsch et al. [FBWZ19] also conducted a similar empirical study by following a qualitative approach to reveal intentions and strategies for such migrations as well as challenges that were faced with. They also used semi-structured interviews with participants from the industrial sectors. The authors stress that the most common migration strategy adopted by participants was the rewriting of the existing application in combination with extending functionality. Moreover, participants used the strangler pattern to gradually replace the existing system with microservices in multiple cases. The approach of this work differs from both Fritzsch et al. and Taibi et al. in that an in-depth investigation of decisions and factors is performed to develop a systematic approach, rather than exploring motivations and the practically applied migration processes. Silva et al. [SCM19] identified relevant phases with various steps as a guideline for a microservice migration after conducting migration studies with two legacy systems towards MSA. They also recommend to migrate the legacy system gradually using approaches such as the strangler pattern similar to Newmann and Li et al. However, their guide does not refer to the exact application of a pattern, but relies on three general phases. The first phase includes the identification of the key functionalities while the second phase involves the use of DDD to characterize of the monolithic system and reduce

the domain complexity. The third phase is the point of migration, where the decision is made whether to refactor the existing architecture or migrate to microservices. This is based on the results of the second phase, which can then be used to determine the complexity. The authors do not make any concrete recommendations for action, but leave it to the development team to decide on the basis of the artifacts created in the previous phases. Each phase itself is divided into sequential steps that systematically describe how to proceed.

3 Methodology

In this chapter, the design of this study is presented. Therefore, the research questions to be answered in this study are described in Section 3.1. In order to answer the research questions, the execution of the study is outlined in Section 3.2. Moreover, the participants involved in the study as well as the analysis of the data will be detailed in Section 3.4.1 and Section 3.4.2.

3.1 Research questions

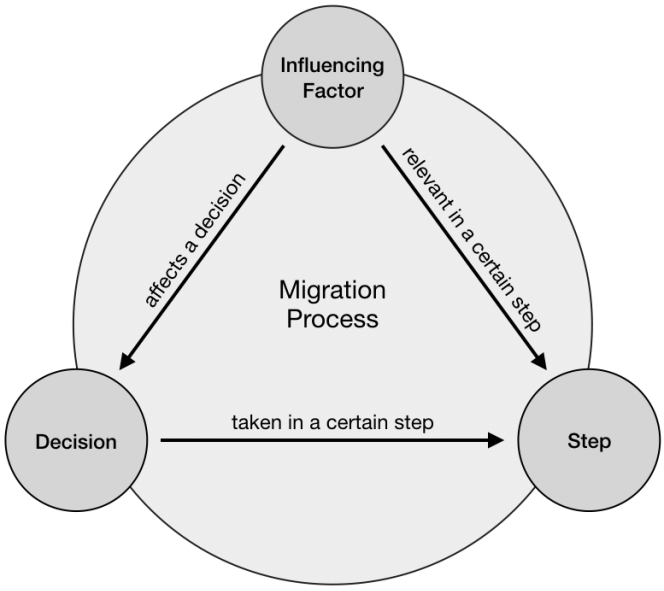

Figure 3.1: Different concepts of a migration process related to the reserachh questions

The primary objective of this work is to develop a systematic process for a continuous evolution from monoliths to microservices. Therefore, the focus is set on the influencing factors, the important decisions and the different steps of a continuous migration process. In order to follow a sequential research design, the identified influencing factors that can occur are first identified. Subsequently, the different steps of a continuous migration process are identified. This allows to bound each factor to the step in which they become relevant during the migration process. This also allows to determine if and how a factor becomes relevant in another step. Finally, the possible decisions based on each influencing factor are considered. Figure 3.1 illustrates the different concepts of a migration process that are identified within the research questions. The sequential structure of the study based on the research questions is shown in Figure Figure 3.2. In this study, the following research questions are addressed to achieve the main objective:

Research Question 1: What are the influencing factors in a microservice migration?

In a first step, the factors that can influence a migration are identified. The factors serve as causes on the basis of which certain decisions are later made in the migration process.

Research Question 2: What are the different steps of a microservice migration?

This question focuses on identifying the different steps of migration process. Furthermore, the factors are assigned to the respective steps. This serves to chronologically assign the occurrence of the factors in the migration process.

Research Question 3: Which decisions are made based on the factors during a microservice migration?

This serves to identify the decisions which are made during the microservice migration. These decisions are based on the influencing factors that occur during the migration.

Research Question 4: What is a feasible process for systematic migration based on the strangler pattern?

This question serves to systematically link the different decision points resulting in a decision guidance for a continuous evolution towards MSA.

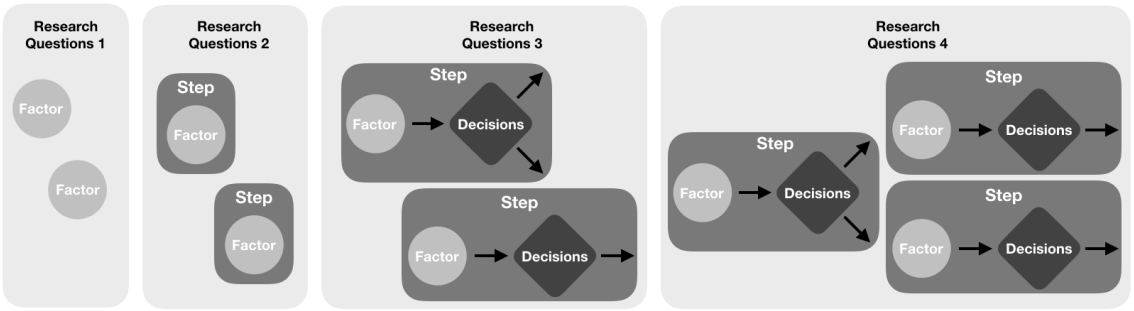

Figure 3.2: Sequential design of the study

3.2 Study execution

Empirical studies in the field of software engineering differ in their characteristic for studying a particular phenomenon. Runeson and Höst [RH08] differentiate between four types of purposes based on Robson's [Rob02] classification: *Exploratory*, *Explanatory*, *Descriptive*, *Improving*. According to the classification, empirical studies also follow different research strategies such as surveys, experiments, case studies and action research which is closely related to case study research [WRH+12]. The literature also differentiates between qualitative and quantitative approaches for collecting data. In this context, several studies in recent years investigated the process of migrating towards MSA, using a qualitative research design to answer research questions related to various aspects of this process. As part of these studies, case studies and interviews have been widely used and have proven to be an appropriate research method to answer *"how"* and *"why"* questions as part of these software engineering researches. In particular, case studies were used to evaluate and improve the migration approaches developed. Therefore, this study also uses qualitative research useful for answering the how question. Interviews and an industrial case study will be conducted to collect data for the approach to be developed and to test the approach in a real-world context to potentially refine it based on the results collected. The study design relies on four main phases. The first phase consists of a literature review to gather existing literature related to the research topic. The approach for the literature review will be explained in Section 3.3. In the second phase interviews are conducted targeting practitioners involved in the process of migrating monolithic applications. The interview strategy, the structure and the art of wording interview questions are described in Section 3.4. These two phases serve to collect data that will be used as input for the migration approach. The migration approach will be elaborated within the third phase. Figure 3.3 illustrates the different phases of the study.

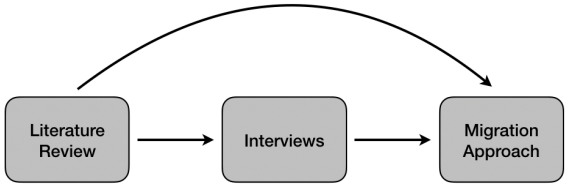

Figure 3.3: Study execution

3.3 Literature review

A literature review is *"the process of searching for, reading, summarizing, and synthesizing existing work on a topic or the resulting written summary of the search"* [R11]. The result is a comprehensive overview of all existing information about a certain phenomena in a thorough and unbiased manner[Lea17]. The purpose of this literature review was to find existing literature that has already addressed the decision-making as well as influencing factors in the context of continuous microservice migrations. Figure 3.4 illustrates the search strategy, which was followed to get most appropriate results in the literature review. Although there are similarities to a systematic literature review, conducting a SLR was not the scope of this work, as the purpose of this literature review was to only search for appropriate publications related to the concepts of this study. Therefore, the search string was determined based on the titles and abstracts of the existing publications mentioned in Section 2.3 and the research questions presented in Section 3.1. This resulted in the following groups of keywords, for which lexical and syntactic alternatives were also considered: (i) *"microservices, monolith"* - These keywords were used to search for publication which are related to migrations from monoliths to microservices, (ii) *"migrate, transform, modernize, refactor"* - This group of keywords consists of nouns or verbs that indicate a transition , (iii) *"factor, challenge, decision, roadmap"* - This group of keywords consists of nouns or verbs which are part of the research questions and therefore important to answer them. In addition to *"factor"*, *"challenge"* was also selected as an alternative keyword. Since various publications address challenges in a broad range, the filtering process ensured the selection of publications that address challenges that have a significant positive or negative impact during the migration process. To remain consistency, the search has been applied to the titles and abstracts of publications in all the data sources. Listing 3.1 shows the search string that has been used for querying the data sources. The following scientific databases and indexing systems were selected as electronic data sources for this literature review: ACM Digital Library, IEEE Xplore and Google Scholar. To filter the results a defined selection criteria has been used. Only publications written in english that also contribute to answering the research questions were considered.

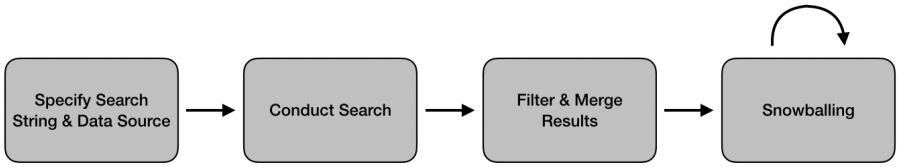

Figure 3.4: Search strategy

```
("microservice*" OR "micro-service*" OR "monolith*")
AND ("migrat*" OR "transform*" OR "moderniz*" OR "refactor")
AND ("factor*" OR "challenge*" OR "decision*" OR "roadmap")
```

Listing 3.1: Search string

3.4 Interviews

With the growing number of empirical studies in the field of microservice migrations, interviews have been widely used to collect data about activities, approaches and experiences during the entire process [DLM18] [TLP17]. They have also been part of empirical investigations on why certain methods have been practically applied in the industry and what impact they had [FBWZ19]. The interview types of the different studies differ fundamentally in their structure, which leads to the fact that the most appropriate should be selected depending on the objective and focus of the study. In this context, a distinction is made between structured, semi-structured and unstructured interviews where a less structured interview allows the respondent more flexibility in responding [Rob02]. Interviews also differ in the degree of control that the interviewer maintains over the interview [Rub11].

In order to ensure reliable and meaningful results, this research follows the principles and recommendations from Patton [Pat02] and Rubin et al. [Rub11]. In addition, the experience of Hove et al. [HA05] in conducting semi-structured interviews in software engineering research is considered, which is based on several literatures about qualitative interviews, including the two literatures already mentioned. For qualitative interviewing, Rubin et al. distinguish between several categories, where the core forms of in-depth qualitative interviews are semi-structured and unstructured interviews. In the field of software engineering research, Hove et al. [HA05] concluded that semi-structured interviews are frequently used in the field of software engineering where the phenomena is often qualitative by nature. Both Patton [Pat02] and Rubin et al. [Rub11] mention the use of conversational guides, such as interview guides or research protocols where questions are written in a formal or informal manner to ensure that the same basic lines of inquiry are pursued with each person interviewed. These guides enable the researcher to balance the need for predictability with the freedom to explore unanticipated topics. The guide may include a limited number of general questions in advance and more specific questions to be asked to explore how the participant qualitatively experienced a particular phenomenon. In this context, Rubin et al. [Rub11] distinguish between *main questions*, *follow up questions* and *probes* where main questions are used to discuss separate parts of the research questions while follow-up questions are used to obtain more detailed information on specific aspects. Probes are used to manage the conversation and to ask for examples or clarification. This strategy allows the interviewer to determine the level of depth to explore a particular aspect or to ask questions about new aspects that emerge during the interview that were not originally anticipated. This for this reason, a semi-structured approach was chosen as the most appropriate for this work. An interview guide was prepared in advance, consisting of worded main questions covering the various aspects of the research questions by gaining insights on the migration steps, factors and the decisions made during the migration process. The interview guide is given in Section A.1 of the appendix. Similar to Rubin et al. [Rub11], Hove et al. [HA05] suggest for interviewing to either start with general questions and then ask more specific questions or to reverse the order and start with a very specific questions, followed by more general questions. The order of the questions can be varied during the interview depending on the situation. In addition, some follow-up questions are formulated in advance to control the level of depth of the answers.

Regarding the wording of questions Patton [Pat02] suggests to ask *"what"* and *"how"* questions to elicit descriptions from the subjects, where *"why"* questions should be avoid since they give more or less speculative explanations. Furthermore, questions that can be answered with *"yes"* and *"no"* should be avoided. Hove et al. [HA05] also expand on this and share their experiences

with the different types of questions used in various interviews during software engineering studies. As with some of Patton's question types, they suggest reflective questions and questions that require participants to describe how they did something to elicit rich and useful information. Rubin et al. [Rub11] refer to these types of questions as tour questions, which ensure that participants provide a comprehensive description of the steps within a process or how they addressed specific issues, e.g. *"Can you explain step by step how you performed the migration?"*. He also emphasizes asking tour questions about participants' experiences as an introduction to a topic, such as *"Can you tell me about your experience with past microservice migrations?"*.

Recording

In order to obtain meaningful results from the interview process, it is useful to keep a record of the interviews so that the analysis can be accurate based on what is said. Hove et al. [HA05] suggest video taping, audio taping and note taking during the interview as common techniques. For this study audio taping via a remote communication software was chosen as the most appropriate technique. This allowed to capture all the details of the interview without focusing on taking notes.

3.4.1 Participants

Participants in the study had to have at least some migration experience. For this purpose, employees who had already participated in migration projects in the company were selected via the supervisor's references at Adesso. In this context, all participants were involved in the further development and modernization of the system described in Section 4.2.1. In order to obtain information from different perspectives, care was taken to select participants with different roles in migration projects. The participants were contacted directly by the interviewer via e-mail and informed as far as necessary about the purpose of the study without giving any information on the interview guide. The interviewees were also given a questionnaire for collecting demographic data including information about experiences with microservices and microservice migrations. To ensure confidentiality, the participants were asked for their consent to create audio recordings. The declaration of consent as well as the questionnaire are included in the appendix under Section A.2 and Section A.3.

3.4.2 Data analysis

In addition to evaluating the results of the literature review, fundamental approaches of qualitative data analysis were followed in processing the raw data from the interviews in order to provide clear and accurate answers to the research questions. Therefore, the interviews were transcribed first. This task involves the complete and accurate written word-for-word reproduction of the interviews serving as a basis for further qualitative inquiry. To perform this process the software f4transcript[1] was used. After transcribing, the participants were given the transcripts to verify the statements and the degree of content anonymity. The transcripts were then used to conduct qualitative data analysis. In this context, a coding process was carried out in which codes in the form of single words or

[1] https://www.audiotranskription.de/en/f4transkript/

short sentences were assigned to parts of each transcript. These codes assign a summarizing and essence-capturing attribute to a piece of linguistic or visual data to enable a deeper reflection on the meaning of the data [Mil14]. Charmaz [Cha01] also describes the process of coding as the "critical link" between data gathering and their explanation of meaning. Figure 3.5 To perform the process of coding, the software toolkit MAXQDA[2] was used. Moreover, the recommendations and experiences of Saldaña [Sal09] were followed. The coding was performed in two stages: *First Cycle* and *Second Cycle* coding.

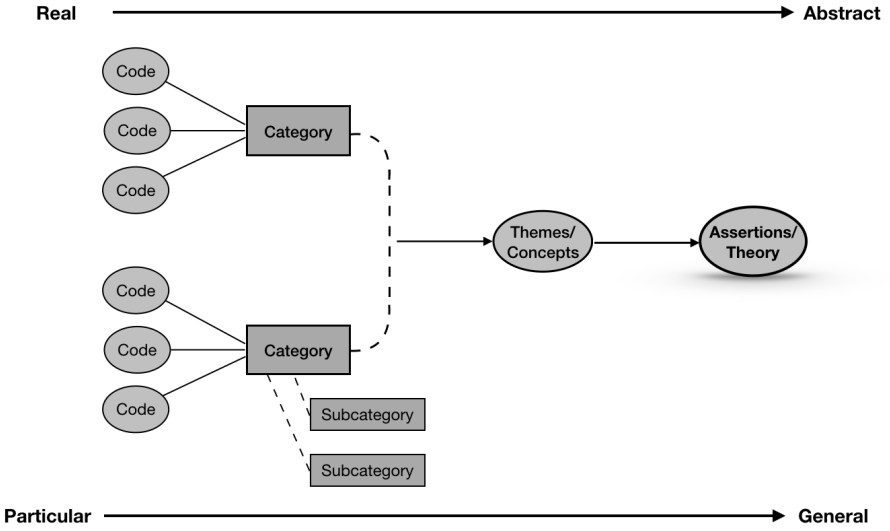

Figure 3.5: Codes-to-theory model for qualitative inquiries [Sal09]

First cycle coding

In first-cycle coding, codes are initially assigned to each data unit [Mil14]. Several coding methods are available for this coding step, which can also be combined as required. For this study, a combination of descriptive coding and process coding was chosen, which, according to Saldaña[Sal09], are suitable methods for almost all qualitative studies. Process coding uses gerunds ("-ing" words) to denote actions in the data, while descriptive coding uses nouns or short phrases to summarize the basic topic of an excerpt [Mil14]. Prior to the first cycle coding, initial categories were derived from the research questions. The categories were defined as *Factors*, *Steps* and *Decisions*. Sections of the data were then initially assigned to these categories. In the course of coding, further parts were added or reassigned to the various categories. In addition, further subcategories were created. An excerpt from the list of codes after the first cycle coding with MAXQDA is given in Section A.4.1.

[2]https://www.maxqda.com/

Second cycle coding

Second cycle coding uses the material from first cycle coding and reorganizes the codes into a list of broader categories, themes, concepts, or assertions according to their similarities [Sal09]. For further analysis in this work, pattern coding was selected as a second cycle method. Pattern coding is a method of grouping the excerpts and codes into a smaller sets of categories, topics or constructs [Mil14]. The categories of codes resulting from pattern coding are shown in Section A.4.2.

4 Results & Discussion

The chapter presents and discusses the results of the study. This also includes the results of the different research methods. Section 4.1 presents the results of the literature review. Section 4.2 describes the participants in the interviews, the evaluation of the demographic data and the legacy system which was part of the migration. In Section 4.3 the different research questions are answered. This also includes the elaborated migration approach.

4.1 Literature review results

	Publications
1	Wolfart et al.: *Modernizing Legacy Systems with Microservices: A Roadmap*. In: Evaluation and Assessment in Software Engineering. pp. 149–159 (2021) [WAS+21]
2	S. Newman: *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media (2019) [New19]
3	Ayas et al.: *Facing the Giant: a Grounded Theory Study of Decision-Making in Microservices Migrations* (2021) [ALH21]
4	Fritzsch et al.: *Microservices Migration in Industry: Intentions, Strategies, and Challenges*. In: IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 481–490 (2019) [FBWZ19]
5	Kalske et al.: *Challenges When Moving from Monolith to Microservice Architecture*. In: Current Trends in Web Engineering. pp. 32–47 (2018) [KMM18]
6	Velepucha et al.: *Monoliths to microservices - Migration Problems and Challenges: A SMS*. In: Second International Conference on Information Systems and Software Technologies (ICI2ST). pp. 135–142 (2021) [VF21]
7	Li et al.: *Microservice Migration Using Strangler Fig Pattern: A Case Study on the Green Button System*. In: International Computer Symposium (ICS). pp. 519–524 (2020) [LML20]

Table 4.1: Literature review results

As a result of the literature review, seven publications were identified that address various aspects of microservice migrations including decisions and factors. The resulting publications are shown in Table 4.1. The literature review also identified three publications which were also mentioned in Section 2.3 and were considered important for answering the research questions. The aspects of microservice migration mentioned in these publications were used to guide the design of the interview guide. In addition, the data from the interview analysis was supplemented with information from the publications to answer the research questions. Publications 2, 4, 5, and 6 were used as a supplement to determine the factors. Publications 1 and 6 were consulted to provide additional information on some of the identified migration steps, as both publications present different steps for their migration approaches. Publication 1 served as a reference guide for the various decision-making processes in the developed migration approach.

4.2 Interview results

During the interviews the aim was to correctly balance main questions, follow-ups and probs in a loosely manner. Care was also taken to ask the main questions in a logically coherent manner to maintain the flow of the interview and prevent it from becoming choppy. The interview guide was adjusted based on participant responses and follow-up questions were asked accordingly if the participant made statements about a particular aspect of the research objective. This led to the use of the *"Main branches of a tree"* and *"Opening the floodgates"* patterns mentioned by Rubin et al. [Rub11] during the interviews. The *"Opening the floodgates"* pattern implied in some part a narrative quality of the interview when the participants recounted their experiences. During the interviews, which lasted between 20 and 50 minutes, all interviewees were communicative and responsive. The possible interaction issues described by Hove et al. [HA05] were not apparent. The participants were mostly software engineers, including developers and architects. In addition, a project manager and requirements engineers was also included. All participants had significant work experience of at least $4\frac{1}{2}$ years in their professional fields and also pursued them during the migration projects. Regarding the experiences derived from the questionnaire results, a low number means little experience and a high number means a lot of experience. In addition, a distinction is made between the question about the experience they have through their work and the competence they believe they have based on their practical and theoretical knowledge. These two types of questions were asked to the participants about microservices and microservice migrations. The results show that the participants have different levels of experience with microservices development and operation. P5 has the most experience in this area. Overall, however, the participants had moderate experience with microservices development and operation and considered themselves competent in this area. The same is true of their experience with microservice migrations, where they consider themselves slightly less competent. The entire demographic data of the participants is presented in Table 4.2. The participants were involved in different migration projects. They also worked together over different periods of time to further develop and migrate the system described in Section Section 4.2.1. The migration of the system has led to the emergence of various services around the Monolith, which have also been further developed in the course of the process. An overview of the system and services addressed by the participants is listed in Table 4.3.

Participant ID	Years of Experience	Participant Role	Participant Role in Migration Projects	Experience in microservices (1-5)	Self-assessment of competence in microservices (1-5)	Experience in micorservice migrations (1-5)	Self-assessment of competence in microservice migrations (1-5)
P1	10	Team Lead, Software Architect	Software Engineer, Software Architect	5	3	2	2
P2	6	Project Manager	Project Manager, Requirement Engineer	1	1	4	2
P3	4,5	Senior Software Engineer	Developer	2	4	2	3
P4	25	Software Architect	Software Architect, Developer	2	4	3	3
P5	20	Senior Software Engineer	Developer, Co-Software Architect	3	3	3	3
P6	15	Senior Software Engineer, Software Architect	Software Engineer	2	2	1	2
P7	10	Software Engineer, Managing Director	Fullstack Developer	3	4	2	3

Table 4.2: Participant demographics

System / Service ID	Purpose	Type
S1	Vehicle lifecycle management system	Legacy system (Section 4.2.1)
MS1	Customer service measures	Microservice extracted from S1
MS2	Processing of master data for cases	Microservice developed beside S1

Table 4.3: System and service overview

4.2.1 Legacy System (S1)

The microservice migration was performed on a legacy application of an automotive OEM, which was further developed and modernized on behalf of Adesso. The system collects data over life cycles of vehicles that have already left the production plant. This involves the management of customer measures, such as a service measure to improve quality or a recall. Moreover, it indicates for a vehicle which customer service measures are currently still active or available. This allows to support employees in that a comprehensible communication of the problem to the market organizations can be carried out. The application combines multiple domains and is connected to other third-party systems via multiple interfaces, resulting in a distributed system landscape with different dedicated servers. The system also includes batch functionalities and file processing steps, which have already been migrated by Adesso as part of different projects. The monolithic core is based on an obsolete Java EE stack and a single database instance. The frontend is written in AngularJS.

4.3 RQ 1: Influencing factors

In the course of the analysis, various influencing factors for a continuous migration towards microservices were identified. Some factors arise from the different complexities of the differing architectures and the shift in these complexities due to migration. Additional factors arise from other changes that accompany the migration at various levels. Moreover, factors differ in the way they affect migration. Factors may have a direct impact on migration, while others influence migration indirectly by affecting other factors. This mutual influence also leads to a variation in the strength of the impact on the migration process. The influencing factors identified in this study are grouped into three categories: *Business Factors*, *Technical Factors* and *Organizational Factors*. Table 4.4 lists the identified factors based on their categories and the participants who mentioned them. Figure 4.1 illustrates the strength of the influence of the different factor categories on the migration process.

Category	Factor	Participants
Business Factors	Importance of migration	P1, P3, P4, P5, P7
	Scope of new requirements	P1, P7
	Selection of microservice candidates	P1, P3, P5
	Cost & effort calculation	P2, P3, P4, P5
	Engagement of stakeholders	P1, P3, P4, P5, P7
	Customer-related restrictions	P1, P2, P4, P5, P6, P7
Technical Factors	Infrastructure limitations	P1, P5, P6
	Complexity of legacy system	P1, P4 P7
	Difference of data consistency	P4
Organizational Factors	Limitation of the organizational structure	P1, P2, P3, P4

Table 4.4: Identified factors

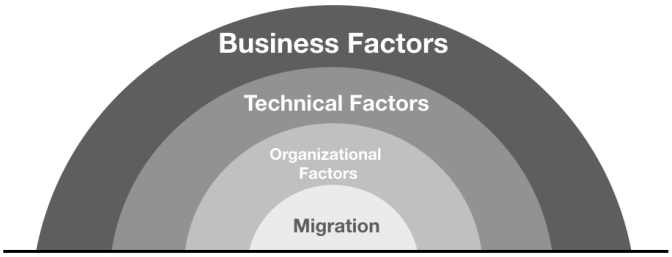

Figure 4.1: Influencing factors in relation to their impact

4.3.1 Business factors

Business factors were identified in the evaluation as the biggest influencing factor of a migration project, as they are crucial to whether and how the operational, technical and organizational changes take place. They also indicate how migration is progressing and are responsible for the possible failure of migration.

Importance of migration

> *...in the end the product wins instead of the migration. So, in case of doubt, the old world wins when something new has to be done urgently and the other is then put on hold and postponed and delayed.*
>
> P4, *Software Architect*

Legacy systems are often still being developed in parallel and new requirements have to be implemented [P4]. In the case of S1, the migration was carried out in parallel with project operation [P5,P4,P7]. Therefore, it is important which importance and consequently which capacity is assigned to the migration activities. This also depends on how the migration is integrated into the project and how the migration activities are prioritized compared to the regular project activities in each sprint. In this context, software architect P4 stated that a parallel migration was necessary because it was the only way they could still deliver the high priority features to the customer. However, this leads to the fact that primarily capacity is used for new features and the remaining capacity is then used for the migration. As a result, the migration can be delayed because there are more frequent tasks with higher priority and it is therefore not possible to fully focus on the migration [P3]. This also means that the extraction of more complex functionalities can only take place to a limited extent. Thus, the prioritization of the service candidates in relation to the effort available for the migration also has a decisive influence. The resulting delay can therefore have an impact on the budget and increase the risk of failure [P1]. However, it should also be noted that migration and further development should be integrated in one process, because separate parallel development has extreme budget implications, according to P4. As a result, all migration processes at S1 were incorporated into the normal development process.

Scope of new requirements

> *There were these new feature packages that had to be implemented in any case, and we looked at the overall scope, i.e. the extent to which functions in the Monolith were affected...we then estimated all of these internally and then looked at which features could be implemented in the respective sprints in terms of the amount of work and then we incorporated the migration things depending on that.*
>
> P7, *Fullstack Developer*

During a migration process, relevant requirements for extending the monolith can arise from the stakeholders. The extent to which the requirements must be implemented also depends primarily on the previously mentioned factor (importance of the migration), because this significantly determines the resulting weighting of migration activities and requirements to be implemented in the project. If, as in the case of S1, new requirements have higher priority, the scope of these requirements become an influencing factor for the progress of the migration [P7]. If this feature depends on a functionality within the monolith, it becomes a potential microservice candidate and migration activity can take place. However, if the scope of the feature is related to an extensive feature set within the monolith, and thus migration is costly, no migration activity can consequently take place [P1]. The same applies if the requirement in the case of MS2 has no intersection with functionalities in the monolith.

Selection of microservice candidates

> *...Personally, now, with more experience, I would have said that maybe we should not have chosen the pain points for migration right away, because experience shows that the pain points are of course the most difficult to solve which is also a risk for the migration plans.*
>
> P1, *Software Engineer and Software Architect*

The way in which service candidates are identified and prioritized for migration is an important factor for migration. In this context, Li et al. [LML20] point out that an extraction order,is crucial for the migration to bring maximum benefits. P1 mentioned that during the migration of S1, functionalities with increased performance problems were initially identified, which were very costly to migrate. Nevertheless, they simply went after those big ”pain points”. In retrospect, he would no longer set the priorities based on only such indicators. In addition to incorrect prioritization, no prioritization at all can also have a negative impact on the migration process, as P3’s experience shows. In his migration project, no prioritization was done, which led to a lot of restructuring during the process. Prioritization should therefore take into account the interests of all stakeholders, even if much of it is related to the view of the customer, who always wants to take advantage of the most visible [P5].

Customer-related restrictions

> *... it was partly hosted by the customer, and they had very strict requirements, also in terms of the technologies to be used, so of course during development we ran into different regulations in terms of compliance, data security, cloud risk and so on. We were also later classified quite differently. That is, we also had to comply with other regulations.*

P1, *Software Engineer and Software Architect*

The development of larger software projects of large companies is often dependent on the compliance with various corporate requirements, which consequently also applies to the execution of migration projects. The use of modern technologies and the transition to the cloud are strongly influenced by this. Particularly in the cloud environment, there are increased restrictions for companies when it comes to data security. In terms of cloud-related regulations, several participants mentioned that various regulations such as data security and compliance affect the migration process of S1, as "cloud risk processes"had to be triggered first [P1, P2, P3]. During the migration were also limitations and requirements in terms of technologies. In addition, other participants mentioned that there were limitations and requirements related to technologies. API authentication and versioning requirements had to be followed, and development had to be done with deprecated technologies [P4, P7]. In this context, P6 also addressed the offered PaaS solution for the development of MS2, which only provided outdated database versions that did not support the required features they wanted to work with.

Cost & effort calculation

> *Most of the time, it was more budget decisions where people said, "Well, it's working at the moment". So, in principle, we would have to invest a lot at once, because it costs significantly more than we had planned. It was not always possible to implement everything ideally.*

P5, *Developer and Co-Software Architect*

Software projects, especially in the IT services sector, are often measured by their costs. This also applies to migration projects, which are also affected by risk factors determined from a project management point of view. Project manager P2 noted that the effort and cost of migration projects are difficult to calculate, which can quickly make them expensive. As a result, the planned budget also affects the migration, which can lead to the extraction of time-consuming system parts not being carried out [P2]. Also Fowler [Fow15] points out the costs and associated risks that must be considered when adapting the complex microservice architecture, which also often lead to serious problems in the project. Also P4 noted that the cost issue for the further development and migration of the system must also be taken into account, as it decides where the "man power" goes to. If this aspect is not considered, the project might fail.

Engagement of stakeholders

> *...it wasn't always clear to the customer why we were doing this, why we should be doing this. Yes, and he didn't always see the point of paying money for it. So the migration had to be scaled back sometimes.*

P1, *Software Engineer and Software Architect*

Stakeholder commitment to the migration project is also an important influencing factor as it is important to create the capacity and the budget for a continuous microservice migration. P4 stated that the project would have needed a clearer commitment from management and the entire stakeholders to create the budget and the capacity for the migration process. A lack of this commitment leads to the fact that modernization operations are left behind compared to any urgent feature relevant requirements. P1 also reported bad experiences with the commitment of stakeholders who could not recognize the migration benefit. P7 also mentioned the impact of engagement on service candidate selection. Microservice candidates that would have been important from a technical perspective beyond extracting "paint points" had to be presented to the customer as a major benefit first. P3 also explained that the reason for this problem is that a IT service provider often has to satisfy customers according to their requirements.

4.3.2 Technical factors

> *All these aspects, especially the issue of complexity, meaning domain complexity, can be better handled with microservices...but I also have to say that with microservices you buy or accept increased technical complexity, so the complexity is not reduced, it just shifts.*

P1, *Software Engineer and Software Architect*

Monolithic applications tend to have a high degree of domain complexity, which should be also reduced through the use of microservices. To achieve this, a microservice migration requires various technical means. As microservices reduce complexity through reduced coupling, this also requires more technical effort. Furthermore, legacy monoliths often consist of outdated technologies with little technology diversity, while microservice applications have a modern polyglot technology stack. This technology diversity refers not only to the technologies used to implement microservices, but also to the deployment and operation of the entire infrastructure. Microservices thus also bring with them a certain basic technical complexity, with which one is also already confronted during migration. In this context, the domain complexity also decreases, but the technical complexity increases. As a result, there are various technical factors that influence the migration which will be explained in the following.

Infrastructure limitations

> *Also the machine, the virtual machine that we had there, which was Windows 7 Enterprise, we also got from them. The application didn't allow it to be any other way. That was already as old as the whole infrastructure, which was also there. That made the whole thing really complicated*

P7, *Fullstack Developer*

A technical influencing factor that arises in a microservice migration is the limitation of the existing infrastructure. Monolythic systems are scaled as a whole, while microservices can be scaled individually due to their nature. As a result, microservice environments are highly scalable and often distributed across multiple processes on different servers [Luk18]. This distribution requires increased routing and load balacing mechanisms of the underlying infrastructure compared to monolithic environments. In this context, both P2 and P6 pointed out that it was necessary to adapt the current infrastructures as API gateways and load balancing became increasingly necessary for the migration. The limiting factor of the infrastructure designed for monolithic systems is also noticeable in the area of provisioning and monitoring of microservices. Monolythic systems have a small number of deployable components with less configuration management resulting in a straightforward process [New15]. Microservice environments are much more complex, so existing CI/CD pipelines are no longer sufficient. Thus, they are often not designed for highly automated deployments in cloud environments such as CloudFoundry[1] and Kubernetes[2]. In addition, the polyglot technology stack also brings with it other necessary components in the infrastructure. Here, P1 relies on the classic SpringBoot stack used in the migration of S1 using Docker, which also entails the provision of Docker registries. This is based on the fact that monolithic applications run in virtual machines, while microservices development often relies on container virtualization, which enables efficient encapsulation of the application and its dependencies by leveraging Linux kernel capabilities.

> *The pace of scale and dynamism of the infrastructure and application environment are accelerating, with the advent of containers, microservices, autoscaling and softwaredefined 'everything' stressing the capabilities of existing Application Performance Management tools*

Cameron Haight [Hai16], *VP Gartner Inc.*

Monitoring monolithic applications also reaches its limits with microservices as it focuses on a few components with their local call stacks. This approach is insufficient due to the highly distributed nature of microservices and the polyglot technology stack they are based on [Sei17]. Cameron Haight has also pointed out these challenges of monitoring microservice architectures, which are

[1] https://www.cloudfoundry.org/
[2] https://kubernetes.io/de/

already faced duting the migration. Therefore, data from more technologies must be considered and technologies such as distributed tracing must be used to also validate extracted services during the migration.

Complexity of legacy system

> *One difficulty we encountered was the large dependencies of the individual functionalities with each other, so that it was quite difficult to really pull out a microservice and then to be able to use it separately later. Because you always noticed that this part was still missing and then this and that was added. Well, that's probably just a property of a monolith, of course.*
>
> P7, *Fullstack Developer*

Finding the right service cut has proven to be a major technical challenge in migration projects, which has a significant impact on the course of microservice migrations. Several publications consider the process of identifying and extracting candidate services to be a time-consuming task and emphasize the importance of ensuring that the decomposition of services is correct [KMM18] [VF21]. Several participants confirmed this by mentioning that the use of DDD was often not smooth due to the domain complexity of the legacy system and large dependencies on the individual functions [P1, P7]. P4 also added that due to analyzed circular dependencies it was not possible to physically split the monolith in the beginning. When reflecting on the process, P1 mentioned that they should have started in smaller steps and tried to extract smaller pieces rather than going straight to the "big pain points."

Difference of data consistency

> *...data consistency has also been such a big factor. If you have such an old monolith, you often have ACID consistency, so atomic transactions. But microservices have their own databases, so you have the same problem of data consistency as in distributed systems.*
>
> P4, *Software Architect*

Monolithic application often rely on single database while microservices encapsulate their own database [New15]. When using a single database, safety and consistency of the data is ensured. However, when migrating microservices, the data is split between different databases according to the microservice paradigm. This limits the ability to use ACID database transactions [New19]. In regard to the transaction, Software architect P4 mentioned that consistency is a big factor, as you have to consider during the migration where you can rely on ACID consistency and where eventually consistency is used.

Technological & architectural knowledge

> *The difficulty that definitely arose was, of course, that you had to know the technical structure of the old system, including the various functionalities. There was also the function that was then outsourced to MS1, and you had to know the entire new architectural environment. That is, of course a balancing act that, you can't do that easily. That also means you simply have to be familiar with new technologies and at the same time you have to be familiar with the old technologies.*

P2, *Project Manager and Requirements Engineer*

Since the migration involves different types of systems that differ on both the technological and architectural levels, a wide range of experience is required to successfully complete the migration from a technical perspective. Fritsch et al. [FBWZ19] state that participants in their study indicated that mastering technologies also required a significant amount of time. Velepucha et al. [VF21] also concluded that selecting and properly using technologies and practices that have already been successfully tested in microservices are part of successfully migrating a monolith. Both P1 and P2 stated that in their case, the diversity of knowledge about the new technologies and the old technologies affected the migration process, as employees who were familiar with the technologies also had to be hired, which required a lot of time. Both also mentioned that a process of knowledge transfer through contacting external experts and other teams has had a positive impact.

Deviation of release & deployment strategies

> *Deployment dependencies had to be taken into account in our case. There is always the danger that you cut a beautiful monolith in such a way that you have beautiful runtime units, but if you then realize that if I change one thing, I have to redeploy all services, then you have gained nothing again.*

P4, *Software Architect*

The release and deployment methods of monolithic applications are considered straightforward and fundamentally different from those used in microservices. With the growth of microservices during migration, this issue becomes more complex as the number of dependencies between components also increases. Therefore, the integration between development and operational practices becomes more and more important. In this context, DevOps practices are often followed in microservice architectures to release faster, more often and in shorter cycles [WAS+21]. Companies like Netflix have also adapted their service delivery processes to meet product requirements, scale team development and better manage their cloud infrastructure when developing microservices []. Also Newman notes that continuous integration and continuous delivery techniques need to be mapped to microservices [New15]. In the case of S1, a huge repository was used to store all the code. According to Newman [New15], this is often used for lock-step releases where it doesn't matter to

deploy multiple services at once. This intent is also reflected in monolithic development, which often results in a single build, while microservices development should ensure that one pipeline per service is used to release services independently [New15]. Depending on the form in which the legacy system still exists, the release and deployment cycles must also integrate the standard release cycles of the monolithic applications. Several participants encountered issues with these release cycles during the migration [P1, P5, P7]. P1 mentioned that the low, fixed release cycles of the monolith affected migration because they had to be adhered to, which often led to time pressure as no integration was "not an option"to the customer. According to P7, this time pressure also led to an influence on the quality of the application. To overcome these issues, feature toggles were used during the development of MS2 as they allowed to activate certain functionalities according to the respective release [P5].

4.3.3 Organizational factors

Limitation of the organizational structure

> *It was actually the case that we set up new teams for the various areas during the course of the project. But we didn't do that because we said, we want to manifest new knowledge only in the different teams. It's not a good idea anyway because silo knowledge is very, very dangerous. The idea came more because we have simply become too large due to the increased capacity to work on all the services at the same time.*

P2, *Project Manager and Requirements Engineer*

In the context of organizational structures and system design, reference is often made to Conway's Law, which states that the organisation that designs the system will produce a system whose structure reflects the structure of the organisation [Con67]. Monolithic systems are usually based on different levels, on the basis of which responsibilities are often separated. This separation of responsibilities with clear roles and horizontal boundaries in large teams is also facilitated by the fact that the organizational structure of most companies is often hierarchical. As a result the responsibilities of teams are also separated by technologies, with one team responsible for the frontend and another team responsible for the database reinforcing the formation of silo knowledge [VF21]. However, microservices development is driven by small, separate and agile teams with different roles based on horizontal boundaries. Since these structures differ fundamentally, a migration must also be accompanied by a transition at the organizational level. This also includes the adaption of the communication structure to this new style of architecture [KMM18]. Consequently, existing organizational structures for the development and operation of monolithic applications are reaching their limits. Figure 4.2 illustrates the outgoing organisational structure of a monolithic system and the structure to be achieved in microservices development. Several participants mentioned that organizational restructuring also took place during their migrations [P1, P2, P3]. The restructuring in this case was primarily driven by the increase in capacity needed for the migration and the fact the the team was already to large. For this purpose, a team for the migration was created at the beginning, but with the intention to avoid silo knowledge as far as possible [P2]. Later on, the teams for the development and maintenance of the microservice were also separated. When

moving to microservices, this separation of teams is also necessary to change code ownership, as it can be difficult when everyone is working on the same code base, as with a monolithic applications. The impact of this factor grows as the architecture scales through the emergence of a large number of different services [KMM18]. Newman[New19] also highlights the change in code ownership with respect to code changes as an important factor in successfully establishing a microservice architecture. Therefore, it is important to move from collective code ownership to strong code ownership between the emerging teams as the project grows larger. This also helps to remain customer-focused. Software architect P4 mentions that this approach has resulted in a more structured migration and minimized the risks of breaking changes to the code.However, this separation also depended on a good modularization of the monolith. From the perspective of project managers P2, the entire restructuring was also associated with the acquisition of new developers, which affected the migration at the beginning, as people first had to get to know each other but also the processes. This change in team structure also leads to a redistribution of roles in the smaller teams. In the case of Project S1, this also took place by assigning new roles such as partial architects and lead developers for the newly created teams [P4]. Nevertheless, in the case of a restructuring that involves a separate migration team, care must be taken to ensure that the migration branch and the further development of the system are not separated, resulting in a completely different development. According to P4, the parallel migration and further development in different teams leads to the fact that migration projects are often discontinued because this often has a significant influence on the budget.

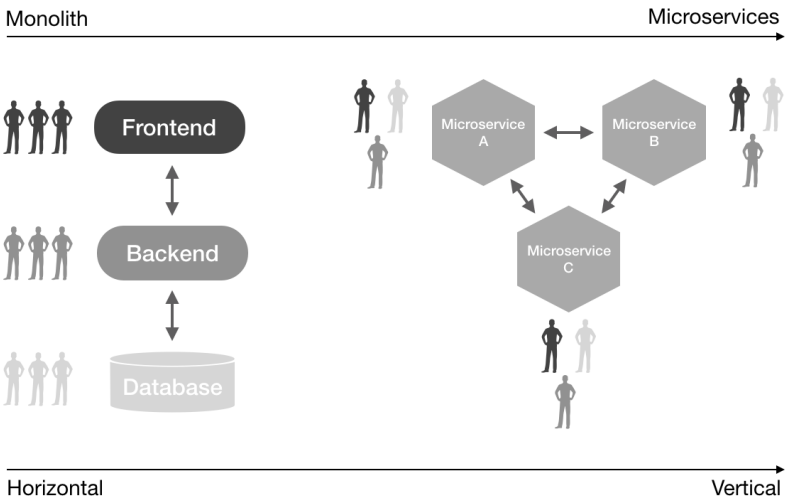

Figure 4.2: Change in organizational structure

4.4 RQ 2: Migration steps

To answer the this research question various steps of a microservice migration were identified. The detailed steps are listed in table 2. Following the phases defined by Wolfart et al. [WAS+21], the identified steps in this study were also divided into specific phases. In contrast to Wolfart et al [WAS+21], the steps were not arranged according to project management guidelines but according to the shared underlying purpose. This resulted in three main phases. The planning phase is used to initially collect all the information required for the migration project and condense it into an overall concept. The implementation phase comprises all organizational and technical steps that are carried out during the actual migration. The validation phase is an additional phase in which the results of the implementation phase and the migration process itself are validated.

Stage	Steps	Factors
Planning	Analysis of drivers [P1, P2, P4]	-
	Analysis of legacy system [P1, P3, P4, P6, P7]	Complexity of legacy system
	Concept design [P2, P4, P5, P6, P7]	Engagement of stakeholders Complexity of legacy system Limitation of the organizational structure
Execution	Analysis of new requirements [P1, P3, P6, P7]	Composition of new features
	Identification of microservice candidates [P1, P2, P4, P5, P6, P7]	Cost & effort estimation Importance of migration Complexity of legacy system
	Consideration of extraction feasibility [P1, P2, P4, P6]	Technical & architectural knowledge
	Infrastructure preparation [P1, P3, P4, P6, P7]	Deviation of release & deployment strategies Infrastructure limitations
	Refactor monolith [P1, P3, P4]	Complexity of legacy system
	Microservice implementation [P1, P4]	Difference of data consistency
Validation	Validation of microservice [P1, P2, P4, P7]	-

Table 4.5: Identified steps and stages

4.4.1 Identification & analysis of drivers

The initial step for a migration project is the analysis of the migration drivers. Project Manager P2 also describes this step as the first assessment with the stakeholders. In this step, besides the actual apparent issues, also existing requirements and long-term goals of the system set by the stakeholders should be taken into account. In the case of the S1 assessment, already implemented and new requirements were also discussed with the customer, which were later taken into account in a concept for the migration process. Therefore, it is important that all stakeholders are involved in order to establish a strong shared commitment already at the beginning. Wolfart et al. [WAS+21] also point out the significance of this step, as it is critical to avoid unsuccessful modernization and waste of resources.

4.4.2 Analysis of legacy system

The analysis of the system takes place at an early stage of the migration process. It serves as a first technical assessment of the system and has also the purpose to gain knowledge about its architecture and implementation [P4]. The source code has proven to be the central artifact for the analysis of the legacy system [P3, P4, P6, P7]. Wolfart et al. [WAS+21] also found that source code is the most common artifact besides development artifacts. Based on this, further insights can be gained with tools such as ArchUnit or SonarQube to determine dependencies of services and the underlying code parts in the system [P4, P1]. In addition to dependencies, the existing module structure can also be identified, which, depending on the degree of modularization, needs to be smoothed out later on [P4]. In addition to these artifacts, various documents can also support gaining knowledge about the system, e.g., architecture documentation or interface documentation. Requirements documents have also proven helpful in this respect, since they describe the features in their completeness [P1].

4.4.3 Concept design

After the analysis of migration drivers, new requirements and the technical assessment an overall concept will be created. Initial ideas for technical implementation should also be included [P2, P5, P6, P7]. The first budget estimates are also made on this basis. In this step, the effort and budget estimation factor is at its highest because the technical and business sides largely meet in the concept and, according to P2, the “price tag“ is defined. Software architect P4 states that it is important to clarify that a redesign on the business side must be done from the beginning, which is very time-consuming. In his experience, this aspect is often overlooked during the conceptual stage of migration projects. At this stage, it is also important to consider the needed to further develop the system, as the effort of the redesign is added on top, which, according to P4, doubles the work of all stakeholders. In the case of S1 a concept was also developed and discussed with the customer before the migration activities started. At this stage, all requirements were also reviewed again, which in this case also served to determine the required budget based on various rounds of estimation involving different teams. In addition, the administration effort for the long-term operation of the architecture should also be taken into account [P5].

4.4.4 Analysis of new requirements

In this step, requirements for the further development of the system are evaluated. These can be new requirements or requirements that were included in the initial assessment with the customer. They are estimated in order to plan which features can be implemented in the upcoming sprints in terms of workload and functionality [P3, P6, P7]. Therefore, it is important to consider to what extent the requirements are self-contained or affect current functionality in the monolith [P1]. Based on this analysis, it is possible to assess the extent to which the requirements will be incorporated into the migration process. This analysis is fundamental to assessing the extent to which the migration process can be incorporated into the implementation of the requirement and thus make progress.

4.4.5 Identification of microservice candidates

The identification of microservice candidates is the first fundamental step, which in a narrower sense refers to the migration of functionality in the monolith. This step also concludes the actual decomposition of the monolith and therefore defines how the system will be able to evolve and scale [WAS+21]. Various approaches can be used to carry out this process. Some of them were already presented in Section 2. Several participants mentioned the application of DDD in the projects to clarify the domains of the system and to consider what to extract on the basis of this domain [P1, P4]. Furthermore, methods such as event storming can also be used to create an abstract domain model for the system based on domain events [P6,P7]. In this step, the complexity of the system, the estimated cost and the importance of migration have their greatest influence. Especially the estimated costs and the importance of the migration determine how far and how detailed the decomposition of the monolith takes place. According to several participants, there was no intention from the customer to decompose the entire monolith because the requirements were service-specific [P1, P5]. This is also influenced by the estimated costs, which subsequently limit the ability to operate in the migration. In the case of S1, this is due to a fixed price quote, which is common in the IT services industry [P2]. According to P2, this results in certain parts being included and certain parts being omitted if they are too time-consuming and the effort therefore exceeds the budget.

4.4.6 Feasibility of extraction

This step involves the fundamental consideration of how the identified microservice candicates should be extracted. In addition, the integration of the service with the monolith and the environment is also considered [P1]. The biggest influence during this step is the existing knowledge about the different technologies and architectures. P6 stated that in one of his migration projects, a long research process was conducted on data migration of the service. In agreement with P1 and P4, he also stated that in the case of S1, several prototype implementations were performed as part of a proof-of-concept. For this purpose, use cases were also outlined with all stakeholders. In addition, P2 pointed out that experienced colleagues were also included to benefit from their experience with the existing environment in which they had already worked.

4.4.7 Infrastructure preparation

Another step is the preparation of the infrastructure for a microservice architecture, which can be done partly in parallel with the development of the services [P3]. This step is driven by the limiting factors of the infrastructure of a monolith. For this purpose, the actual proxy mechanism including load balancing is set up as well as the respective API gateway required for the service [P1, P3, P6, P7]. This is necessary to perform requests to the existing system as well as to the new microservices according to the strangler pattern [P1]. Moreover, IP, port forwarding and firewall activations can be performed, depending on the restrictions of the infrastructure [P5]. Another factor that becomes apparent in this step is the difference in the deployment and release strategies related to the different architectures, which require changes to CI/CD pipelines. In the case of MS2 development, this included specific concepts for automated and productive deployments on cloud platforms such as CloudFoundry [P6]. According to P4, this step was also necessary to technically implement the elaborated deployment strategies with regard to "zero downtime deployments", so that it is ensured that one instance is always up and running and a new one can be deployed at the same time.

> *"A colleague had to fiddle around with these infrastructure topics, since they also changed the build infrastructure, i.e. things like continuous integration and so on, which took quite a bit of time."*
>
> P6, *Software Engineer*

4.4.8 Monolith refactoring

Depending on the state of the legacy application it is necessary to refactor the monolith. The factor that significantly affects this step is the complexity of the monolith, as the module structure of the monolith may need to be refined to separate individual modules before they can be physically separated [P1, P3, P4]. Furthermore, it is important to minimize interdependencies of individual code components identified during the initial analysis. In addition, when using event sourcing, the necessary interfaces often need to be implemented. P5 also mentioned that prior to the actual development of MS2, interfaces were created in the monolith to retrieve the data processed by the microservice. During this step, P4 found that removing cyclic dependencies and design of interfaces on the monolith to use messaging was extremely complex.

4.4.9 Microservice implementation

After the microservice candidate is identified, the implementation is performed where the functionality is then migrated from the monolith. The implementation may be accompanied by adjustments to the monolith and partial adaptation of the infrastructure. Since the database is also decomposed in this step, the factor of data consistency arises. In addition, this step may require recoding entire functionalities [P1,P4] when combining different technologies from different programming languages. P4 stated that when MS1 was implemented, parts of the code could be converted one-to-one to Kafka Stream. However, switching to other technologies may lead to complete

recoding. In this context, P4 stated that automatic transcoding did not lead to a satisfactory results. The participants did not mention how the choice of technology affects this step as a factor. Since this factor is nevertheless relevant, it is also taken into account for the migration approach.

4.4.10 Validation of microservice

After successful extraction, the quality requirements defined by the stakeholders for the microservice should be verified [P2]. In the environment, the behavior of the deployed microservices can be checked through monitoring [P1]. Furthermore, it is also useful to validate the old implementation against the new implementation to check for performance improvements [P1, P4].

4.5 RQ 3: Decisions during a migration

The migration to microservices involves a variety of decisions of different types in different phases. The identified decisions made by the participants are listed in Table 4.6. This also shows the factors on which the decisions are based as well as the steps in which they occurred. Figure 4.3 shows a graphical blueprint of how the decision points are illustrated in the following. Similar to the decision-making process of Ayas et al. [ALH21] , a decision on a certain DP can be made between exactly one or more options, which are shown on the right. However, unlike Ayas et al. [ALH21] the stakeholders involved in the decision (on the top) and the influencing factor on the basis of which the decision is made (on the left) are indicated for each DP.

Decision Point	Factor	Step
Migration feasibility	Engagement of stakeholders Complexity of legacy system	Create and evaluate concept
Composition of legacy system	Complexity of legacy system	Refactoring monolith
Implementation of new requirements	Composition of new features	Analysis of new requirements
Procedure of microservice implementation	Complexity of legacy system Technical & architectural knowledge	Consideration of extraction feasibility
Restructuring of the organization	Organizational limits	Concept design Before microservice implementation

Table 4.6: Identified decision points

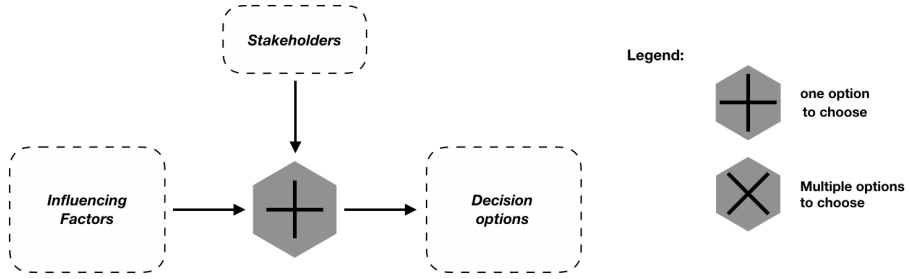

Figure 4.3: Graphical notation for DP

4.5.1 Migration feasibility

The first decision point has been identified in the early planning phase. The elaboration of the concept, which takes into account information from the technical inventory, business requirements and the actual migration drivers, is used to determine the extent to which a migration can be realized. This decision is made primarily on the basis of stakeholder engagement and the estimated effort and costs. Both should be available for this long-term process. Then it can be decided if a microservice migration is viable or if a modularization of the monolith is sufficient for the time being. Several participants mentioned that also modular monolith was investigated as an option. However, this was discarded due to the fundamental drivers of scalability and maintainability which can be better solved using microservices [P2, P4].

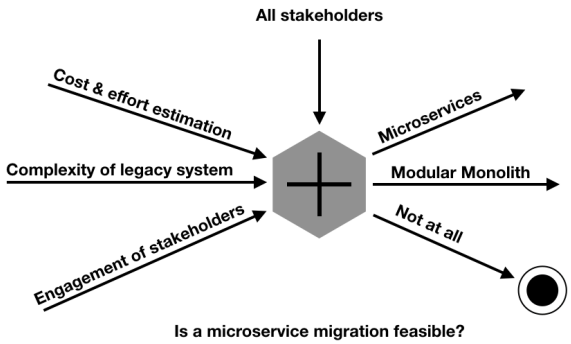

Figure 4.4: DP-Migration feasibility

> *we had initially considered keeping our monolith and using modules or something like that instead of really microservices. Because if you simply say that you put it in your own modules, then you actually have the separation in exactly the same way, even if it's actually still one application*
>
> P3, *Developer*

4.5.2 Composition of legacy system

Before the monolith can be split into possible microservice candidates, it might be refactored depending on its composition. Depending on the presence of mutual dependencies in the monolith, it is decided whether the module structure needs to be refactored [P3, P4].

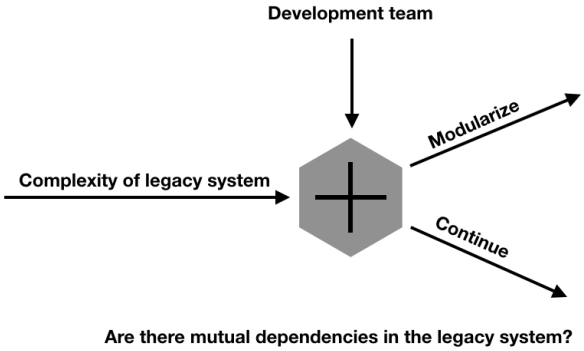

Figure 4.5: DP-Composition of legacy system

> *So we decided to smooth the module structure of the monolith in a first step before it can be physically split*
>
> P4, *Software Architect*

4.5.3 Implementation of new requirements

A decision point also arises in the early stage of the execution phase when the stakeholder requirements to be implemented are analyzed. The development teams are significantly involved in this decision point. They decide how the requirements are to be implemented based on their scope[P2, P3]. The decision is based on the degree to which the functionalities on which the requirements depend are interconnected in the monolith[P1]. This is also related to the complexity of the system. Effort can be estimated based on these factors, leaving the development team with two options. If the effort is low, the associated functionality is extracted and implemented together with the requirement as a separate microservice. If the effort is too high, the requirement is implemented directly in the monolith [P1].

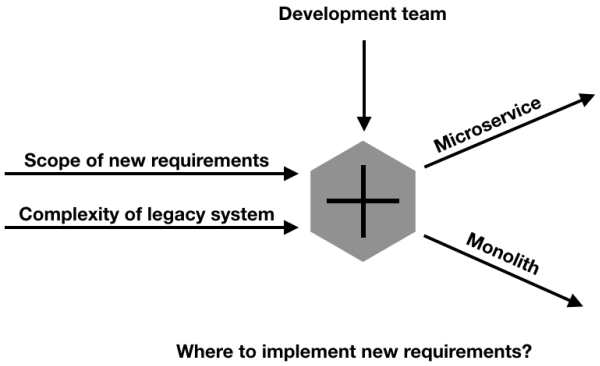

Figure 4.6: DP-Implementation of new requirements

> *...we simply evaluated it and said, where does this fit in, what does it cost us to implement this as a separate service or to pull it out with a feature complex. Is that too expensive, is that feasible? If it wasn't feasible, it went into the monolith.*

P1, *Software Engineer and Software Architect*

4.5.4 Procedure of microservice implementation

When considering the extraction feasibility, the technical and arcnitectural experience of the development team plays an important role. This can be additionally influenced by the complexity of the entire system. For this reason, a self-assessment of the skills of the development team should take place prior to the extraction process in order to identify necessary knowledge gaps in the team [New19]. Based on these factors, a decision can be made on how to implement microservices when there is a certain lack of knowledge. Several participants mentioned that a proof-of-concept was carried out for the migration of S1 [P1, P7]. In this process, a detailed use case was sketched with the stakeholder [P1] and different prototypes were created [P1, P4, P6]. Furthermore, best practices were also attempted to be followed [P1]. Project manager P2 also explained that he also asked for assistance from experienced colleagues who had already worked with the environment. Based on this information, three different decision options for the implementation procedure were identified. The implementation of a proof-of-concept with different prototypes, the following of known best practices as well as the consultation of expert knowledge. All of these options can also be combined with each other. The stakeholders involved at the decision point were identified as the development team, the project management and the customer. The customer and project management should be involved to some extent, as the necessary acquisition of specific knowledge can delay the project and also increase costs.

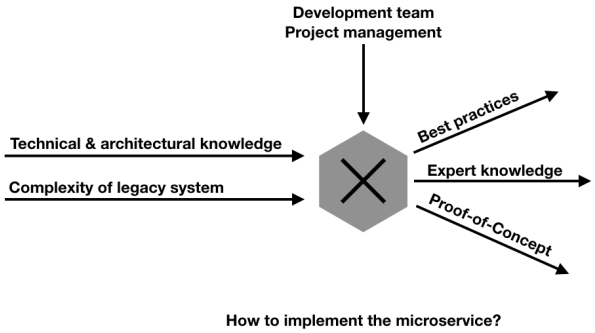

Figure 4.7: DP-Procedure of microservice implementation

4.5.5 Restructuring of the organization

> *...at this point we said, okay, there is, just one team that only takes care of this one or these two services. And from from development to operation in the end (...), so that was a structural change. That means that one team has become two and at the end three.*
>
> P4, *Software Architect*

During migration, changes are required as the current organizational structure reaches its limits as the migration continues and the new microservices evolve. Newman [New19] also points out that a change of code ownership is necessary when migrating towards a microservices architecture as the number of developers and microservices increases. In the process, the structure will gradually change to an organizational structure with corresponding agile and cross-functional teams according to the microservice paradigm. In this context, several participants also mentioned that the increasing number of developers led to the decision to migrate as a separate team [P1, P2, P3]. Other participants mentioned that teams were split up to operate and develop specific microservices [P4, P7]. Based on the statements of the participants, it was determined that this DP occurs both during the concept design, when capacity is checked and during the execution of the migration, when new microservices are implemented .Therefore, it should be considered at these stages whether restructuring is necessary based on the current status of the project or whether the current structure can be maintained.

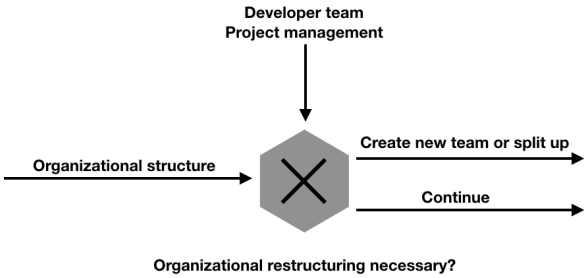

Figure 4.8: DP-Restructuring of the organization

4.6 RQ 4: Migration approach

In this chapter, the migration approach based on the strangler pattern is presented. Figure 4.9 shows the entire approach with the different steps, DPs and resulting artifacts. The steps are grouped into the following phases: *Planning*, *Execution*, *Validation*. In the following, the different components of the approach will be discussed. Reference is also made to the steps, decisions and factors already identified. In addition, further components were added to complete the approach. These additional components also emerged from the self-reflections on the past migrations during the interviews. During these reflections participants identified several aspects that could have been improved in the migration process. An important component besides additional steps is the introduction of artifacts. Artifacts are products produced in a step by the stakeholders and are important for a successful and smooth migration.

4.6.1 Planning phase

In this phase, all the information necessary for the decision to migrate is collected. Migration-related artifacts are also created in this phase, which are also used during the execution of the migration.

Analyze drivers

Referenced migration step: Identification & analysis of drivers (Section 4.4.1)

At the beginning of a migration project, the drivers must be determined. Based on these, it can be concluded to what extent a change in the architecture of the legacy application is necessary. All stakeholders should be involved to discuss the drivers and already find a common understanding, which ensures a collective commitment of all stakeholders. In addition, all requirements for planned enhancements to the legacy application's functionality and already completed enhancements under development should also be included. This allows an initial overall status of the system to be determined. Moreover, documents can already be collected, which are necessary for the subsequent analysis of the legacy system.

Analyze legacy system

Referenced migration step: Analysis of the legacy system (Section 4.4.2)

The identification and analysis is followed by a technical assessment of the legacy system. In addition to the source code, additional tools as mentioned in Section 4.4.2 can also be used for this purpose. Also documents that could be obtained in the previous step by the customer can be used. The analysis also allows assumptions to be made about the existing module structure. In this context, mutual dependencies can be identified in the system that need to be resolved later in the migration process in order to extract individual services. This information also allows to determine the necessary technical effort for refactoring the application, which is an essential part for the development of the concept in the next step.

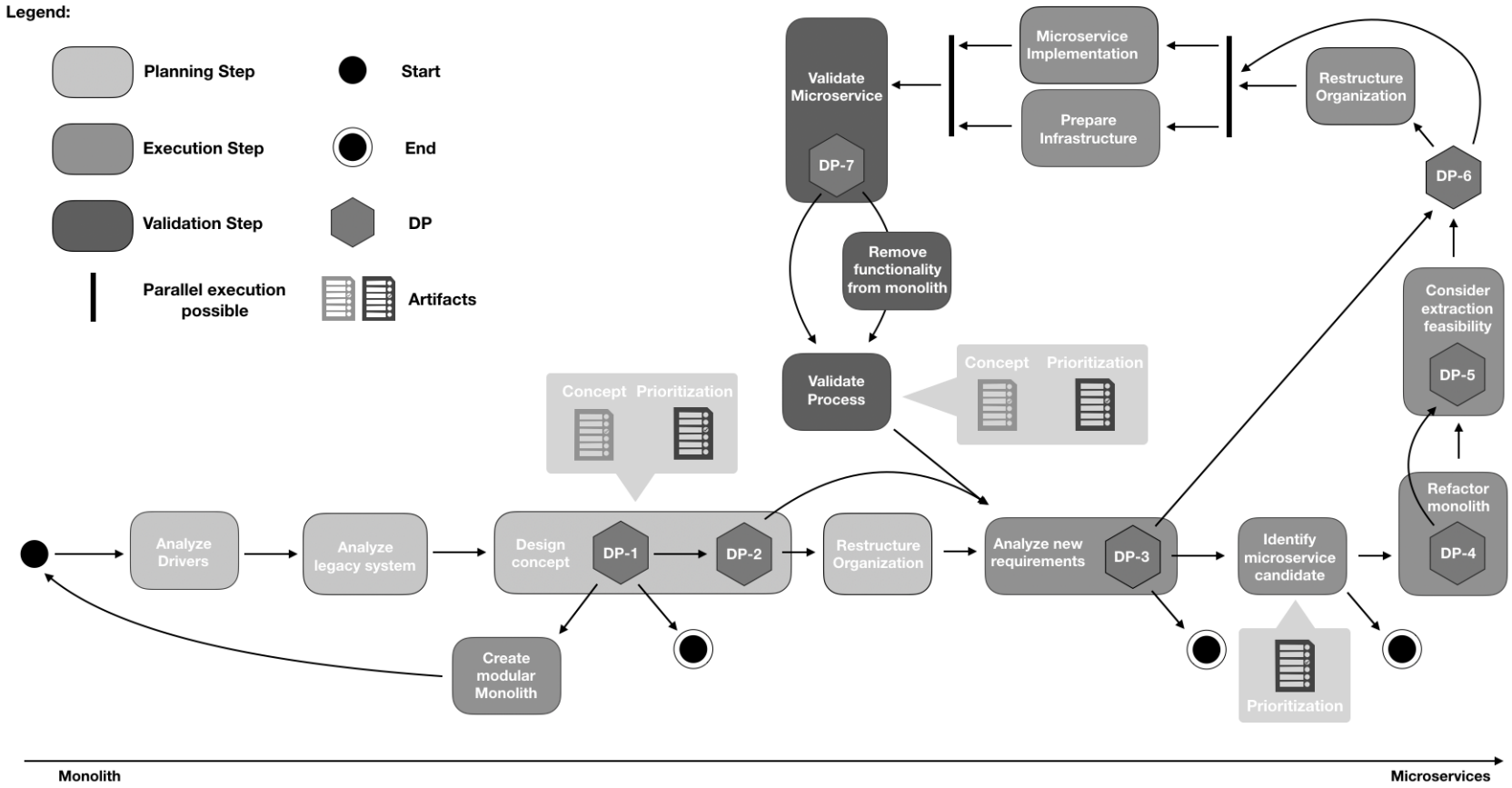

Figure 4.9: Migration approach

Design concept

Referenced migration step: Concept design (Section 4.4.3)

After the identification and analysis of the drivers as well as the analysis of the legacy system, a concept should be developed. This concept contains a rudimentary technical implementation for the migration as well as all requirements that must be implemented initially with the migration. This also allows the effort and costs to be estimated. An artifact of this step is therefore the concept itself. This concept should contain all the intentions of the stakeholders and thus serves as an approximate roadmap for the process. This roadmap should be continuously communicated to all stakeholders and should be adapted during the migration process in order not to deviate from the original path during this long-lasting process. Both P2 and P4 stated that constant communication and involvement of all stakeholders throughout the entire duration would have significantly improved the progress of the migration. Based on this concept, it is possible to determine whether there is support for a microservice migration from all stakeholders. Therefore, it also makes sense for all stakeholders to check whether their capacities are sufficient for this long-term process, because according to P4, a migration is to be considered a redesign and thus leads to a doubling of the effort for all stakeholders. In this way, the problem mentioned by P7 of overworked actors who no longer have time because they had to manage other projects on the side is solved. In this way, a possible overload of stakeholders can also be identified early on. P7 mentioned in this context that there were problems with overworked actors who had less time during the project because they had to manage other projects at the same time. As a second artifact, this process should lead to a prioritization for the extraction of microservice candidates according to defined criteria that can later be adapted to other circumstances. This can prevent the constant restructuring of the extraction order by the stakeholders during the process described by P7. Li et al. [LML20] also propose to define a service prioritization to bring maximum benefits. However, the creation of this artifact depends on whether a microservice migration in DP 1 is decided on the basis of the overall circumstances.

> *You would have had to invest time to create a proper concept that would have been convincing in the long term. The management would certainly have said, oh, that makes sense to do that. But because the willingness wasn't there, that's what happened.*
>
> P7, *Fullstack Developer*

> *...we should also have communicated constantly with the stakeholders and involved all stakeholders in the development to make everyone transparent and keep bringing people along...., so this didn't go so well.*
>
> P4, *Software Architect*

DP-1

Referenced DP: Migration feasibility (Section 4.5.1)

This DP should be used to decide whether a microservice migration is feasible or whether a refinement of the existing architecture is sufficient for the time being. The effort required to operate the target architecture should also be considered. P6 also points out the increased effort required to operate a microservice architecture. Therefore, Newman [New19] also highlights the benefits of a monolith, which is why a migration is not always necessary, as microservices have costs and one has to decide if the cost is worth the options that could be taken. Even with a modular monolith, it is possible to come back to the microservice migration option again later.

DP-2

Referenced DP: Restructuring of the organization (Section 4.5.5)

After the decision for a microservice migration in DP-1, it should be decided whether a restructuring of the organization is already reasonable. This can be necessary as the number of developers may have to be scaled due to the increased capacity required for the migration. In addition to P2, P3 also states in this context that the team has become too large to work together on the application at this stage.

Restructure organization

Referenced migration step: -

In this step, the change of the organization is performed. Initially, the migration can be carried out in a separated team.

4.6.2 Execution phase

The planning phase is followed by the execution of the migration. Following the strangler pattern, this is an incremental process where microservices are extracted step by step. Several participants mentioned that the extraction process was repeated for each service [P1, P4, P7]. Therefore, this approach treats the execution phase, along with the validation phase, as a cyclic process that continues until the monolith is finally decomposed or the requirements of all stakeholders are fulfilled.

Analyze new requirements

Referenced migration step: Analysis of new requirements (Section 4.4.4)

Analyzing new requirements is the first step of the execution phase. These are upcoming requirements that need to be implemented. In this context, is determined how they can be implemented and to what extent related functionalities can be extracted in the monolith.

DP-3

Referenced DP: Implementation of new requirements (Section 4.5.3)

In this DP, the analysis of the upcoming requirements is used to decide whether a microservice candidate can also be extracted and therefore a migration activity can be performed. If the requirement is a self-contained has its own bounded context, it can be directly implemented as a separate microservice. If the requirement depends on a self-contained functionality in the monolith, it can be extracted with the implementation of the requirement. If the requirements depend on different functionalities in the monolith and the extraction is too complex for the current sprint, it should be implemented in the monolith. Therefore, Figure 4.10 visualizes the different options. Due to the prioritization there is the possibility that the migration is terminated if the customer's requirement was only to fix existing problems and not to further decompose the monolith. For this purpose, an additional option was added to the DP to stop the migration process, as the migration of S1 was also stopped when all customer requirements were implemented.

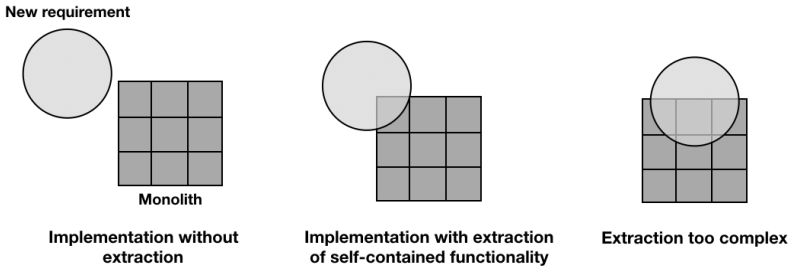

Figure 4.10: Scope of requirements related to the monolith

Identify microservice candidates

Referenced migration step: Identification of microservice candidates (Section 4.4.5)

In this step, the possible microservice candidates are identified with the approaches presented in Section 2.3. The microservice candidates are selected based on the prioritization. For example, if solving performance problems is the highest priority, the problems can be assigned to features in the monolith by analyzing logs as described by P1. This allows approaches such as DDD to be applied more specifically to certain areas in the monolith. However, if the intention of the stakeholders is to completely decompose the monolith, a new domain design should be applied to the complete monolith as mentioned by P4. However, this also requires more effort. In addition, if the monolith has already been successfully decomposed at this stage, the migration process can be terminated.

DP-4

Referenced DP: Composition of legacy system (Section 4.5.2)

After deciding which service candidate to extract next, it is necessary to decide whether the monolith needs to be refactored first, depending on its module structure.

Refactor monolith

Referenced migration step: Monolith refactoring (Section 4.4.8)

In this step, the refactoring of the monolith is performed. Moreover, the circular dependencies identified during the initial analysis of the system will be resolved. As mentioned by P7 and P4, the necessary interfaces for the microservice can already be integrated. In this process, P4 described a procedure similar to the branch-by-abstraction pattern mentioned in the section. This pattern can be applied in this step to integrate the microservice with the monolith when data needs to be exchanged between them.

Consider extraction feasibility

Referenced migration step: Feasibility of extraction (Section 4.4.6)

After refactoring the monolith, it should be decided which appropriate technology is going to be used for the microservice implementation [P1, P4]. In addition, the development team should also assess the level of knowledge about the different technologies to avoid sudden trial and error as described by P2. In this context, Newman [New19] also points out the skills required to perform the implementations during a migration and suggests skill charts as an example to identify possible knowledge gaps of the individual developers and the whole team. This can then be used to decide which implementation method to choose.

> “ *... I didn’t want to say that it was a complicated technology and that’s why it took so long, but people didn’t know that much about it yet. And accordingly, there was some experimentation. And that experimentation takes time, which you don’t really have usually at certain points in the project. At least not when the whole team is trying. That would be better, of course, if there were some experienced people with us.* ”
>
> P2, *Project Manager and Requirements Engineer*

DP-5

Referenced DP: Migration feasibility (Section 4.5.1)

Once the technology to be used and the general level of knowledge of the development team are determined, a decision can be made about the type of feasibility. This can be done using best practices found through a research process, by consulting experts, or even by prototyping a use case in form of a proof-of-concept. The various options can also be combined, as described in Section 4.5.4.

DP-6

Referenced DP: -

After all possible preparations have been made on the monolith, it should be decided to what extent a separate team will be established for the service to be developed. After all possible preparations have been made on the monolith, it should be decided to what extent a separate team will be established for the service to be developed.

Restructure organization

Referenced DP: Restructuring of the organization (Section 4.5.5)

In this step, a change of the organization is performed again. Care should be taken to ensure that the team is cross-functional.

Prepare infrastructure & implement microservice

Referenced migration step: Infrastructure preparation (Section 4.4.7)
Referenced migration step: Microservice implementation (Section 4.4.9)

In this step, the necessary infrastructure is prepared and the micorservice is implemented. This step can also be done in parallel as mentioned by P6. When setting up the infrastructure, the necessary load balancers and API gateways for the functionality are set up according to the strangler pattern described in detail in Section 2.2.1. Based on the developed release strategy, the necessary CI/CD pipelines can also be set up. With regard to the development of the microservice, code parts can also be transformed as described in Section 4.4.9, depending on the technology used.

4.6.3 Validation phase

The validation phase takes place after the execution phase has been completed. This phase ensures that the process and the implemented microservice satisfy the requirements of all stakeholders.

Validate microservice

Referenced migration step: Validation of microservice (Section 4.4.10)

The first step of the validation phase is the validation of the micorservice. Therefore, monitoring tools with distributed tracing such as Elastic APM[3] or Zipkin[4] can be used to verify how the microservice performs in the environment. Especially when solving performance problems, possible improvements can be detected. Since both systems coexist for a certain time when using the

[3] https://www.elastic.co/blog/distributed-tracing-opentracing-and-elastic-apm
[4] https://zipkin.io/

strangler pattern, comparative measurements can also be made. Due to the proxy mechanism used in the strangler pattern, canary releases[5] or dark launching[6] can also be performed to observe the microservice performing under productive conditions.

DP-7

Referenced DP: -

In this DP it is decided whether the microservice meets all qualitative and quantitative requirements and the extracted functionality can be removed from the monolith. If this is not the case, the entire cycle can be run again in a minimized form to solve the issues of the microservice.

Validate process

Referenced DP: -

In addition to the successful validation of the microservices, a validation of the migration process should also be performed. In this way, any problems that arose during the migration cycle can be discussed with everyone involved. In this way, optimizations can be made to the process itself. The concept and prioritization originally established should also be used. These documents can be refined again depending on new or changing requirements. This recurring communication allows the commitment of all involved to be maintained and the vision of continuous migration to be pursued. Afterwards, the entire migration cycle can be performed again.

4.7 Threats to validity

Many threats are identical to other researches in the MSA domain using similar research methods to those used in this study [DLM18] [FBWZ19] [ALH21]. The threats that have been identified in this work are described as follows.

Sampling method & sample size

In regards to the interview process, threats may arise from the sampling method and the sample itself. In this study the participants were recruited based on the reference of Adesso employees. The choice of this method may have a negative impact on the resulting sample. To address at least some concerns about the generalizability of the data obtained, an attempt was made to select a heterogeneous sample consisting of interview participants with different project roles and work experiences. Nevertheless, the sample size in this study is very small with regard to the large amount of MSA software engineers involved in microservice migrations. Moreover, it should be mentioned that this work does not claim representativeness of the study demographics for the entire software industry, as all participants are provided by the same company.

[5] https://martinfowler.com/bliki/CanaryRelease.html
[6] https://martinfowler.com/bliki/DarkLaunching.html

Participant experiences

A further threat arises because of the experience of the interview participants. The participants had very different experiences and some had experience with microservices but little experience with migration

Migrations were performed long ago

Since the migration projects in which the participants of the interviews had taken part were often further in the past, it was also not always possible to gain detailed findings about the methods or approaches used.

5 Conclusion

This chapter summarizes the thesis and suggests possible directions for future work.

5.1 Summary

In summary, this work addressed the process of a gradual microservice migration by a systematic application of the strangler pattern. Therefore, basic concepts of influencing factors, steps and decisions were introduced, which are part of a decision process for a continuous microservice migration. Through conducting interviews with various stakeholders who have already performed microservice migrations, their experiences were used to identify the various concepts in a microservice migration. To this end, the various factors that influence a microservice migration were first identified. In addition, the different steps of a migration were identified and the factors were assigned to the steps. Based on these results, a migration approach was developed to guide developers and architects through the decision-making process for microservices migration. In order to achieve this, the individual steps were systematically linked to each other through the various decision points, resulting in a decision guidance for continuous microservice migration. Furthermore, a retrospective view of the participants on their past migration projects enabled the identification of further necessary steps and decision points, which completed the approach.

5.2 Future Work

5.2.1 Validation of the approach

The developed approach has not yet been applied in practice. Therefore, it could be practically applied as part of a microservice migration. The knowledge gained from this could be used to improve the approach.

Bibliography

[] *Adopting Microservices at Netflix: Lessons for Architectural Design*. https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/. Accessed: 2021-10-26 (cit. on pp. 19, 45).

[ALH21] H. M. Ayas, P. Leitner, R. Hebig. "Facing the Giant: a Grounded Theory Study of Decision-Making in Microservices Migrations". In: *CoRR* abs/2104.00390 (2021). arXiv: 2104.00390. URL: https://arxiv.org/abs/2104.00390 (cit. on pp. 35, 52, 64).

[BHJ16] A. Balalaie, A. Heydarnoori, P. Jamshidi. "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture". In: *IEEE Software* 33.3 (2016), pp. 42–52. DOI: 10.1109/MS.2016.64 (cit. on p. 19).

[Cha01] K. Charmaz. "Grounded Theory: Methodology and Theory Construction". In: vol. 10. Dec. 2001, pp. 6396–6399. ISBN: 9780080430768. DOI: 10.1016/B0-08-043076-7/00775-0 (cit. on p. 33).

[Con67] M. E. Conway. "How do committees invent." In: 1967 (cit. on p. 46).

[DGL+16] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina. "Microservices: yesterday, today, and tomorrow". In: *CoRR* (2016) (cit. on p. 19).

[DLM18] P. Di Francesco, P. Lago, I. Malavolta. "Migrating Towards Microservice Architectures: An Industrial Survey". In: *2018 IEEE International Conference on Software Architecture (ICSA)*. 2018, pp. 29–2909. DOI: 10.1109/ICSA.2018.00012 (cit. on pp. 31, 64).

[EB18] S. Eski, F. Buzluca. "An Automatic Extraction Approach: Transition to Microservices Architecture from Monolithic Application". In: *Proceedings of the 19th International Conference on Agile Software Development: Companion*. XP '18. Porto, Portugal: Association for Computing Machinery, 2018. ISBN: 9781450364225. DOI: 10.1145/3234152.3234195. URL: https://doi.org/10.1145/3234152.3234195 (cit. on p. 24).

[FBWZ19] J. Fritzsch, J. Bogner, S. Wagner, A. Zimmermann. "Microservices Migration in Industry: Intentions, Strategies, and Challenges". In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2019, pp. 481–490. DOI: 10.1109/ICSME.2019.00081 (cit. on pp. 25, 31, 35, 45, 64).

[FBZW19] J. Fritzsch, J. Bogner, A. Zimmermann, S. Wagner. "From Monolith to Microservices: A Classification of Refactoring Approaches". In: *Lecture Notes in Computer Science* (2019), pp. 128–141. ISSN: 1611-3349. DOI: 10.1007/978-3-030-06019-0_10. URL: http://dx.doi.org/10.1007/978-3-030-06019-0_10 (cit. on pp. 15, 23).

[FL14] M. Fowler, J. Lewis. *StranglerFigApplication*. 2014. URL: http://martinfowler.com/articles/microservices.html (visited on 10/26/2021) (cit. on p. 20).

[FML17] P. D. Francesco, I. Malavolta, P. Lago. "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption". In: *2017 IEEE International Conference on Software Architecture (ICSA)*. 2017, pp. 21–30. DOI: 10.1109/ICSA.2017.24 (cit. on p. 16).

[Fow04] M. Fowler. *StranglerFigApplication*. 2004. URL: https://martinfowler.com/bliki/StranglerFigApplication.html (visited on 10/26/2021) (cit. on p. 20).

[Fow14] M. Fowler. *BranchByAbstraction*. 2014. URL: https://martinfowler.com/bliki/BranchByAbstraction.html (visited on 10/26/2021) (cit. on p. 21).

[Fow15] M. Fowler. *MicroservicePremium*. 2015. URL: https://martinfowler.com/bliki/MicroservicePremium.html (visited on 10/26/2021) (cit. on p. 41).

[GKGZ16] M. Gysel, L. Kölbener, W. Giersche, O. Zimmermann. "Service Cutter: A Systematic Approach to Service Decomposition". In: Sept. 2016, pp. 185–200. ISBN: 978-3-319-44481-9. DOI: 10.1007/978-3-319-44482-6_12 (cit. on p. 23).

[HA05] S. Hove, B. Anda. "Experiences from conducting semi-structured interviews in empirical software engineering research". In: *11th IEEE International Software Metrics Symposium (METRICS'05)*. 2005, 10 pp.–23. DOI: 10.1109/METRICS.2005.24 (cit. on pp. 31, 32, 36).

[Hai16] C. Haight. *APM Needs to Prepare for the Future*. Tech. rep. Gartner Research, 2016 (cit. on p. 43).

[HS17] W. Hasselbring, G. Steinacker. "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce". In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 2017, pp. 243–246 (cit. on p. 15).

[HW04] G. Hohpe, B. WOOLF. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. The Addison-Wesley Signature Series. Prentice Hall, 2004. ISBN: 9780321200686. URL: http://books.google.com.au/books?id=dH9zp14-1KYC (cit. on p. 21).

[KH18] H. Knoche, W. Hasselbring. "Using Microservices for Legacy Software Modernization". In: *IEEE Software* 35.3 (2018), pp. 44–49. DOI: 10.1109/MS.2018.2141035 (cit. on pp. 16, 24).

[KMM18] M. Kalske, N. Mäkitalo, T. Mikkonen. "Challenges When Moving from Monolith to Microservice Architecture". In: *Current Trends in Web Engineering*. Ed. by I. Garrigós, M. Wimmer. Cham: Springer International Publishing, 2018, pp. 32–47. ISBN: 978-3-319-74433-9 (cit. on pp. 35, 44, 46, 47).

[KZH+20] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, D. Kröger. "Microservice Decomposition via Static and Dynamic Analysis of the Monolith". In: *CoRR* abs/2003.02603 (2020). arXiv: 2003.02603. URL: https://arxiv.org/abs/2003.02603 (cit. on pp. 16, 23).

[Lea17] P. Leavy. *Research Design: Quantitative, Qualitative, Mixed Methods, Arts-Based, and Community-Based Participatory Research Approaches*. Guilford Publications, 2017. ISBN: 9781462530014. URL: https://books.google.de/books?id=4HGnDQAAQBAJ (cit. on p. 30).

[LML20] C.-Y. Li, S.-P. Ma, T.-W. Lu. "Microservice Migration Using Strangler Fig Pattern: A Case Study on the Green Button System". In: *2020 International Computer Symposium (ICS)* (2020), pp. 519–524 (cit. on pp. 16, 21, 25, 35, 40, 59).

[Luk18] M. Luksa. *Kubernetes in Action*. Manning Publications, 2018 (cit. on pp. 15, 43).

[Mil14] M. B. Miles. *Qualitative data analysis : a methods sourcebook (3rd ed.)* Thousand Oaks, California : SAGE Publications, 2014 (cit. on pp. 33, 34).

[New15] S. Newman. *Building Microservices*. 1st. O'Reilly Media, Inc., 2015. ISBN: 1491950358 (cit. on pp. 15, 19, 43–46).

[New19] S. Newman. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O´Reilly Media, 2019 (cit. on pp. 16, 19–22, 24, 35, 44, 47, 55, 56, 60, 62).

[NMMA16] I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen. *Microservice Architecture: Aligning Principles, Practices, and Culture*. 1st. O'Reilly Media, Inc., 2016 (cit. on pp. 15, 19).

[Pat02] M. Q. Patton. *Qualitative research and evaluation methods*. Thousand Oaks, 2002 (cit. on p. 31).

[R11] A. E. S. C. R. *An invitation to social research: How it's done (4th ed.)* Wadsworth Cengage Learning, 2011 (cit. on p. 30).

[RH08] P. Runeson, M. Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14 (2008), pp. 131–164 (cit. on p. 29).

[Rob02] C. Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers (2nd ed.)* Blackwell Publishing, 2002 (cit. on pp. 29, 31).

[Rub11] H. J. R. I. S. Rubin. *Qualitative Interviewing: The Art of Hearing Data (3rd ed.)* SAGE Publications, 2011 (cit. on pp. 31, 32, 36).

[Sal09] J. Saldaña. *The coding manual for qualitative researchers*. English. SAGE London, 2009, x, 223 p. : ISBN: 9781847875488 1847875483 9781847875495 1847875491 (cit. on pp. 33, 34).

[SCM19] H. H. Silva, G. F. Carneiro, M. Monteiro. "Towards a Roadmap for the Migration of Legacy Software Systems to a Microservice based Architecture". In: *CLOSER*. 2019 (cit. on p. 25).

[Sei17] V. Seifermann. "Application Performance Monitoring in Microservice-Based Systems". Bachelor's Thesis. University of Stuttgart, 2017 (cit. on p. 43).

[TLP17] D. Taibi, V. Lenarduzzi, C. Pahl. "Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation". In: *IEEE Cloud Computing* 4.5 (2017), pp. 22–32. DOI: 10.1109/MCC.2017.4250931 (cit. on pp. 25, 31).

[VF21] V. Velepucha, P. Flores. “Monoliths to microservices - Migration Problems and Challenges: A SMS”. In: *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*. 2021, pp. 135–142. DOI: 10.1109/ICI2ST51859.2021.00027 (cit. on pp. 35, 44–46).

[WAS+21] D. Wolfart, W. K. G. Assunção, I. F. da Silva, D. C. P. Domingos, E. Schmeing, G. L. D. Villaca, D. d. N. Paza. “Modernizing Legacy Systems with Microservices: A Roadmap”. In: *Evaluation and Assessment in Software Engineering*. EASE 2021. Trondheim, Norway: Association for Computing Machinery, 2021, pp. 149–159. ISBN: 9781450390538. DOI: 10.1145/3463274.3463334. URL: https://doi.org/10.1145/3463274.3463334 (cit. on pp. 35, 45, 48–50).

[WRH+12] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, A. Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN: 3642290434 (cit. on p. 29).

All links were last followed on March 17, 2018.

A Interview Documents

A.1 Interview guide

Main Questions	Follow-Up Questions
Erzähle mir doch einmal, welche Erfahrungen du mit den vergangenen Microservice Migrationen gemacht hast? *Can you tell me about your experiences with past microservice migrations?*	Was waren die Anforderungen der Stakeholder in Bezug auf die Architektur? ISO 25010
Was wolltet ihr mit einer Microservice Architektur erreichen? *What did you want to achieve with a microservice architecture?*	Kannst du mir die Entscheidungen erläutern, warum sich gegen oder für diese Alternativen entschieden wurde?
Was waren mögliche Alternativen anstelle der Microservice Architektur? *What were possible alternatives instead of microservice architectures?*	
Erzähle mir doch einmal, wie ihr bei der Migration schrittweise vorgegangen seid? *Can you explain step by step how you performed the migration?*	
Welche Einflussfaktoren sind während der Migration aufgetreten? *What influencing factors did you encounter during the migration?*	Wann traten die Faktoren im Migrationsprozess auf? Inwiefern haben sich die Faktoren positiv oder negativ auf das Projekt ausgewirkt? Welche organisationalen Veränderungen gab es? Wie haben sich die Faktoren auf die Projektplanung ausgewirkt?

Wie seid ihr mit den aufgetretenen Einflussfaktoren umgegangen? *How did you deal with the influencing factors that occurred?*	Welche Entscheidungen gab es? Wie habe sich diese ausgewirkt?
Welche Einflussfaktoren traten im Laufe der Migration erneut auf? *What influencing factors did arise again during the migration?*	Wann traten die Faktoren im Migrationsprozess erneut auf?
Wie konntet ihr feststellen, ob die Migration wie geplant funktioniert hat? *Can you explain how were you able to determine if the migration process was working as expected?*	Inwiefern gab es quantitative oder qualitative Messungen? Inwiefern wurden Methodiken zur Reflexion des Migrationsprozess einbezogen?
Kannst du beschreiben, wie ihr die Migrationsaktivitäten und die fortlaufende Projektbetrieb priorisiert habt? *Can you describe how you prioritized the migration activities with the ongoing project business?*	Wie wurden neue Kundenanforderungen im Projekt eingebunden? Wie wurden neue Features implementiert?
Was denkst du hätte man im Migrationsprozess besser machen können? *What do you think could have been done better in the migration process?*	
Was war die Vorgehensweise für das Identifizieren der Servicekandidaten? *What was the process for identifying the service candidates?*	
Welche Veränderungen wurden am System und am Monolithen vorgenommen? *What changes were made to the system environment and the including monolith?*	Welche Vorbereitung wurden getroffen? Beispiel: • API Gateway • Load Balancing • CD/CD Pipeline

A.2 Questionnaire

adesso

Questionnaire for the master thesis „How to Strangle Systematically: An Approach and Case Study for the Continuous Evolution of Monoliths to Microservices“

Thank you for agreeing to participate in my master's thesis on "How to Strangle Systematically: An Approach and Case Study for the Continuous Evolution of Monoliths to Microservices".

Demographic questions:

(1) First and last name:

_____

Work-related questions:

Years of professional experience ?

___ years

In which position are you currently in the company?

_____

What role did you have in previous migration projects?

_____

How much experience do you have in operating and developing microservices?

	1	2	3	4	5	
No experience	☐	☐	☐	☐	☐	Very much experience

How competent do you consider yourself in operating and developing microservices?

	1	2	3	4	5	
No competence	☐	☐	☐	☐	☐	Very much competence

How much experience do you have in microservice migrations?

	1	2	3	4	5	
No experience	☐	☐	☐	☐	☐	Very much experience

How competent do you consider yourself in microservice migrations?

	1	2	3	4	5	
No competence	☐	☐	☐	☐	☐	Very much competence

A.3 Declaration of consent

adesso

Declaration of consent for the collection and processing of personal data for the preparation of a master thesis

A. Subject of the research and basis of the declaration of consent

1. **Subject of the research**:

   Experiences of continuous microservice migrations

2. **Description of the research:**

   The research with the topic "How To Strangle Systematically: An Approach and Case Study for the Continous Evolution of Monoliths to Microservices" is carried out in the context of a master thesis with the goal to provide a systematic approach based on influencing factors and decisions for a continuous microservice migration by practically applying the strangler pattern.

3. **Research management**:

   The study is conducted by Valentin Seifermann, who is a member of the master's program in Software Engineering at the University of Stuttgart. During the study, he is responsible for conducting the interviews as well as managing the entire study.

4. **Type of personal data of the interviewee**

   During the interviews, personal data is expected to be requested, in particular: Name, age, gender, professional background and recordings (especially sound recordings) will be requested. In addition, there is a possibility that during the interview process, one or more of these pieces of information will be addressed. Based on your consent, the interviews will be recorded using a digital tape recorder.

5. **Interviews**

   The duration of the interviews is approximately 45 - 60 minutes. They are conducted via Microsoft Teams and recorded using its functionality and the Open Broadcast Software (OBS).

adesso

B. Declaration of consent and information on the collection of personal data

1. Declaration of consent

I hereby consent to the processing of my personal data collected in the course of the research project described in A. above, to Valentin Seifermann in the form of original recordings of the interview and their transcript, for the purposes of the evaluation in accordance with section 2. If I specify or have specified special categories of personal data, these are covered by the declaration of consent.

Your consent is voluntary. You may refuse consent without incurring any disadvantages.

You can revoke your consent at any time to Valentin Seifermann with the consequence that the processing of your personal data, in accordance with your declaration of revocation, will become inadmissible for the future. However, this does not affect the lawfulness of the processing carried out based on the consent until the revocation.

Relevant definitions of the data protection terms used are contained in the appendix **Explanation of terms** in german.

2. Purpose of data processing and goal for the research

According to 2), the purpose of the data processing in the broader sense is to gain an understanding on how participants of migrations projects at Adesso SE have experienced the migration process including influencing factors and decisions that are made based on the factors.

In this context, personal data will be collected prior to the investigation in the form of a short questionnaire in accordance with 5). In addition, it is possible that personal data, such as first and last name or company affiliation as well as other information, are also addressed during the investigation and remain on the tape recording.

This tape recording is necessary for the investigation to be able to ensure that the conversation proceeds smoothly as well as to avoid any loss of information. Furthermore, it will be used to create a proper interview transcript according to the quality of the research, which serves as a basis for the analysis of the data.

3. Contact details of the researcher

Valentin Seifermann: valentin.seifermann@adesso.de

adesso

4. Legal basis

Valentin Seifermann processes the personal data collected from you based on your consent according to Art. 6 para. 1 p. 1 DSGVO. If special categories of personal data are involved, Valentin Seifermann processes the personal data collected from you based on your consent according to Art. 9 para. 2 DSGVO.

5. Duration for which the personal data is stored

The personal data collected by means of digital tape recording will be replaced by synonyms during transcription and anonymized to this extent. These collected personal data will be stored until the end of the study on 31.10.2021, but for a maximum of 2 years.

6. Your rights

In context of the statutory provisions, you are generally entitled to receive from Valentin Seifermann:

- Confirmation as to whether personal data related to you is being processed by Valentin Seifermann,
- Information about this data and the circumstances of the processing,
- Correction, as far as the data is incorrect,
- Deletion, as far as there is no justification for the processing and there is no longer an obligation to retain the data,
- Transfer of your personal data – as far as you have provided it - to you or a third party in a structured and machine-readable format
- Restriction of processing in special cases determined by law

In addition, you have the right to revoke your consent at any time towards Valentin Seifermann, with the consequence that the processing of your personal data, in accordance with your declaration of revocation, becomes inadmissible by him for the future. However, this does not affect the lawfulness of the processing carried out based on the consent until the revocation.

adesso

7. No automated decision making (including profiling)

Processing of your personal data for the purpose of automated decision-making (including profiling) according to Art. 22 para. 1 and para. 4 DSGVO does not take place.

_____

Surname and firstname

_____ _____

Place and date Signature

adesso

Appendix: Explanation of terms

- „Personenbezogene Daten“ (refer to as „personal data“) sind gemäß Art. 4 Nr. 1 DSGVO alle Informationen, die sich auf eine identifizierte oder identifizierbare natürliche Person (im Folgenden „betroffene Person“) beziehen. Als identifizierbar wird eine natürliche Person angesehen, die direkt oder indirekt, insbesondere mittels Zuordnung zu einer Kennung wie einem Namen, zu einer Kennnummer, zu Standortdaten, zu einer Online-Kennung oder zu einem oder mehreren besonderen Merkmalen identifiziert werden kann, die Ausdruck der physischen, physiologischen, genetischen, psychischen, wirtschaftlichen, kulturellen oder sozialen Identität dieser natürlichen Person sind. Das kann z.B. die Angabe sein, wo eine Person versichert ist, wohnt oder wie viel Geld er oder sie verdient. Auf die Nennung des Namens kommt es dabei nicht an. Es genügt, dass man herausfinden kann, um welche Person es sich handelt.
- „Besondere Kategorien“ (refer to as „special categories“) personenbezogener Daten sind gemäß Art. 9 Abs. 1 DSGVO Daten, aus denen die rassische und ethnische Herkunft, politische Meinungen, religiöse oder weltanschauliche Überzeugungen oder die Gewerkschaftszugehörigkeit hervorgehen, sowie die Verarbeitung von genetischen Daten, biometrischen Daten zur eindeutigen Identifizierung einer natürlichen Person, Gesundheitsdaten oder Daten zum Sexualleben oder der sexuellen Orientierung einer natürlichen Person.
- „Gesundheitsdaten“ (refer to as „health data“) sind gemäß Art. 4 Nr. 15 DSGVO personenbezogene Daten, die sich auf die körperliche oder geistige Gesundheit einer natürlichen Person, einschließlich der Erbringung von Gesundheitsdienstleistungen, beziehen und aus denen Informationen über deren Gesundheitszustand hervorgehen.
- „Verarbeitung“ (refer to as „processing“) ist gemäß Art. 4 Nr. 2 DSGVO jeder mit oder ohne Hilfe automatisierter Verfahren ausgeführten Vorgang oder jede solche Vorgangsreihe im Zusammenhang mit personenbezogenen Daten wie das Erheben, das Erfassen, die Organisation, das Ordnen, die Speicherung, die Anpassung oder Veränderung, das Auslesen, das Abfragen, die Verwendung, die Offenlegung durch Übermittlung, Verbreitung oder eine andere Form der Bereitstellung, den Abgleich oder die Verknüpfung, die Einschränkung, das Löschen oder die Vernichtung.

A.4 Interview analysis

A.4.1 List of codes

MAXQDA 2020 17.10.21

List of codes

List of codes	Frequency
Codesystem	314
Migration Drivers	0
Independent development	6
Scalability	5
Performance issues	7
Maintainability	2
Comprehensive understanding required	4
Risk of breaking changes	2
Increased costs for further development	2
Decisions	0
Deciding to use feature toggles	1
Deciding which data consistency to use	1
Deciding which appropriate technology to use	1
Deciding to keep development process with minor modifications	2
Deciding to start with pain points	4
Deciding to refactor and modularize before decomposing	1
Deciding to consult external parties with experience	1
Deciding to follow best practices	1
Deciding based on outcome of prototype	3
Deciding who will develop and operate the new microservice	4
Deciding where to implement new requirements based on dev cost	6
Choosing Monolith if too expensive	3
Deciding to extract service candidate with new coherent feature	1
Deciding to put new standalone functionalities in microservices	2
Deciding to conduct migration based on identified drivers	3
Deciding to conduct migration in subteam	1
Deciding to conduct Proof-of-Concept	2
Steps	0
Identifying other migration options	4
Creating priorization	2
Measuring throughput	1

MAXQDA 2020 17.10.21

Extract peristance layer	1
Modifying the monolith	11
Create interfaces	2
Modularizing and refactoring	5
Resolving circular dependencies	3
Create Definition-of-Done for microservice criteria	1
Reflecting over the migration processes	2
Defining thresholds for testing	1
Conducting tests	2
Restructuring Organisation	6
Qualitative data validation after extracting service	4
Elaborate a concept for the migration	6
Establish deployment strategie	2
Analysing documents	1
Analysing legacy system	20
Analysing dependencies	2
Applying Domain-Driven-Design	5
Analysing source code	4
Conducting architectural tests	3
Setting up infrastructure	9
Setup API gateways	3
Changing CI/CD pipelines	3
Implementing prototypes	6
Checking performance after migration	2
Checking business consistency after migration	2
Rating requirements based on development costs	5
Developing concept to integrate extracted services	3
Integrating microservice using proxy mechanism	1
Gaining experiences with technologies	3
Sketching use case	1
Conducting PoC	1
Indentifying service candidates	4
Event-Storming	1

MAXQDA 2020 17.10.21

Factors	0
Risk managment	1
Monitoring	1
Asynchronous processing	2
Identifcation of service candidates	1
Priorization of service candidates	2
Concept phase	3
Database migration	2
Transfering code	3
Code ownership	2
Migrating in parallel	8
Consider migration as re-design	1
Budget	8
Preparatory work on the monolith	2
Missing or obsolet documentation	1
Communication with stakeholder	5
Project duration	4
Experience with technologies	10
Prevent team silos	1
Functionality nested deep inside the monolith	2
Integration with legacy system	3
Considering side effects	1
Bounded to technologies provided by customer	6
Costs and effort of migrating are difficult to calculate	1
Establishing deployment strategies	2
Starting with big pain points	2
Resolving dependencies of the micorservice to the monlith	3
Adherence to cloud regularities	4
Data exchange between microservices	1
Creating the capacity for migration	3
Increased coordination and evaluation	3
Advancing in small steps	2
Setup infrastructure	9
Changing CI/CD pipelines	4

Setup API gateways	2
Exchanging experience	5
New or changing requirements	8
Release management	3
Changing roles within team	2
Changing team structure	6
Divide responsibility difficult	1
Team too big	1
Vertical team structure	1
Domain knowledge of legacy system	3
Knowledge of new achitectural style	3
Service cut	6
Commitment of stakeholder	2
Customer expectations	5
Convincing the customer	3
Significance of migration	6

A.4.2 Categories

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature