

Institute of Software Engineering
Empirical Software Engineering Group

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master Thesis

Migrating Monolithic Architectures to Microservices: A Study on Software Quality Attributes

Daniel Koch

Course of Study: Softwaretechnik
Examiner: Prof. Dr. Stefan Wagner
Supervisor: Jonas Fritsch, M.Sc.

Commenced: June 20, 2022
Completed: December 20, 2022

Abstract

There are many drivers for migrating from a monolith to a microservice architecture, such as high scalability or improved maintainability. To do this, however, several factors must be taken into account in the migration process, including quality attributes. Since the migration towards a microservice architecture is not an easy task, defined quality goals can assist in selecting a suitable migration approach and subsequently making appropriate architectural design decisions. The goal of this thesis is to investigate how quality attributes can be incorporated into the migration process, so that practitioners and software architects can be supported in this regard. Likewise, it is examined which role they play in the migration process. To this end, a literature review was conducted to first identify the quality attributes that are relevant for a microservice architecture. Then, quality attributes were assigned to the migration approaches that optimize them towards the target architecture. Similarly, quality attributes were assigned to architectural patterns and best practices. Based on the previous results collected, a quality model was created, also addressing interdependencies and trade-offs among them. In doing so, the quality model was intended to act as a guide by facilitating the selection of appropriate techniques and architectural choices based on defined quality goals. The developed quality model was subsequently integrated into a tool, which was designed to guide practitioners through the migration process. To investigate the usability of the tool with respect to the quality model, an evaluation in the form of a survey was conducted with four practitioners from industry. The result of the evaluation shows that the integrated quality model can support the migration process based on defined quality goals in practice and that the extension of the tool has a high usability.

Contents

1	Introduction	12
2	Background and Related Work	14
2.1	Monolith and Microservices Comparison	14
2.2	Migration from Monolith to Microservices	15
2.3	Related Work	16
2.4	Architecture Refactoring Helper Tool	17
3	Methodology	19
3.1	Execution of Literature Review	19
3.2	Research Objectives	20
3.3	Quality Model and Implementation	26
4	Quality Attributes in Migration to Microservices	27
4.1	Optimization of Quality Attributes as a Driver	27
4.1.1	Role in Migration	27
4.1.2	Quality Attributes for Microservices	29
4.1.3	Correlation System Properties and Quality Attributes	37
4.1.4	Quality Trade-Offs	39
4.2	Migration Approaches Overview	40
4.3	Microservices Design Patterns and Best Practices Overview	41
4.4	Quality Attributes in Microservices Design	45
4.4.1	From Quality Attributes to Migration Approaches	47
4.4.2	From Quality Goals to Design Patterns and Best Practices	49
4.4.3	Patterns Trade-Offs	55
4.5	Quality Model	56
4.5.1	Design and Overview	57
4.5.2	Model in Detail	58
4.5.3	Quality Model acting as Guide	60
5	Implementation	62
5.1	Prerequisites	62
5.2	Requirements	63
5.2.1	Backend Requirements	63
5.2.2	Frontend Requirements	65
5.3	Development Process	66
5.4	Backend Design	67
5.5	Frontend Design	73

6	Evaluation of Usability	79
6.1	Survey Design	79
6.2	Survey Results	80
6.3	Discussion of Survey Results	82
7	Discussion	85
8	Threats to Validity	88
9	Conclusion and Future Work	89
	Bibliography	91
A	Survey Material	103

List of Figures

2.1	Architectural Refactoring Framework Overview [64].	17
3.1	Selection Procedure from Stage 1.1.	22
3.2	Selection Procedure from Stage 1.2.	22
4.1	ATAM Quality Tree with Scenarios based on Starke [136].	29
4.2	Number of Occurrences of the Quality Attributes in the 13 found Papers.	31
4.3	Overview of the identified Quality Attributes and their Subattributes.	31
4.4	Quality Attributes and System Properties in the Migration Process to Microservice Architecture.	46
4.5	Quality Model Overview.	57
4.6	Quality Model on Migration Strategy Level.	58
4.7	Quality Model on Architectural Design Level.	59
5.1	User Interface of Scenario Component.	74
5.2	Overview of Results of Recommendations for Migration Approaches.	76
5.3	Configuration Overview in Sidepanel.	78

List of Tables

3.1	Methodology Definition Protocol of Stage 1.1.	21
3.2	Methodology Definition Protocol of Stage 1.2.	23
3.3	Methodology Definition Protocol of Stage 3.1.	25
4.1	Influence of System Properties on Quality Attributes.	38
4.2	Classification of Migration Approaches from Monolith to Microservices.	41
4.3	Design Patterns Overview and Categorization.	44
4.4	Quality Attributes optimized by Migration Approaches.	49
4.5	System Properties considered by Migration Approaches.	50
4.6	Quality Attributes optimized by Architectural Patterns.	52
4.7	Quality Attributes optimized by Best Practices.	54
4.8	Architectural Patterns Trade-offs Overview.	56

List of Listings

5.1	Configuration of Relationship of Quality Attributes and Subattributes (C#).	67
5.2	Configuration of Quality Attributes and System Properties (C#).	68
5.3	Data Model of a Scenario (C#).	69
5.4	Get all Qualities of a Scenario via Controller (C#).	69
5.5	Get all Qualities of a Scenario via Service Method (C#).	70
5.6	Pattern Example for JSON Data Seeding.	71
5.7	Request Scenarios for Scenario Component (Typescript).	73
5.8	Set Scenario-based Mode by Routing Parameter (Typescript).	75
5.9	Quality Tendency Table Cell of Recommendation Results (HTML).	77

List of Algorithms

5.1	Weighted Quality and System Property Score Calculation.	72
-----	---	----

Acronyms

- API** Application Programming Interface. 43
- ATAM** Architecture Tradeoff Analysis Method. 12
- CD** Continuous Delivery. 43
- CI** Continuous Integration. 14
- CRUD** Create, Read, Update and Delete. 64
- DBMS** Database Management System. 62
- DMC** Dynamic Microservice Composition. 40
- EF Core** Entity Framework Core. 62
- HTML** Hypertext Markup Language. 62
- HTTP** Hypertext Transfer Protocol. 73
- ISA** Independent System Architecture. 43
- JSON** JavaScript Object Notation. 62
- MDA** Meta-Data aided. 40
- NPM** Node Package Manager. 63
- ORM** Object-relational mapping. 62
- QA** Quality Attribute. 12, 89
- REST** Representational State Transfer. 62
- SCA** Static Code Analysis aided. 40
- SCSS** Sassy Cascading Style Sheets. 62
- SE** Software Engineering. 19
- SP** System Property. 30, 89
- SRP** Single Responsibility Principle. 14
- WDA** Workload-Data aided. 40

1 Introduction

The migration from a monolithic to a microservice architecture is often undertaken by companies for a number of justifiable reasons. These include the capacity for greater scalability, improved maintainability, and delegating responsibilities in a more efficient manner [139]. Fundamentally, the goal of companies is to slow down the ever-increasing complexity that occurs as monolithic systems grow through migration [139]. Thus, many are moving from monolithic systems to microservices to take advantage of cloud technologies and enable a more agile development process. There are several ways to accomplish this, including incrementally restructuring the existing code, rebuilding the application by using modern technologies, or starting from scratch with an entirely new system [15]. The appropriate approach for migration depends on the resources available, the technologies used, and the timeframe for the migration. There has been a lot of progress in developing strategies, approaches, and techniques to standardize, partially automate, and control the migration process through research in this area [1, 58, 112].

As explained by Capuano and Muccini [30], when migrating to a microservice architecture, several parameters must be considered, including Quality Attributes (QAs). Likewise, the authors state that “quality-driven migration to microservices is a growing trend in the research community” [30]. To this end, the QAs that are influenced by a microservice architecture have become increasingly important, not only in industry but also in research [93]. This is due to the fact that the influence of a microservice based architecture on these attributes is mostly perceived as positive or at least they are not negatively affected by it [21]. According to Li et al. [93], the QAs in the process play a major role in the selection of suitable approaches and techniques for migration. This is because, when deciding on an approach or technique, it is important to consider the QAs that are to be optimized by the chosen method. To this end, the quality goals must be determined as part of an architecture assessment using suitable methods, for instance the Architecture Tradeoff Analysis Method (ATAM) according to Kazman et al. [84], even before the migration process.

Moreover, these QAs are crucial during the subsequent architectural design phase. This is because many architectural patterns and best practices are associated with certain QAs that either enhance or influence them [94, 144, 145]. During a migration, this can help architects by considering quality aspects when making decisions related to architecture decomposition and design.

The overarching research objective describes, how QAs can be incorporated into the migration process so that they can assist architects or practitioners in choosing migration approaches and subsequently in making architectural design decisions. Inferring from this the purpose of this thesis is to suggest a new approach for directing migrations based on quality aspects. In order to achieve this, the following research questions will be addressed:

- *RQ 1*: How can different quality goals be incorporated in the microservices migration process to guide the selection of decomposition approaches?

-
- *RQ 2*: How can these goals subsequently support the choice of architectural design patterns and best practices?
 - *RQ 3*: How can the ‘Architecture Refactoring Helper’ tool [65] be extended by a quality model that guides practitioners in a migration scenario?

In order to address the first two research questions of the thesis, a literature review will be conducted in different stages. For each stage either a Rapid Review according to Cartaxo et al. [32], a manual search or a mapping of the results is performed. The findings of this literature review result in a quality model that considers not only the mappings of the attributes to the techniques, patterns, and best practices, but also interdependencies among them. One goal of the work is that the quality model can act as a guide that can provide recommendations to practitioners at each stage of the migration process based on QAs. To address the third research question, the tool ‘Architecture Refactoring Helper’, which was developed in the context of a master thesis by Haller [64], will be extended by the quality model. In order to subsequently investigate the usability of the extension, an evaluation is carried out with four practitioners from the industry. The evaluation consists of conducting a survey with the goal of obtaining qualitative data. This is done by asking the participants of the survey to perform tasks within the extension developed in the context of this work. Afterwards statements about usability are given, which the participants can agree or disagree with and furthermore give additional reasons and suggestions.

For this purpose, the work is structured as follows: First, **Chapter 2** will explain the background and the most important aspects needed for further understanding. Furthermore, related work to the research goals of this work are presented. Then, **Chapter 3** will address how the methodology for this thesis was designed in terms of the literature review conducted, the quality model, and the implementation. Using the literature review, **Chapter 4** examines the role of QAs in the migration from a monolith to a microservice architecture. In addition, an overview of relevant migration approaches and architecture decisions is given here. Subsequently, the QAs relevant for microservices are assigned to the individual approaches, architectural patterns, and best practices so that they can support finding suitable techniques for migration based on a selection of QAs and then making appropriate architecture decisions. Furthermore, trade-offs and relationships between the quality aspects and possible drawbacks of architecture decisions are addressed. In addition, the resulting quality model is explained in detail. **Chapter 5** describes the steps of the implementation of the extension carried out in the context of this work. In order to examine the usability of the implemented extension, an evaluation is then carried out in **Chapter 6** with the help of practitioners. For this purpose, a qualitative survey is conducted, and the results are analyzed and subsequently discussed. Following this, **Chapter 7** addresses the research questions and discusses the results of the work in relation to them. After the threats to validity are pointed out in **Chapter 8**, the thesis is closed in **Chapter 9** with a conclusion and possible future work.

2 Background and Related Work

In order to put the background for this work into context, the most important characteristics of monolith architectures and microservice architectures are discussed and compared first. Furthermore, the migration process from monoliths to microservices is covered. For this purpose, the reference to QAs, which represent a main aspect, is also established. After that, the related works connected to the research goals of this thesis are described.

2.1 Monolith and Microservices Comparison

A monolithic architecture is a single unit of software in which all logic and tasks of the application are contained in one process or operation [52, 124]. At best, this is a simple and lightweight application [82]. A programming language or framework for the entire software is often used for this purpose [124]. These characteristics of the architecture result, among other factors, in the advantages of easy testability and implementation [79, 124]. It is also easy to scale and deploy due to its simplicity, provided the size of the codebase is kept small [20, 79]. A load balancer is suitable for carrying out such scaling for smaller applications [116]. However, when the applications become too large or complex to maintain, some problems come along [79, 82]. Because then the tight coupling, resource conflicts when scaling and the more difficult continuous software delivery can become a major disadvantage. This also builds on the fact that a monolithic application has to be completely deployed and built every time [52, 79]. However, if the advantages of avoiding the technical and organizational challenges of microservices are no longer given, or if modularity is required, a microservice architecture should be considered in the long term [79, 82].

The architecture with microservices tries to solve the challenges of a monolith architecture by dividing the application into smaller services [82]. Each service is responsible for a single task - it works according to the Single Responsibility Principle (SRP) [105]. Because each service runs its own process, they are (automatically) deployable independently of one another [52]. The communication between and with the services can be realized via messaging and lightweight mechanisms [52]. The advantages of the architecture here include the characteristic of loose coupling due to the natural influence of the distribution of services [79, 124]. As a result, the boundaries are clear and changes to one service can be made independently of one another without affecting the other service. The characteristics mentioned of such an architecture can also enable Continuous Integration (CI) and more frequented deployment [124]. In addition, due to the loose coupling, scalability can be better achieved, since only one part and not the entire application has to be scaled [79]. The same reason enables the individual services to be independent of languages and technologies [124]. Moreover, it must be noted that the microservice architecture also has disadvantages that need attention, and these shortcomings are mainly inherent in the nature of distributed architecture [20]. Because in this context the coordination and decomposition of the individual services are much more complex than coordinating a single application [20]. In addition,

the communication and design of it represents a major challenge [79]. Furthermore, there are more points of error due to the distributed architecture, which in turn has the advantage that errors in one service do not affect the other aspects [79]. The services, which are ideally isolated as far as possible, also make the design of the data management more difficult and imply that different database systems may be used, that have to be coordinated [20]. The use of a microservice architecture can therefore have many advantages. However, the migration from a monolithic application to one with microservices must be well conceived and sensible due to the effort involved in coordination and design.

2.2 Migration from Monolith to Microservices

In recent years, various companies have migrated from monolithic architecture to microservice architecture [139]. The reasons for this can be derived from the advantages of microservice architectures from Section 2.1 and can be explained by the results of the survey by Taibi et al. [139], which comes to the conclusion of the same aspects. Scalability, maintainability, independent teams for parts of the application and easier and more frequent deployment are listed as motivation aspect for migration.

Since each application has a different architecture and is designed individually, a suitable selection for the right approach to the migration process is difficult, since this also changes the requirements for the target architecture [30, 68]. Many factors should be considered here, including quality aspects [30].

In order to perform a migration from monoliths to a microservice architecture there are fundamentally three different methods for the entire process [15]:

1. **Re-Factor** - The existing code base is refactored step by step without changing the functionality of the application.
2. **Re-Build** - The entire application is re-architected by using new technologies suitable for the requirements.
3. **New Application** - An entire new system is written from scratch in order to implement the microservice architecture with the desired functions.

Furthermore, Bajaj et al. [15] state that there are many approaches in the literature that describe how this migration can be performed, but there is no solution that is suitable for all systems. According to the authors, this is because there are two basic groups into which the migration methods can be classified. On the one hand, there are the greenfield approaches, which carry out a development from scratch. This includes the *New Application* method. On the other hand, there are the methods of the brownfield approaches, which revise an existing application by refactoring it or redesigning the entire architecture. Specifically in the brownfield area, a distinction can be made here according to Bajaj et al. [15] again between two types. There are static approaches, which analyze only static elements for the migration, and dynamic approaches, which examine the behavior of the application during runtime.

In summary, there are fundamentally different approaches that can lead to the desired goal of migration to a microservice architecture, depending on the requirements and prerequisites.

2.3 Related Work

To carry out the related works for the presented research questions and scientific research with expedient results for QAs in the context of microservice architecture are dealt with first. After that, already existing quality models are considered, which can create a reference to microservices and are possibly composed of different QAs.

In the research area of QAs in microservices, a systematic literature review has already been conducted by Li et al. [93], which examines reported evidence for these from studies. However, only the most concerned QAs are considered, such as scalability or maintainability. Further efforts on this have been made by Cojocararu et al. [37], where a literature survey was conducted, which focuses on identifying a set of quality assessment criteria for microservices, which are minimally required. In this regard it must be noted that the scope of the work was exclusively in the area of semi-automated migration tools and techniques.

A more in-depth examination of QAs is provided by the work of Valdivia et al. [144]. The challenge of creating a mapping between architectural patterns and QAs is addressed by the authors. To do this, a systematic review is conducted, which describes the architectural patterns, establishes associations with QAs, and identifies needed trade-offs in pattern selection. The mentioned trade-offs are also considered by Vale et al. [146]. In this work, one of the issues addressed is, how QAs are influenced when applying different microservice patterns. This work will also help to understand how QAs are influenced by architectural patterns and thus can assist in guiding the decision making. The first approaches to setting up and classifying QAs for microservices therefore already exist. However, there is a lack of reference to the migration process when refactoring the architecture from monoliths to one with microservices, or only a coarse superclass of QAs is used.

In the area of quality models for microservices or their architecture, the available research is limited. Nevertheless, there is for example an approach by Yu et al. [151] which provides a proposal for such a model. For this purpose, characteristics (or attributes) of quality and the theoretical validation of these characteristics are reflected. However, the cohesion between the mentioned QAs is less treated, wherein their suggestion would be to incorporate further work. In their work, Lichtenthäler and Wirtz [94] have developed a quality model for cloud-native applications whereby they extract the quality aspects on which the applications have an impact. Likewise, the QAs are set in relation to each other. Since, according to Lichtenthäler and Wirtz [94], microservices are only a subset of cloud-native applications, this model is relevant. For this reason, it also includes many factors and relationships and is therefore complex in structure. While Waseem et al. [148] have not developed a quality model, they have created a decision-making model about how to decompose monolithic architecture into a microservice architecture. The model considers conditions such as the fulfillment or non-fulfillment of certain QAs or textual descriptions, and uses this information to select the most appropriate strategies and patterns for the decomposition process towards microservices.

One of the main aspects of this thesis will be to close the gaps identified and examining the relevant factors from the individual areas of the objectives mentioned and composing them in a meaningful way into a proposal for a way of guiding the migration process based on quality criteria.

2.4 Architecture Refactoring Helper Tool

For the implementation of an extension of the ‘Architecture Refactoring Helper’ tool [65], a basic understanding of the tool and the underlying framework is important. In the context of a master thesis by Haller [64], a tool has been developed in cooperation with the designer of the ‘Architectural Refactoring Framework’, which implements this framework prototypically. In general, the ‘Architectural Refactoring Framework’ is a research project of the Empirical Software Engineering (ESE) group of the Institute for Software Engineering (ISTE) at the University of Stuttgart, which is driven by ongoing research activities. The basic idea of the research project is that support for refactoring a monolithic architecture towards a microservice architecture is given in the form of guidance. Above all, software architects and developers should be helped to be able to proceed in a structured manner in the process. The basic structure of the ‘Architectural Refactoring Framework’ can be illustrated in Figure 2.1.

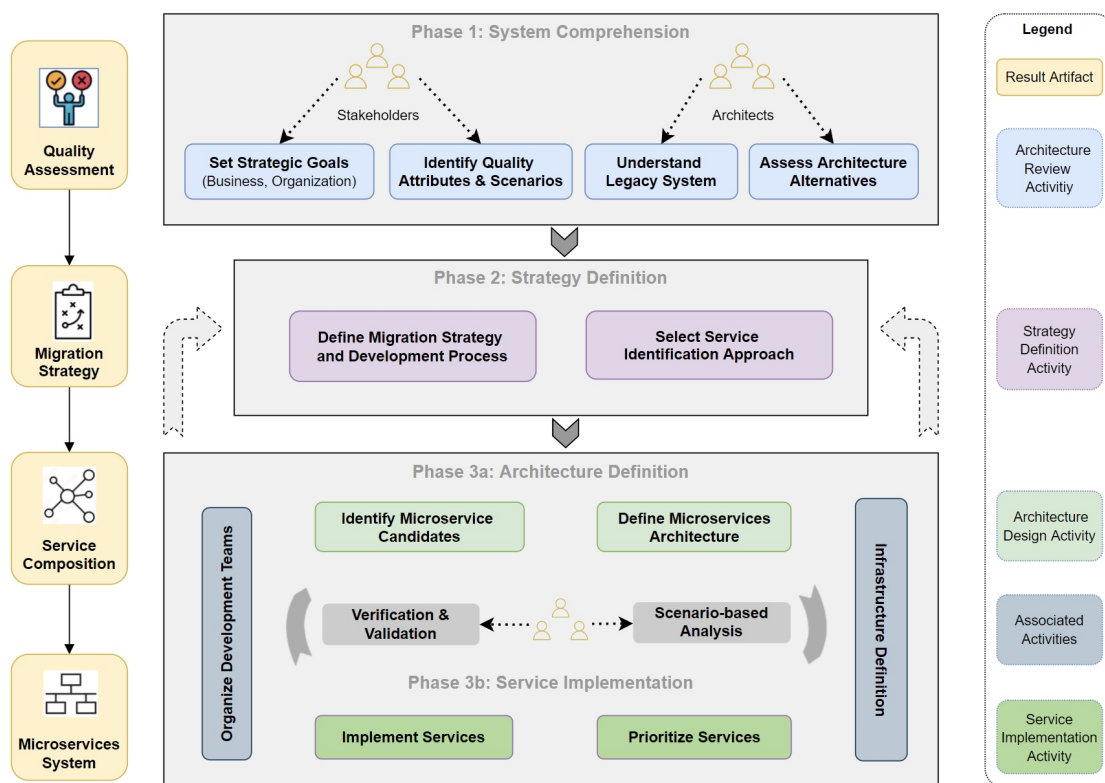


Figure 2.1: Architectural Refactoring Framework Overview [64].

Here it becomes clear that the framework is fundamentally divided into three phases. First there is the System Comprehension phase, which is to serve for the comprehension of the system, for which a migration is to be accomplished. In this phase, the stakeholders consider organizational and quality aspects, among other things, and define possible goals and drivers for the target architecture. The assessment can then be used to provide an overview of whether a migration makes sense or whether a different architecture might be better suited to the requirements. In the prototype by Haller [64], no functionality was initially implemented for this phase of the framework, but the intended functionality was presented in the form of user interface sketches.

The second phase, the Strategy Definition, deals with the migration strategy in case of a potential migration and the service identification approach. For this, Haller [64] has already made efforts, so that recommendations for migration approaches and service identification approaches can be generated and displayed through configurations and presets of various input attributes. Likewise, the individual approaches can be edited, exported and imported, so that adaptations in future research work are facilitated. Also an overview of all approaches available in the database can be given by means of the implementation of this phase.

Phase three deals with the architecture definition, which includes how the architecture can be defined in detail and how the microservice candidates are identified. The further process of implementing the services is then outside the scope of the framework. In the implementation of the ‘Architecture Refactoring Helper’ tool [65], no implementation was realized here.

The implementation as an essential part of this work will start in phase one, where the quality goals for the target architecture can be defined. In phase two, the functionality will be extended by a resulting quality model and in phase three, patterns and best practices for the defined quality goals will be recommended.

3 Methodology

In general, the methodology of the thesis can be divided into three major steps, which are examined one after the other. In the first step, a literature review is conducted. Here, the role of QAs in the migration from monoliths to microservices is examined. In addition, migration approaches are considered collectively and related to QAs identified in advance. The same mapping of QAs is applied to existing design patterns/best practices for microservice architectures. Likewise, efforts are made to relate the target architecture patterns (design patterns) to the migration approaches. Afterwards, a quality model for migration is developed using the results from the literature review. For this purpose, the QAs are related to the different approaches to migration and the design patterns that occur in them or are considered by these techniques and patterns.

In the third step, the developed quality model will be implemented into the already existing web-based application 'Architecture Refactoring Helper'. Therefore, established software engineering standards are used for the prototypical implementation. To assess the usability afterwards, a survey will be conducted with a number of software experts with experience in microservice migrations.

3.1 Execution of Literature Review

In order to carry out the literature review, the individual aspects are first divided into different 'stages'. This means that the individual parts can be viewed independently of one another and links can be made from the results. This is necessary because the creation of relations and connections between the separately considered aspects is a crucial workload for this thesis and only in this way the research questions can be answered. A distinction must be made between three types of stages. First there are stages that require a separate literature review. Second there are stages that create a mapping between the results of previous stages. And third for the overviews, existing literature reviews or mapping studies may be used, if available.

To enable a structured procedure for the stages for which a literature review is required, the guidelines from Cartaxo et al. [32] for a Rapid Review in the field of software engineering are used. The choice of this compared to other guidelines (e.g. systematic literature reviews) is justified by the possibilities of time and scope given in the context of this work. Because with a Rapid Review, in comparison, less work and, above all, time reasons can be a motive [33]. In contrast to systematic literature reviews, individual steps can be simplified or left out, such as the review process or the quality appraisal of the resulting works [33], since this type of review is primarily suitable for a fast but extensive knowledge transfer. Due to the simplification, the question of validity in comparison to systematic literature reviews arises. Cartaxo et al. [31] took up this question and carried out a study of the research community's viewpoints on Rapid Reviews. The result was that there are some concerns because it is a new concept, at least in the Software Engineering (SE) area. However, based on the results, they were able to come to the conclusion that both systematic literature reviews

and Rapid Reviews can be carried out very poorly or very well. There are also different studies in the field of medicine that come to the conclusion that although there are many differences between systematic literature reviews and Rapid Reviews, the results achieved between the two are usually very similar [32]. And to take this aspect into consideration, a structured Rapid Review is attempted using the guidelines. In the following, reference is made to the guidelines according to Cartaxo et al. [32], which are then answered individually for the appropriate stages:

1. **Search strategy**
2. **Selection procedure**
3. **Quality Appraisal**
4. **Extraction Procedure**
5. **Synthesizing the data**

For each stage of the literature review, it is indicated which **search strategy** was used, i.e. it is stated which databases were searched using which search strings (respectively adapted to database). Furthermore, additional search strategies such as manual search or complementary Snowballing are listed, if applicable. For the **selection procedure**, possible inclusion or exclusion criteria for the papers are listed, for example, the date of publication or the type of paper. Details of the review process are also given. For the **quality appraisal**, it is explained how and whether a quality check is executed for the results of the search. In the case of a Rapid Review according to the guideline, this can even be omitted entirely. For the **extraction procedure** of the data, different possibilities are offered, which are indicated in each case. For example, an extraction form can be used, which is then explained or presented separately for each relevant stage. For the **synthesis procedure**, the presentation and synthesis of the findings is described in more detail. Should a mapping from the results follow in the next stage, it is advisable to already design the extraction form in a target-oriented manner, so that the relevant aspects can be directly linked to the results to be mapped.

3.2 Research Objectives

In the following, the stages are listed in ascending order with the associated research objective. This means that the stages are addressed sequentially, with the sub-stages of each stage being based on the results of the previous sub-stage.

Stage 1.1: Evidence on the role of QAs in the migration from monolithic to microservice architectures

To conduct a research on the role of QAs, the aspects listed in in Table 3.1 were followed to perform the literature review. In this regard, after the basic information, the different steps of the guidelines for the Rapid Review according to the guidelines of Cartaxo et al. [32] are followed.

For the selection procedure (illustrated in Figure 3.1), the search string given in Table 3.1 and the date range in the given databases initially yielded 80 results. In this step, duplicates in the results were also removed. After the first round, unsuitable papers could then be sorted out on

Basic Information	
<i>Type:</i>	Literature Review on the role of QAs in the migration process from monolithic to microservice architecture with regard to Section 4.1 and Section 4.1.1
<i>Method:</i>	Rapid Review according to the guidelines from Cartaxo et al. [32]
Search Strategy	
<i>Search String:</i>	((“Title”: quality) AND (attribute OR characteristic OR aspect OR goal) AND (role OR impact OR influence OR improv* OR optimiz*) AND (microservice OR micro-service) AND (migrat* OR decompos* OR refactor OR partition OR transform*))
<i>Database:</i>	Google Scholar, IEEE Xplore
<i>Complementary Method:</i>	Snowballing
Selection Procedure	
<i>Inclusion Criteria:</i>	In the explicit context of QAs in microservices; Reference to the migration process is present
<i>Exclusion Criteria:</i>	Too specific for individual use cases; Access to the entire content of the paper not available
<i>Date Range:</i>	2011-2022
<i>Publication Type:</i>	The search result may not be a thesis
<i>Language:</i>	English or German
Quality Appraisal	
<i>Quality Check:</i>	-
Extraction Procedure	
<i>Method:</i>	Collection of notes and search for commonalities
<i>Collection:</i>	Annotated in keywords or in the form of complete quotations (explicitly marked)
Synthesis Procedure	
<i>Type:</i>	Narrative

Table 3.1: Methodology Definition Protocol of Stage 1.1.

the basis of the titles already according to the exclusion criteria. Thereafter, the exclusion criteria were used to continue sorting out by reading the abstract. In the last round, unsuitable papers were then filtered out after the full paper reading and possible papers were added to the results manually by Snowballing. This results in a total of eight relevant papers, of which five papers are from the conventional methodology and two papers and one online survey are from the Snowballing methodology.

Beside the findings about the role of QAs in the migration process, a method could be found additionally via manual research, which can help for the architecture assessment in advance. This is especially important for the later implementation, since this assessment method is an essential part of the tool extension.

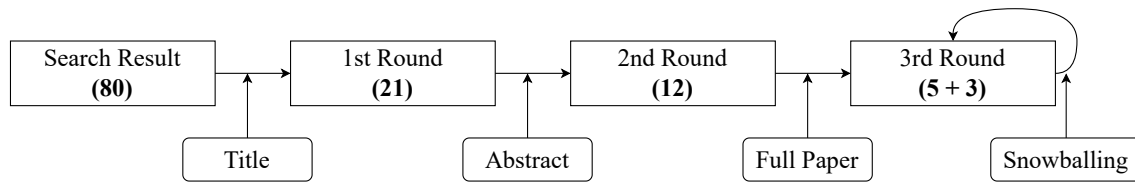


Figure 3.1: Selection Procedure from Stage 1.1.

Stage 1.2: Identification of Quality Attributes for microservices architectures

In order to investigate the relevant QAs for microservice architectures, research was first conducted using the information provided in Table 3.2. This is primarily used to identify QAs by forming the intersection of the attributes contained in the findings. In order to cover as many attributes as possible, Snowballing is also applied in the relevant works, provided that the reference also picks up QAs. Thereupon, the characteristics of the individual QAs and their broken down subattributes or the design properties can be completed with manual search and Snowballing. The selection procedure for the pure identification of the QAs is conducted using the same procedure as in **Stage 1.1: Evidence on the role of QAs in the migration from monolithic to microservice architectures**. This can be illustrated with the help of Figure 3.2. Likewise, another work was added via manual search, but this is not part of the structured search methodology. This described process leads to the actual result of a total of 13 works found. In addition to the papers already found, it was ascertained that ISO 25010 [71] was often also used as a reference. Thus, it was important to include this ISO as well, although this is not part of the actual selection process.

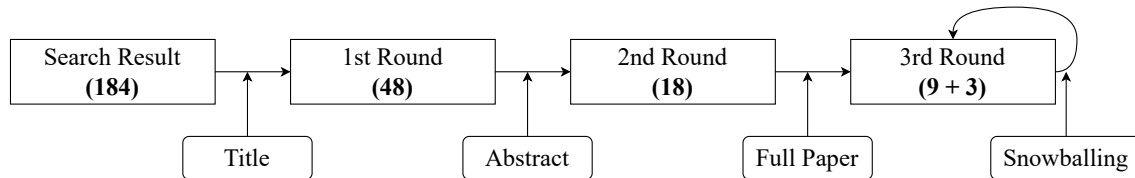


Figure 3.2: Selection Procedure from Stage 1.2.

In this case, the QAs were extracted from the individual works and presented in the form of a table. This was necessary because QAs were often set up as subattributes of these attributes or vice versa. With the help of the table, the occurrences of the respective QAs could then be counted and the subattributes linked so that they could be correctly grouped into the higher-level attributes. In addition, the table could be used to check whether the attributes and subattributes mentioned are present in the product quality model of ISO 25010 [71].

All QAs that were found less than twice in total are not included in the results. When QAs appear as subattributes in the ISO 25010 product quality model [71], they are subordinated to the subattribute level. If subattributes appear as actual QAs in ISO 25010 [71], they are classified as possible QAs on the associated level. In some cases it can happen that QAs also appear as subattributes. If in this case the reference cannot be made to the classification of ISO 25010 [71], the rule is applied that they are assigned to the level in which context they appear more often. In detail, this means that as soon as the number of found attributes as subattributes is greater than or equal to the number of the same attribute as a QA, this attribute is regarded as a subattribute. If, on the other hand, it appears more frequently in the context of the QA, this attribute is assigned to this level. Here, the methodology

for identification refers exclusively to the data found in relation to the number of findings and the reference to the product quality model of ISO 25010 [71]. Semantic classifications or deletions are only made after the findings have been analyzed - this additionally allows to distinguish between properties and actual attributes in the results.

After the QAs were identified, associated descriptions were added using the same literature which was used for the identification. In addition, Snowballing was used to find other literature that could be used to complete the descriptions of each attribute.

Basic Information	
<i>Type:</i>	Literature review on the identification of QAs in microservice architectures with regard to Section 4.1.2
<i>Method:</i>	Rapid Review according to the guidelines from Cartaxo et al. [32]
Search Strategy	
<i>Search String:</i>	((<i>"Title": quality</i>) AND (<i>attribute OR characteristic OR aspect OR concern OR propert* OR requirement OR goal</i>) AND (<i>microservice OR micro-service</i>))
<i>Database:</i>	Google Scholar, IEEE Xplore
<i>Complementary Method:</i>	Snowballing and Manual Search
Selection Procedure	
<i>Inclusion Criteria:</i>	At least two QAs are considered simultaneously, so that the explicit context to QAs is present
<i>Exclusion Criteria:</i>	Too specific for individual use cases; Access to the entire content of the paper not available
<i>Date Range:</i>	2011-2022
<i>Publication Type:</i>	The search result may not be a thesis
<i>Language:</i>	English or German
Quality Appraisal	
<i>Quality Check:</i>	-
Extraction Procedure	
<i>Method:</i>	Collection of notes on respective mentioned QAs and counting the number of frequency of occurrences of the QAs
<i>Collection:</i>	Classification of the respective QA designations found and the description of the respective properties in the form of keywords
Synthesis Procedure	
<i>Type:</i>	Narrative structured listing of the results for each relevant attribute found (if appeared several times in different papers)

Table 3.2: Methodology Definition Protocol of Stage 1.2.

Stage 1.3: Trade-offs and impacts of Quality Attributes are taken into account

In order to identify possible trade-offs between the QAs, the same literature resulting from the results of **Stage 1.2: Identification of Quality Attributes for microservices architectures** was used. During Stage 1.2, in addition to the identification of the individual attributes, indications of possible trade-offs were detected and extracted directly from the full text. In addition to the information collected in this way, the full text of the literature cited was searched again for the search strategy using a keyword search. The following keywords were used: *trade-off*, *trade off*, *tradeoff*, *compromise*, *conflict*, *impact* and *influence*. Based on the extracted indications by the mentioned methods, the complementary method Snowballing was used to find further indications for trade-offs. As inclusion or exclusion criteria it was important that a relationship of exchange between two QAs was explicitly mentioned. By means of the collected data, these could then be collated and presented as results. These are presented textually as well as in a listing for overview.

Stage 2.1: An overview of migration approaches from monoliths to microservices is created

In this case, no literature review is conducted to collect the migration approaches. Since in the context of a doctoral research project by Fritzsich [56] within the Empirical Software Engineering (ESE) group of the Institute for Software Engineering (ISTE) at the University of Stuttgart research is carried out in the area of microservices and specifically also for migration, a list of migration approaches that have already been collected was provided for this thesis. With the help of the list, the individual works could be examined in a structured manner. For this purpose, screening of the abstract and conclusion as well as the full-text search with suitable keywords was used to identify the characteristics of the approaches. The keywords relevant to the characteristics were: *input*, *output*, *technique*, *method*, *model*, *analysis*, *requirement*, *artifact* and *result*. Based on the characteristics collected, the approaches can then be assigned to different categories in order to provide an overview.

Stage 2.2: Mapping of the Quality Attributes to the migration approaches

In order to create a mapping between the attributes, subattributes and properties for the migration approaches, the approaches must be considered individually. For this purpose, the terms of attributes and properties were searched for with the help of a combination of an abstract, introduction and conclusion screening, a full-text search and an examination of the results of possible experiments or studies. As a result, the approaches could then be assigned to the findings in tabular form. It was always ensured that when the terms were mentioned, a proof, a result, an assertion or at least an attempt was presented that suggested that the approaches optimize or improve the attributes towards the target architecture. A mere mention of the attributes without suggested optimization was ignored. For the properties, it was checked whether they were considered by the migration approach and what possible influence the approach has on them in the target architecture.

Stage 3.1: An overview of design patterns and best practices is created

Similar to the migration approaches, in the area of patterns and best practices, literature from a doctoral research project by Fritzsich [56] within the Empirical Software Engineering (ESE) group of the Institute for Software Engineering (ISTE) at the University of Stuttgart has already been submitted or made available, which can be used to identify them. However, an additional manual search was performed to identify possible gaps and to cover as much scientific work as possible. The goal is to identify architectural patterns with respect to microservices and if possible best practices independent of the patterns. All patterns that have been mentioned at least twice during the literature review are included for further investigation. Using the different categorizations of patterns from the literature and combine them, a categorization that is useful for this scope of the work was elaborated, so that an overview of the patterns in the microservice context can be created. In addition, the best practices are based on the work of Lichtenthaler and Wirtz [94] and are supplemented by further indications of best practices where appropriate.

For the manual search, the guidelines of the Rapid Review of Cartaxo et al. [32] have been followed as closely as possible. This results in the definition protocol in Table 3.2.

Search Strategy	
<i>Keywords:</i>	<i>patterns, best practices, microservices, anti-patterns, quality attributes</i>
<i>Database:</i>	Google Scholar, IEEE Xplore
<i>Complementary Method:</i>	Snowballing
Selection Procedure	
<i>Inclusion Criteria:</i>	A reference to QAs is present
<i>Exclusion Criteria:</i>	Too specific for individual use cases; Access to the entire content of the paper not available
<i>Date Range:</i>	2011-2022
<i>Publication Type:</i>	The search result may not be a thesis
<i>Language:</i>	English or German
Quality Appraisal	
<i>Quality Check:</i>	-
Extraction Procedure	
<i>Method:</i>	Tabular listing of the mentioned patterns and best practices and counting of the references.
<i>Collection:</i>	Differentiation between patterns and best practices and classification based on appropriate terms
Synthesis Procedure	
<i>Type:</i>	Narrative structured listing

Table 3.3: Methodology Definition Protocol of Stage 3.1.

With the help of manual search and Snowballing, in addition to the scientific papers provided for the overview of the patterns (4 in total), two more papers and one website could be found.

In addition to the work by Lichtenthäler and Wirtz [94], which is intended to function as a basis, two further works could be referred to for identification of best practices for microservice architectures. Additionally, through Snowballing, the work of Wolff [150] could be found, which establishes basic rules for the successful development of architectures in the microservice context.

Stage 3.2: Mapping of the Quality Attributes to the design patterns and best practices

In order to assign the established QAs to the design patterns and best practices identified with the methodology of **Stage 3.1: An overview of design patterns and best practices is created**, the indications and evidences from all treated works are collected. For this purpose, a table was created wherein for each pattern the QAs and subattributes could be put into relation, which could be found in the literature and explicitly assigned to the pattern. Therefore, the respective references were attached in addition to the indications. The same tabulation was done for the best practices.

Stage 3.3: Trade-offs of the Quality Attributes in the design patterns are considered

Using the table for patterns from the methodology of **Stage 3.2: Mapping of the Quality Attributes to the design patterns and best practices**, additional possible trade-offs were added in the area of QAs. For this purpose, indications and direct mappings were collected again, although only two works directly addressed the trade-offs. In the area of best practices, no indications or evidence of trade-offs could be identified in the literature and thus no listing is provided for this.

3.3 Quality Model and Implementation

After the literature review is completed, a quality model is designed based on the results of the previous findings, which maps the quality goals in a migration process to the approaches and to the patterns and best practices for a microservices architecture. In addition, the described trade-offs of the quality goals are taken into account, which should then be included in the quality model. With the quality model as the core of the work, a proposal for a guide through the migration process (or the selection of suitable approaches or patterns/best practices) is then made possible, in which the quality goals and the associated trade-offs are considered. An exemplary representation of a guidance based on the goals is useful here and will be included.

The implementation of the now designed quality model as an essential part of the web-based application ‘Architecture Refactoring Helper’ is then carried out. This model should afterwards expand the existing guiding tool based on the quality goals. For this purpose, established SE standards for the prototypical implementation will be employed. In addition, the structure of the user interface is designed based on the design elements of the existing tool. To evaluate its usability, a number of software professionals experienced in microservices migration will be interviewed.

The following chapters are the outcome of the findings that could be achieved with the help of the previous methodology. A detailed description for the creation of the quality model, the implementation steps and for the process of evaluation will be addressed separately in the corresponding chapters.

4 Quality Attributes in Migration to Microservices

Two crucial goals of this work are to answer the first two research questions *RQ 1* and *RQ 2*, namely how the choice of the migration approach and the architecture pattern in a migration scenario can be supported with the help of the quality goals and where these can be incorporated in the process. These questions can be answered by first examining the role of QAs in the migration and in which phases they can be incorporated. QA are then explicitly identified in the context of microservices and assigned to the migration approaches and architectural design patterns and best practices that optimize them toward the target architecture. In addition, possible trade-offs and interdependencies among each other are examined.

4.1 Optimization of Quality Attributes as a Driver

There are many different drivers for migrating from a monolithic to a microservice architecture. From the results of a survey by DZone [61] why a microservice architecture is used in the employer company of the respondents, the main reasons for a migration with the most answers can be identified. While easier scalability (80.7 %) and faster deployment (69.5 %) are the top drivers, there are also drivers which aim for improving quality. The third and fourth most frequently given answers are the quality increase by focusing teams on only one part of the application (50.1 %) and by better identifying the source of failure (40.9 %). This shows that the quality aspects play a major role in the migration from monolith to microservices. For example, a semi-structured interview by Bogner et al. [21] shows that a main driver for migration is maintainability, which is known as a QA. Furthermore, from the same findings of Bogner et al. [21] it results that a microservice architecture can improve the quality of a software with respect to the QAs. When choosing a suitable architecture, the discrepancy between the approach and the desired quality goals should be considered [144]. In addition to the functional requirements, the non-functional requirements and/or the QAs have a crucial effect on the target architecture [128].

It becomes apparent, however, that there is a lack of understanding of the meaning of QAs in this regard [93]. Therefore, insights into the QAs related to approaches can help practitioners when targeting the driver for the migration of quality improvement.

4.1.1 Role in Migration

In general, the question must always be asked if a migration to a microservice architecture makes sense and is necessary, and if so, the selection of the target architecture must be chosen with care [93]. In order to choose the right approaches to migration, a decision must be made, which in turn can be supported by considering QAs in this process [93].

As also mentioned in [128], the QAs initially play an important role in the identification or design phase for microservices of the migration process. It is crucial here to see it as a whole process and to identify the QAs accordingly. This identification can be carried out by various involved practitioners such as requirements engineers or software architects.

This assumption is also reinforced by the fact that a baseline must always be created for the architectural refactoring process or goals should be defined, which in turn can be represented by defined QAs [155].

From this raises the question of how a possible identification of the QAs in the process could be proceeded. For example, in the identification phase according to [155], the ‘Quality Stories’ mentioned in the work can be used. These are based on agile user stories and can be used to define the desired attributes, the results expected from the choice of attributes and the acceptance of trade-offs. With this, QA-scenarios can help to define the goals in advance.

Another way to do this is described by Starke [136], whereby in principle it is stated, that software can be assessed by artifacts or processes. Here it is mentioned that artifacts are for example requirements or architectures. In addition it is pointed out that architectures are particularly suitable for the assessment, since these are created or defined early in the development process and the decisions have a large influence on the system. In this respect, it is important that quality goals, such as maintainability, which must be met by the architecture, can be identified. For this purpose scenarios are particularly useful, since these can describe, which criteria should be fulfilled by a system in functional and non-functional regard. Thereby the reference to QAs emerges.

This leads to the question, how the procedure is arranged with the assessment of the architecture regarding the quality goals. For this, Starke [136] refers to the **ATAM**, which consists of different phases leading to the architecture assessment. The ATAM can be described in detail for a better understanding by Kazman et al. [84]. In the context of this work the concrete assessment phase is of importance. Here the quality goals are defined by the stakeholders. The further process of the method is described by Starke [136] as follows: With the help of brainstorming or by qualitative statements of the stakeholder, a quality tree can be provided, which visualizes the specific quality characteristics of the architecture. To refine the criteria, scenarios can be added to the individual quality goals. As an example, a quality tree with scenarios can be illustrated in Figure 4.1. The example scenario contained therein specifies that a certain throughput is required so that more than 1000 real parallel users can be processed with a response time of less than one second. This addresses the QA of performance, which is required by the architecture. In addition, the scenarios can be assigned an importance based on a scale (A = important, B = medium, C = less important), depending on how they are significant from a usage or business point of view. A difficulty according to the technical aspect can also be assigned to the scenarios using a similar scale.

Likewise, the author clarifies that a subsequent actual assessment can be made in several ways, although there is no correct way, as this mostly depends on subjective assessments. However, the procedure of the quality tree with scenarios gives an overview of the concrete quality goals for the architecture with weighted prioritization [136].

In this respect, QAs already play a major role in the assessment phase for the target architecture and can be collected by using the ATAM described above.

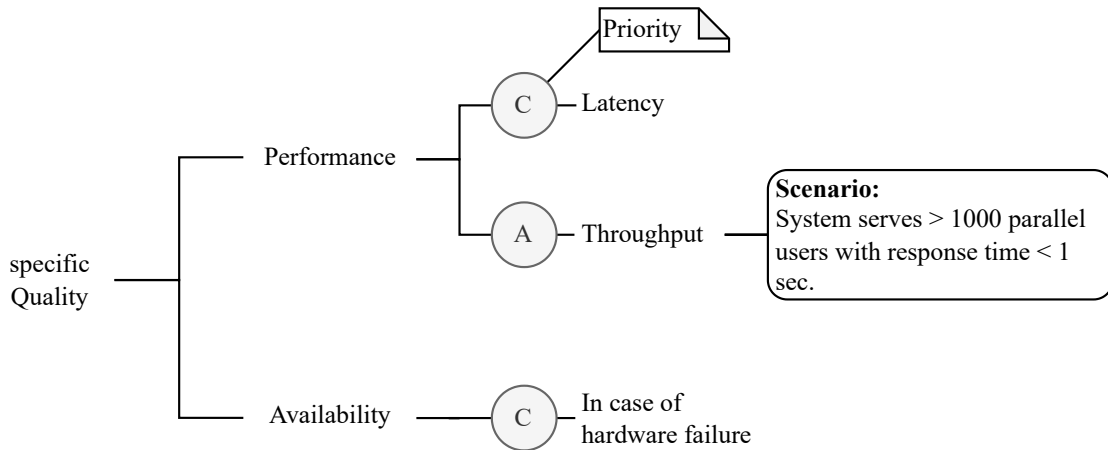


Figure 4.1: ATAM Quality Tree with Scenarios based on Starke [136].

However, in order to examine the role of the quality aspects in the entire migration process, the other phases must also be considered. There is a project by Capuano [29] that aims to develop a process and optimize the quality of the initial system through the migration and guide practitioners in the process. In the next step, as part of Capuano’s research project, Capuano and Muccini [30] examined how and in which phases of the process the quality aspects in migration approaches are considered by researchers. The result of the research revealed that the quality aspects are taken into account by researchers at different phases during the migration process. These play a role above all in the identification and assessment phase, but also in the system comprehension phase. Moreover, they introduce the term ‘quality-driven migration’ - i.e. any migration approach in research that considers QAs, which the target system is supposed to meet or even improve by migrating to a microservice architecture. Nevertheless, in [30] it is also mentioned that although the QAs are taken into account in the migration phases, it can be deduced from the results that they are often not specifically taken into consideration in order to optimize the system. This is due to the fact that only a few approaches already pay attention to QAs in the system understanding phase.

4.1.2 Quality Attributes for Microservices

With the help of the literature review and the associated methodology from **Stage 1.2: Identification of Quality Attributes for microservices architectures**, 14 top-level QAs could be identified from a total of 32 potential QAs for microservice architectures from 13 found papers [7, 13, 21, 30, 37, 53, 69, 93, 94, 103, 114, 126, 144]. The identified and in the product quality model of ISO 25010 [71] already existing attributes are: *Reliability*, *Performance (Efficiency)*, *Security*, *Compatibility*, *Maintainability*, *Portability* and *Usability*. Additional (microservice specific) attributes are: *Scalability*, *Independence*, *Granularity*, *Cohesion*, *Coupling*, *Execution Cost* and *Organizational Alignment*. The *Usability* attribute was removed from the list because, although it is included in ISO 25010 [71], it is aimed at the usability of the system or software product by users, as described in ISO 25010. However, this is not relevant for the context of this work, since the QAs should be used as target attributes for migration approaches and architecture patterns by architects or developers. *Independence* is directly related to *Portability* (from ISO 25010 [71]) and is therefore grouped with it, but remains as a separate QA, as it is also relevant

to other areas besides *Portability*. For example, independent deployment can be achieved, which would be subordinated to the attribute of *Portability*, but also allows independence in development by individual teams [119]. The same applies to *Scalability*, which occasionally appears as a completely independent attribute in the papers mentioned, but is also frequently considered as part of *Performance*. For example, the paper of Francesco et al. [53] mentions that *Scalability* is included in *Performance*. Therefore, this attribute is also grouped.

The two attributes *Execution Cost* and *Organizational Alignment* are specific *Business Attributes* in this case, since they are influenced by several external business factors and are thus treated separately, but are still on the same level as the QAs. On closer inspection, *Cohesion*, *Granularity* and *Coupling* are not QAs, since they cannot indicate the quality of a microservice architecture. Rather, these terms are properties of a system or a specific architecture. In order to differentiate between QAs and the System Properties (SPs) now mentioned, the question must always be asked if the terms mentioned can be optimized and the quality increased as a result, which in turn points to QAs. Or whether only properties of a system are described, which are realized in different forms to a certain degree, which then points to SPs. For example, an attempt can be made to improve the *Maintainability* of an architecture through migration. However, the *Granularity* itself cannot be optimized for this. But the degree of *Granularity* can be set in such a way that the QAs can be optimized. This also implies that the QAs are closely related to the SPs or that the SPs influence the QAs and, conversely, the QAs are achieved by implementing certain SPs to a certain extent.

In addition to the QAs, associated subattributes could be identified. With the help of the methodology from **Stage 1.2: Identification of Quality Attributes for microservices architectures** 28 out of 31 possible subattributes could be extracted. However, this also includes the already assigned SPs *Coupling*, *Granularity* and *Cohesion*. Among the remaining 25 candidates, the same previous classification was made to distinguish between SPs and quality (sub)attributes. This results in that the subattributes of the QA of *Independence*, namely *Isolation* and *Autonomy*, are also SPs. The subattributes *Technology Heterogeneity* and *Complexity* of the QA *Maintainability* are thereby also assigned to the SPs.

The frequency of the found QAs and their associated subattributes in the papers is illustrated in Figure 4.2. Based on this overview, it can be concluded that the QA of *Maintainability* and the associated subattributes are most frequently considered in relation to microservices. The second most common reference is to *Performance*, its subattributes and the *Scalability* grouped with it. While the number of mentions of *Reliability*, *Security* and *Portability* (+ *Independence*) is in the middle range, the QA *Compatibility* and the newly introduced *Business attributes* are the least found in the literature.

An overview of the result of the selection of the QAs and their subattributes can be given using Figure 4.3. In addition, the SPs resulting from the previous distinction can be listed for an overview: *Granularity*, *Coupling*, *Cohesion*, *Isolation*, *Autonomy*, *Complexity* and *Technology Heterogeneity*.

In the following, a brief description of the individual QA and their subattributes is provided. Additional to the description of the characteristics of the attributes, possible metrics are mentioned, with which the degree of the attributes can be measured or verified. For this purpose, general definitions for software products or systems are used, for which a more specific description in the context of microservice architectures is additionally included, if possible.

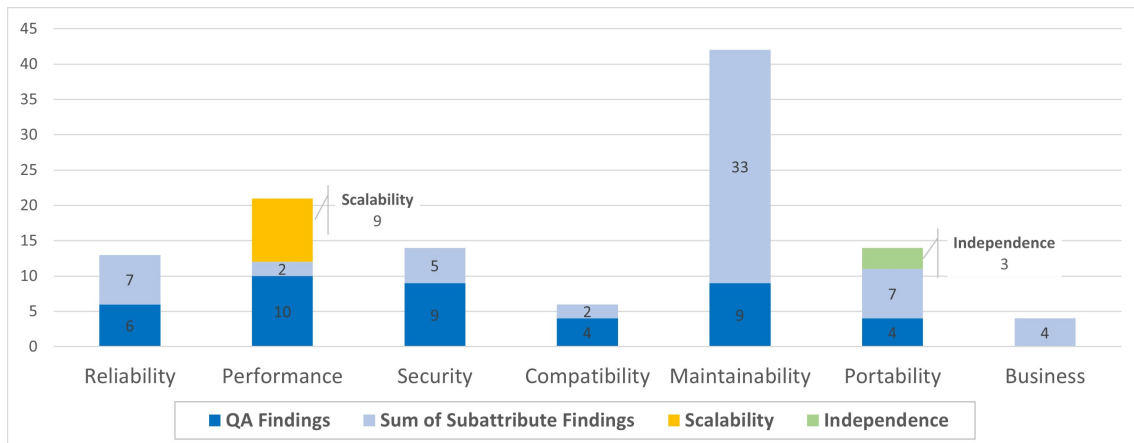


Figure 4.2: Number of Occurrences of the Quality Attributes in the 13 found Papers.

Quality Attributes

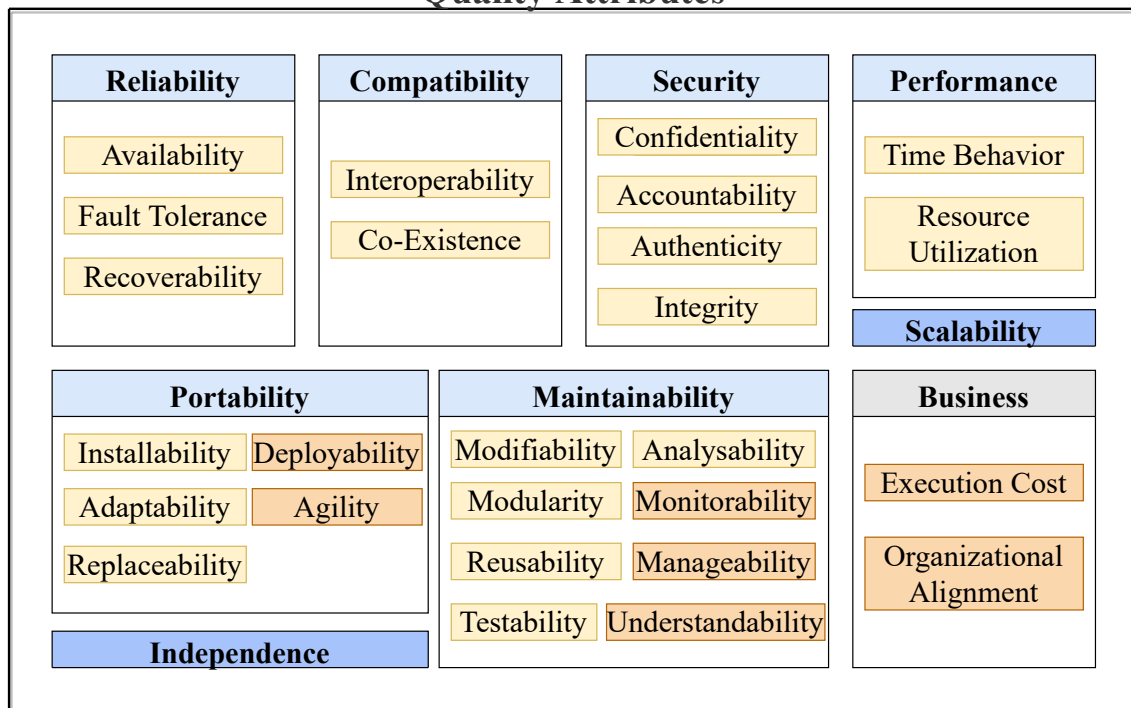


Figure 4.3: Overview of the identified Quality Attributes and their Subattributes.

Reliability

This attribute is defined as the “degree to which a system, product or component performs specified functions under specified conditions for a specified period of time” [71]. In the context of microservices, the terms *Health Management* or *Resilience* are also used [7]. According to Cojocaru et al. [37], in the aforementioned context, this attribute is about how failures are handled. That’s why, they explain that for this purpose, the state is saved and then the service is restored with the

state that is available last, which in turn is very important for dealing with a failure. According to Auer et al. [13], possible metrics for *Reliability* are mean time to repair, mean time to failure and mean time between failure.

Availability is a subattribute of *Reliability*. According to the definition from ISO 25010 [71], this describes that if a system or component (in this case a microservice architecture) is required, the degree to which it can be used or is accessible. The *Availability* should always be given to the best degree [103]. Additionally, in an architecture, the overall *Availability* can be greatly increased if the *Availability* of a microservice is increased even slightly, but the microservice has a strong coupling to other services [37]. Thereby, the attribute is not influenced by the service itself, but rather this is often also very dependent on the platform of the deployment [37]. The *Availability* can be specified metrically with the help of the specification of *Availability* in percent or the mean time between downtimes [13].

Fault Tolerance describes the “degree to which a system, product or component operates as intended despite the presence of hardware or software faults” [71]. Also, it includes the possibility that if partial errors occur, that the system can be recovered. It also means that the errors should not be spread to other parts of the system [7]. To achieve a high degree of *Availability*, this attribute should always be considered in a microservice [93]. According to Auer et al. [13], the number of bugs, number of features blocked or code coverage can be used as metrics.

Recoverability is defined as the “degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system” [71]. This attribute must already be taken into account during the design phase of the microservice architecture, so that a process can be defined in advance that describes how a recovery can be performed [93]. This includes, for example, automated restarts [94].

Compatibility

The attribute means the extent to which a system or component (in this case a microservice) is able to interact with other systems or similar in a homogeneous environment [71]. To achieve *Compatibility*, it is helpful that uniform distribution channels and protocols are used for communication [21].

Interoperability is more specific than *Compatibility*, and refers to the exchange of data and to the subsequent use of data between systems or components [71]. For this, however, they do not have to be in the same environment. A possible metric is the percentage of requests accepted [144].

Co-Existence, on the other hand, then involves the degree to which a product functions in a shared environment without having unwanted influence on other parts of the system [71].

Security

The term describes the extent to which the data and other information of a system are protected, which therefore also includes which systems and persons have access to this information [71]. In the context of microservices, this attribute must be treated with caution due to the communication between the services [7]. This is necessary, because the division into multiple microservices creates a more complex network for communication in terms of *Security*, thus creating more opportunities for attacks [93]. The level of security depends on many different factors such as the

technology used or the implementation [37]. Security hotspots and configurations can be used to analyze *Security* [103]. Another possibility would be execution timelines in combination with UML diagrams of the system [37].

Confidentiality as a subattribute of *Security* specifies how and to what extent it is ensured that only authorized persons or systems have access to the information of a system [71].

Accountability indicates the degree to which actions performed by an entity in or with a system can be traced [71].

Authenticity describes whether and to what extent a system is able to recognize and confirm the actual identity of an entity [71].

Integrity, according to ISO 25010 [71], is the “degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data”.

Performance

The *Performance* or often also *Performance efficiency* determines the degree of capability of a system under certain conditions in relation to the resources used [71]. In the area of microservices, this is strongly dependent on internal optimization and the technologies used [37]. The metrics for the *Performance* are considered rather indirectly, since one is for example the response time [13], which is actually part of the subattribute of *Time Behavior*.

Time Behavior is fundamentally the “degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirement” [71]. Thereby, in the context of performance, this attribute is about the response time and the waiting time [13, 137]. The response time is formed from the execution time and the delay time [137] (or the network delay [37]). This is the interval in time between the request sent to a service and the response or result received [13, 137]. This is important to guarantee the performance [37]. The waiting time is the elapsed time that a request spends in the queue or that is needed until the start of processing in the service [13]. In general, *Time Behavior* consists of the aforementioned metrics themselves, which can be measured in seconds, for example [137]. Another metric would be throughput in a given time [13].

According to ISO 2501 [71], **Resource Utilization** is the degree to which the resources of a system are used in the execution of its functions. For example, CPU consumption or memory consumption can be used here [13]. This in turn could also be considered as a direct metric for *Performance* [13].

Scalability

Scalability stands for the ability of a system to add or remove resources so that they can be used effectively, which means that this allocation is not associated with a lot of effort and that running operations are not interrupted as a result [19]. Likewise, no loss of *Performance* should be accepted as a result of scaling [151]. In the case of *Scalability*, a fundamental distinction is made between the horizontal and the vertical, where the horizontal means the allocation of resources to logical units and the vertical for the allocation to physical units [19]. In the context of microservice architectures, the horizontal type is particularly important, as it enables scaling out through different instances of

a microservice [93]. According to Yu et al. [151] the degree of *Scalability* can be measured “by analyzing the distribution of synchronous requests provided by the exposed interfaces, a high diversity of requests indicating poor scalability.”

Portability

This attribute describes how easily the platform or environment of a software or system can be changed [19, 71]. This attribute can be achieved, for example, by encapsulating dependencies or by reducing platform dependencies to a minimum [19]. In this respect, the use of container technologies (for example, Docker) can be an advantage towards achieving the attribute [21].

Installability is, according to ISO 25010 [71], the “degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments”.

Adaptability indicates the extent to which a system can adapt to changing or evolving environments or platforms [71].

Replaceability describes the degree to which another system or software in the same environment can be replaced by a system or software to ensure the same functionality [71].

Deployability is the way a software product or system is delivered to the host or target platform, how it is integrated and how subsequent possible updates are handled [19]. This also includes the question of whether integration during runtime is possible [19]. For this, a high degree of the attribute is required in the software development and design process [103]. Possible metrics for measuring *Deployability* include build time, deployable size or pipelines [103].

Agility stands for the ability of a software product to what extent it is possible to carry out simple adjustments, whereby this is in the context of development and is therefore also highly dependent on the way of integration and the platform [37].

Independence

In the field of microservices, this attribute represents the fundamental separation of the individual services from each other, so that they only communicate with one another via interfaces [37]. This implies that each microservice assembles and compiles individually and without dependency on other services [103]. The *Independence* of microservices from among one another means that independent development teams can also derive advantages from this [13, 103]. In order to make the *Independence* measurable or detectable, it is useful to create dependency graphs of the architecture [7].

Maintainability

According to ISO 25010 [71], this attribute describes the ability of a system or software product to be adapted, modified or corrected, including, for example, “installation of updates and upgrades” [71]. This attribute should be considered in the best case already during the development of the software product, since in such a way in the long run the costs of the maintenance can be lowered [25].

Modifiability describes according to Bass et al. [19] with which simplicity a system can be changed. This includes the aspect that the already existing quality may not be impaired by the change at the software product [71]. Factors for the measurability of the attribute can be for example the number of lines of code [13].

Modularity is the “degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components” [71]. Here, also the division by business responsibilities plays a role among other things [37]. A metric for the degree of *Modularity* can be, for instance, the code complexity [13].

Reusability is described as the “degree to which an asset can be used in more than one system, or in building other assets” [71]. In order to enable this attribute, the costs for the development must first be increased to then produce lower costs in the future due to *Reusability* [37].

Testability is “degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been” [71]. *Testability* can be measured by coverage of the tested code [13] or by test covered conditions [103].

Analysability is defined according to the ISO as follows: “degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified” [71]. In the microservice context, this subattribute can be made measurable, for instance, by the number of individual microservices or by the interactions between the individual services [13].

Monitorability is the attribute that describes to what extent the system or software product can be controlled or monitored during execution in order to detect problems during runtime or to inspect the system [19]. Due to the fact that the dynamics in a microservice architecture increase and the complexity of the architecture is higher, this attribute is of major importance in this context [93].

Manageability describes the “degree of centralization” [37], whereby the classification is such that less centralization is better [37].

Understandability is the degree to which a system or organizational factors of a system can be understood in relation to the context of the system [72].

Business

An additional attribute group is formed by the introduced *Business* attributes. This includes attributes that do not describe the quality of the microservice architecture itself, but those that address external factors in the immediate context.

Execution Cost is according to Sun and Zhao [137] “the cost associated with the invocation of this service”. The costs here can be composed of the costs for the infrastructure and the effort, so for example on the infrastructure side the costs for the cloud environment and on the effort side the costs for development, testing, maintenance or deployment [13]. The consideration is of great importance, especially in the industrial domain [37]. This attribute can be measured in a number in a currency [137].

Organizational Alignment in this context, according to Cojocaru et al. [37] (who in turn quotes Canway's Law [38]), represents the extent to which the structure of the teams and the communication between them can be mapped to the structure and dependencies within the microservice architecture. To that extent, this includes everything related to the organization of the teams, such as team assignment for single responsibility for a microservice [103]. This also includes the coordination between the individual teams as well as the ability to work autonomously [13].

System Properties

The SPs are described below by providing a basic description of each property, supplemented by a reference to microservices when possible.

Granularity describes the size of a microservice [37] or the size of the decomposition in a microservice architecture [23]. This attribute is always objected to, since there is no correct optimal size for a single microservice [37].

Coupling essentially describes how strong the connections or dependencies of components or modules of a system are to each other [23, 37, 72]. According to Hutapea and Wahyudi [69], in addition to the dependencies on one another, the "power of interaction" in a system is also one of the characteristics of the property in relation to services. *Coupling* can be measured with the help of internal microservice coupling, which "calculates the sum of the dependencies between the modules inside the microservice" [151] and with weighted microservices coupling of interface which "measures the degree of dependency between microservices and microservices in a system" [151].

Cohesion encompasses the extent to which a microservice is limited to a standalone functionality [23, 37]. In other words, it describes the extent to which individual functions within a microservice are interrelated [72]. In this context, according to Cojocaru et al. [37], this can be measured with the help of the equality of the microservice interfaces in terms of parameter types and names.

Technology Heterogeneity describes the degree to which one can be unconstrained in deciding between programming languages or technologies used based on the characteristics of a microservice architecture [7].

Isolation is about the isolation principle, which according to Cojocaru et al. [37] means that the individual microservices in an architecture are designed in such a way that they are self-contained in the deployment and development process (also with regard to teams) and also do not depend on each other.

Autonomy is defined in the context of microservices as follows: the "microservices are responsible for the execution of their own underlying processes, which can be initiated, operated, and shut down independently. They can be also developed and deployed separately within their own development lifecycle" [97].

Complexity describes the "degree to which a system's design or code is difficult to understand because of numerous components or relationships among components" and the "degree to which a system or component has a design or implementation that is difficult to understand and verify" according to ISO/IEC/IEEE 24765 [72]. In addition, the "amount and variety of internal work carried out" [23] in the context of services is included here. The *Complexity* can be measured

by means of the microservice model propagation cost, which specifies the dependencies and the information flow between modules and thus indicates the degree to which changes must be made to other modules when a module is changed [151]. Another metric, according to Yu et al. [151], is the microservices parameter count, which indicates the number of differing data structures - more specifically, those being used in the interfaces.

4.1.3 Correlation System Properties and Quality Attributes

The introduced SPs can have different influences on the QAs in a microservice architecture by increasing or decreasing them. In this respect, directional effects can be identified for each SP by adapting the properties and are explained for this purpose and subsequently listed in a structured manner.

For example, how granular the microservice architecture is structured has an impact on *Modularity*, as this increases the *Complexity* of the architecture, assuming the *Granularity* becomes higher [37]. Also, due to the increased *Complexity* caused by more *Granularity*, *Performance* is negatively influenced as a result, since more interaction between microservices is thus inevitably required [50]. Moreover, based on fine *Granularity*, *Maintainability* should be improved [37]. According to Li et al. [93] a reduction of the *Resource Utilization* and a decrease of the *Execution Costs* can be achieved by a high *Granularity*, since thus only the individual microservices are scaled instead of the entire architecture. Here, Li et al. [93] mention in the same context that these attributes are positively influenced by loose Coupling.

For the *Coupling* attribute, there are other effects on certain qualities of the architecture in microservices. If the “coupling increases reusability of services and the system performance decreases” [39]. In addition, however, Daghighzadeh and Babamir [39] also mention that loose Coupling can improve *Maintainability* and *Testability*. Also mentioned by Daghighzadeh and Babamir [39] is that low Coupling “directly contribute to more reusability and composability of services”. A similar conclusion can be made from the works [22] and [23], as they state that a small number of Couplings between services simplify maintenance and thus increase *Maintainability*. A small degree of *Coupling* can also improve *Manageability* [37]. There are no directed effects for the *Independence* attribute, but this attribute is strongly influenced by the degree of *Coupling* [37].

Maintainability is not only affected by *Granularity* and *Coupling*, but also a high degree of *Cohesion* can increase *Maintainability* [22, 23, 109]. Moreover, similar to a low degree of *Coupling*, a high degree of *Cohesion* can strengthen *Manageability* [37]. Similarly, again in the *Maintainability* domain are the *Analysability* and *Testability* attributes which are improved by high *Cohesion* [109]. In addition, high *Cohesion* can help improve *Reusability* and composability [39]. So in principle, high *Cohesion* mainly causes the improvement of *Maintainability* and its associated subattributes in a microservice architecture.

For the *Autonomy* and *Isolation* for a microservice, this mainly implies *Independence*. For this purpose, according to Alshuqayran et al. [7], *Isolation* and *Autonomy* are explicitly emphasized as an alternative term and a common meaning for *Independence*, respectively.

As with *Cohesion*, the SP *Complexity* primarily influences the *Maintainability* of a microservice architecture. Thus, increased *Complexity* leads to a reduction in *Maintainability* [23]. According to Cojocaru et al. [37], as already mentioned in the case of the *Granularity* property, *Modularity* is influenced by the *Complexity* of the system.

Although the possible *Technology Heterogeneity* is among the advantages of microservice architectures, an improvement in *Maintainability* is achieved by a particularly reduced *Technology Heterogeneity* or a high technology homogeneity within the microservice architecture [37].

Based on the implications that have emerged, the relations of SPs to QAs and/or possible subattributes can be established. For this purpose, the results can be summarized and presented using Table 4.1.

	Impact of increase (+)	Impact of decrease (-)
<i>Granularity</i>	Modularity (↑/↓) Performance (↓) Maintainability (↑) Resource Utilization (↓) Execution Cost (↓)	Modularity (↑/↓)
<i>Coupling</i>	Independence(↑/↓) Reusability (↓) Performance (↓)	Independence(↑/↓) Maintainability (↑) Testability (↑) Reusability (↑) Manageability (↑)
<i>Cohesion</i>	Maintainability(↑) Reusability (↑) Manageability (↑) Analysability (↑) Testability (↑)	
<i>Isolation</i>	Independence (↑/↓)	Independence (↑/↓)
<i>Autonomy</i>	Independence (↑/↓)	Independence (↑/↓)
<i>Complexity</i>	Modularity (↑/↓)	Modularity (↑/↓) Maintainability (↓)
<i>Technology Heterogeneity</i>		Maintainability (↑)

Table 4.1: Influence of System Properties on Quality Attributes.

The table shows the impacts on the QAs of increasing or decreasing the SP. For this purpose, the QAs are assigned to the corresponding column. An arrow pointing upwards indicates that the QA is improved, and an arrow pointing downwards indicates that the QA is reduced. If the QA appears in both columns and both arrow directions are indicated, the SP has a direct effect on the QA, but no directional statement can be made.

The results also imply that the opposite view can be taken. Based on the improvement or reduction of the QAs, it can be concluded which SPs must be present to a low or high degree in an architecture, so that the QAs are improved or reduced as a result. This can be illustrated by an example: The result shows that the *Maintainability* can be improved by reducing the *Coupling*. This in turn suggests that a low degree of *Coupling* within the architecture is required for high *Maintainability*.

4.1.4 Quality Trade-Offs

In the area of microservice architectures, various trade-offs between the individual attributes can be identified. The trade-offs can occur across attributes and subattributes.

In the area of *Security*, these are in conflict with *Maintainability*. This can be reasoned by the fact that after Milić and Makajić-Nikolić [103] “security should be maintained at a high level”. From this, it can be derived that the fulfillment of a high *Security* of an architecture increases the effort of the maintenance, which has the consequence that the *Maintainability* is decreased. Likewise, towards decomposition from a monolith into a microservice architecture, the balance between *Security* and *Scalability* is required to enable requirements elicitation [4]. In the area of *Security*, there is another trade-off that needs to be considered when choosing the methodology of decomposition and architecture. According to Li et al. [93], certain tactics for authentication and authorization (to achieve *Security*) lead to a trade-off between *Performance* and *Security*.

Even though *Performance* and *Scalability* are considered together in certain cases [53], there is a trade-off between these two attributes [86, 93]. Here, minimizing the size of individual microservices in an architecture can increase *Scalability*, but in doing so, *Performance* is reduced due to the resulting increase in the number of required interactions in an architecture. Conversely, increasing the size or combining potential microservices can increase *Performance* for the aforementioned reasons, but it can decrease *Scalability* as a result [86, 93].

Availability in a microservice architecture is not inherently high, but must be maintained so that it can be achieved [13]. According to Auer et al. [13], this “is not a simple task”. From this it can be implied that the effort to achieve good *Maintainability* is increased as long as high *Availability* is required. Therefore, a trade-off between *Availability* and *Maintainability* ensues.

Microservices can always undergo changes and are complex due to the relationships between each other [93]. Therefore, if a high degree of *Modifiability* is given, these changes can occur. This, according to Li et al. [93], must be considered on the *Testability* of the architecture, so that possible effects on other attributes of the architecture are limited.

Based on the derivations from the literature, the following trade-offs result for QAs in microservice architectures. Here, these influence each other bidirectionally.

- *Security* \iff *Maintainability*
- *Security* \iff *Performance*
- *Security* \iff *Scalability*
- *Availability* \iff *Maintainability*
- *Performance* \iff *Scalability*

- *Modifiability* \iff *Testability*

It should be emphasized here that *Security*, *Maintainability*, *Performance* and *Scalability* occur in several trade-off relationships, with other attributes not mentioned in any such relationship.

4.2 Migration Approaches Overview

In order to get an overview of existing approaches for migrating from monoliths to microservice architectures, the 90 migration approaches provided by the Institute of Software Engineering at the University of Stuttgart are first grouped into categories. Essentially, the approaches presented are different strategies that can be used to decompose the monolith application [58].

Since classifications for migration approaches already created by other researchers exist, they are used for the categorization. A classification of the types for this was set up by Abdellatif et al. [1], which can classify the service identification approaches for legacy applications in the analysis. Here, these analysis types are classified into three types. First of all there is the *Static analysis*. This is based only on the application's source code and can be performed without running the system. In particular, dependencies between classes can be determined. The second classification is *Dynamic analysis*. This collects the required information such as function calls during the runtime of the application. Finally, the *Lexical analysis* is mentioned. This is particularly considerably for approaches that are looking for similarities in textual form and is suitable for examining "the similarity between the classes" [1].

Ponce et al. [112], on the other hand, have made a similar classification with regard to *Static* and *Dynamic analysis*. However, the third class listed is not *Lexical analysis*, but *Model-Driven analysis*. This technique differs in that it uses "design elements" [112] or models as input.

An even more detailed classification of the strategies was introduced by Fritzsche et al. [58]. In addition to *Static Code Analysis aided (SCA)*, there is also the category of *Meta-Data aided (MDA)*, *Workload-Data aided (WDA)* and *Dynamic Microservice Composition (DMC)*. The approaches assigned to the *MDA* category are those that require "more abstract input data, like architectural descriptions in form of UML diagrams, use cases, interfaces or historical VCS data" [58] as input. Included therein as a subset may be the by Ponce et al. [112] introduced group of *Model-Driven analysis*, since models represent abstract input data. The classification of Fritzsche et al. [58] of *WDA* approaches overlaps with the *Dynamic analysis* by Ponce et al. [112] and Abdellatif et al. [1] because the input used here is the operational data of the application that is collected during runtime, such as web-access logs. The fourth classification introduced has the terms of *DMC*. However, it must be ensured that this essentially differs from the *Dynamic analysis* by Abdellatif et al. [1] and Ponce et al. [112]. This includes the approaches that enable a constantly changing service composition by collecting the data required for this during runtime.

Due to the further splitting up by Fritzsche et al. [58] and the fact that the classifications presented by Ponce et al. [112] and Abdellatif et al. [1] each represent a subset or intersection of a category, the use of this classification is reasonable. With the help of the methodology from **Stage 2.1: An overview of migration approaches from monoliths to microservices is created** and the classification of Fritzsche et al. [58], a general overview of the approaches resulted, which is

illustrated in Table 4.2. In addition to the categorization, there was also the possibility of combining the categories [1, 58, 112]. For approaches that are too specific in their spectrum of application, a new category with additional classification was added in each case.

Class	Reference
Static Code Analysis aided (SCA)	[130], [46], [9], [48], [54], [100], [152], [154], [2], [96], [98], [111], [121], [24], [51], [63], [80], [131], [50], [89], [120], [133]
Meta-Data aided (MDA)	[16], [115], [91], [41], [67], [147], [45], [26], [27], [40], [42], [60], [90], [17], [108], [43], [92], [107], [123], [135], [10], [77], [110], [134], [143], [18], [36], [125], [5], [62], [74], [85], [95], [141]
Workload-Data aided (WDA)	[28], [78], [149], [14], [153], [3], [75], [140], [76], [129]
Dynamic Microservice Composition (DMC)	[66], [86], [85], [87], [113], [49]
SCA/MDA hybrid	[55], [47], [44], [11], [12], [73], [122], [35], [34], [132]
SCA/WDA hybrid	[88], [8], [101], [104], [117]
Database specific, MDA	[83]
Workflow specific, MDA	[6], [142]

Table 4.2: Classification of Migration Approaches from Monolith to Microservices.

The overview shows that the largest number of approaches perform decomposition using static code analysis and meta-data. Likewise, the combination of both mentioned is the most common. Since the migration approach of Kazanavičius et al. [83] is only specific to the database migration scenario, this is treated separately. Also treated separately are the approaches of Alaasam et al. [6] and Tusjunt and Vatanawood [142] since these are specifically designed for workflow scenarios and thus are also not applicable to every monolithic application.

An example of how the categorization was done can be clarified by the description of the approach of Kamimura et al. [80]. They state: “We use static analysis as analyzing the source code without execution to collect the dependencies which are program calls and data access” [80]. From the statement it can be concluded that the static code analysis is used without further execution of the program and serves as input. Thus, considering the previously mentioned characteristics of the individual categories, this approach is exclusively included in the SCA group.

4.3 Microservices Design Patterns and Best Practices Overview

For the architecture of microservices, there are various so-called best practices and patterns that can be applied or observed in the design of these. In the following, an overview of existing patterns and best practices is provided and, if possible, these are categorized.

In the literature, there are different categorizations of architecture patterns for a microservice architecture. For this purpose, all occurring categories were first collected and, if necessary, merged or removed from the listing. Accordingly, the works [93, 99, 118, 138, 144, 145, 146] result in the categorization groups of *Distribution*, *Communication*, *Entry Point*, *Deployment*, *Data management* and *Complementary*.

For the selection of categories, decisions were made based on the terminology or possible merging of categories from the literature. The Orchestration and Coordination categories are assigned to the *Distribution* category. The Supplementals category has been renamed to *Complementary* for better comprehensibility. Very specific categories, such as fault tolerance, can be assigned to the *Complementary* category. If it is apparent from the literature that a pattern could be assigned to more than one category, the result is categorization in such a way that the description of a single category best fits the pattern. The *Data management* category is derived from the data persistence and data storage category. Other categories were dropped because all patterns can already be categorized in a sensible way. A description for each of the defined categories for patterns in microservice architectures is presented below:

Communication

This category includes all architectural patterns that contribute to the process of communication among each other through message exchange [99]. In addition to improving communication, a channel can also be created or coordination between services can be improved [144].

Data management

This category includes all patterns that deal with the management of data in the context of microservices [138] or actually improve the data management [145].

Deployment

Deployments of the respective microservices are considered such as containers and the strategies of deployment in the environment of these [138]. According to Valdivia et al. [145] this category includes patterns that support software teams in managing the deployment pipeline.

Distribution

According to Valdivia et al. [144] the category of architecture patterns, which belong to the distribution, are patterns, which can cause an improved structure for organizing integrations into the system. Thereby these “improve the distribution of services in a logical way” [144].

Entry Point

These patterns, which fall into the category of entry points, cover patterns which “are oriented to supply control access and entry point to services or types of backends” [145].

Complementary

A special case is the group of complementary patterns, which solve concrete problems and are therefore very specific [145]. These patterns can improve other patterns by complementing them [145]. For example, an *Application Programming Interface (API) Gateway* pattern can use the *Circuit Breaker* pattern to invoke different services [118].

Using the methodology of **Stage 3.1: An overview of design patterns and best practices is created** an overview of the architectural patterns of microservice architectures can be given. Based on the works of Li et al. [93], Márquez and Astudillo [99], Taibi et al. [138], Valdivia et al. [144], Valdivia et al. [145], and Vale et al. [146] and the website of Richardson [118], which have made efforts for categorizing the patterns, a categorization of the patterns can be presented in Table 4.3.

In the case of some patterns, no unambiguous categorization could be determined on the basis of the various specifications in the literature. In the case of the pattern *DB is the service*, there was information on the category of *Distribution*, *Deployment* and *Data management*. Even though the pattern represents a service in its own and thus can contribute to organized *Distribution*, since the business logic is integrated directly into the database [102], the focus is rather on the data, since “the service is strictly coupled to the data” [102] through the pattern. The categorization in the *Deployment* is not discussed further, since the pattern does not explicitly target the strategies of the *Deployment*. The *API Gateway* pattern was placed in the *Communication* group on the one hand and in the *Entry points* group on the other. Based on the description of Richardson [118] with “Implement an API gateway that is the single entry point for all clients” the assignment is clear - so the assignment to the category of *Entry points* is carried out. For *Backend for frontend* there are different categorizations from the literature, once in *Entry points* and once in *Deployment*. According to Richardson [118], since the pattern is a variation of the pattern *API Gateway*, as it represents a separate *API Gateway* for each of the clients, it is also classified in the category of *Entry points*. All of the remaining patterns are given a unique assignment from the literature based on the merged or renamed categories.

Best Practices

In addition to the architecture patterns, there are also best practices that should be taken into account when designing a microservice architecture. Based on the underlying works of Lichtenthäler and Wirtz [94], Pulnil and Senivongse [114], and Schirgi and Brenner [127], no precise categorization could be made in order to create an overview of the individual best practices. However, according to Wolff [150], there are general principles according to the so-called Independent System Architecture (ISA), which can represent the best practices for the architecture of microservices at a higher level.

According to one of these principles, the entire system should be divided into independent modules that can access each other via interfaces. In addition, these modules must have separate tasks and be their own containers or virtual machines, or represent a single process. Another principle is that two levels must be considered for each system. On the one hand, the micro-architecture, which concerns each individual decision per service, and on the other hand the macro-architecture, which concerns a decision for the entire system and its modules. So, part of the micro-architecture is also that each module has its own Continuous Delivery (CD) pipeline.

Communication	
- <i>Asynchronous messaging</i>	- <i>REST integration</i>
- <i>Change code dependency to service call</i>	- <i>Secure channel</i>
- <i>Competing consumers</i>	- <i>Service discovery</i>
- <i>Event notification</i>	- <i>Service locator</i>
- <i>Log aggregator</i>	- <i>Service registry</i>
- <i>Pipes and filters</i>	- <i>Service registry client</i>
Data management	
- <i>CQRS</i>	- <i>Local sharing-based router</i>
- <i>DB is the service</i>	- <i>Scalable Store</i>
- <i>Event sourcing</i>	- <i>Shared DB server</i>
- <i>Local database proxy</i>	- <i>DB per Service</i>
Deployment	
- <i>Multiple service per host</i>	- <i>Single Service per host</i>
Distribution	
- <i>Anti-Corruption Layer</i>	- <i>Microservice DevOps</i>
- <i>Container</i>	- <i>Self-containment of services</i>
- <i>Deploy cluster and orchestrate containers</i>	- <i>Strangler</i>
- <i>Externalized configuration</i>	- <i>Enable continuous integration</i>
Entry Point	
- <i>API gateway</i>	- <i>Backend for frontend</i>
- <i>Auth-service</i>	- <i>Gatekeeper</i>
Complementary	
- <i>Ambassador</i>	- <i>Health check</i>
- <i>Asynchronous completion token</i>	- <i>Internal load balancer</i>
- <i>Asynchronous query</i>	- <i>Key-Value Store</i>
- <i>Circuit breaker</i>	- <i>Load balancer/load-balancing</i>
- <i>Correlation ID</i>	- <i>Page cache</i>
- <i>Edge server</i>	- <i>Priority queue</i>
- <i>External load balancer</i>	- <i>Result cache</i>
- <i>Gateway Aggregation</i>	- <i>Sidecar</i>
- <i>Gateway Offloading</i>	

Table 4.3: Design Patterns Overview and Categorization.

In addition, the integration options, the types of communication and the authentication should be standardized and limited. One principle is that operations should be standardized so that, for instance, monitoring, configuration and deployment always conform to a standard. Accordingly, the aforementioned standards should be defined at the interface level.

The last principle deals with the resilience of the individual modules in an architecture. A failure of one module may not have a negative impact on the others, and no data or states may be lost in the process.

4.4 Quality Attributes in Microservices Design

The QAs and SPs that have already been filtered out in the context of microservices can be used to define quality goals for the target architecture during migration. The representation of Figure 4.4 shows the role of the QAs and the SPs in the migration process from a monolith to a microservice architecture. It becomes clear that this primarily has an influence on the choice of the migration approach, which leads to a certain decomposition, as well as on the choice of the architecture pattern for the target architecture. In order to be able to achieve these defined quality goals, the QAs and their subattributes must be assigned individually to the respective migration approaches and also to the architectural patterns or best practices.

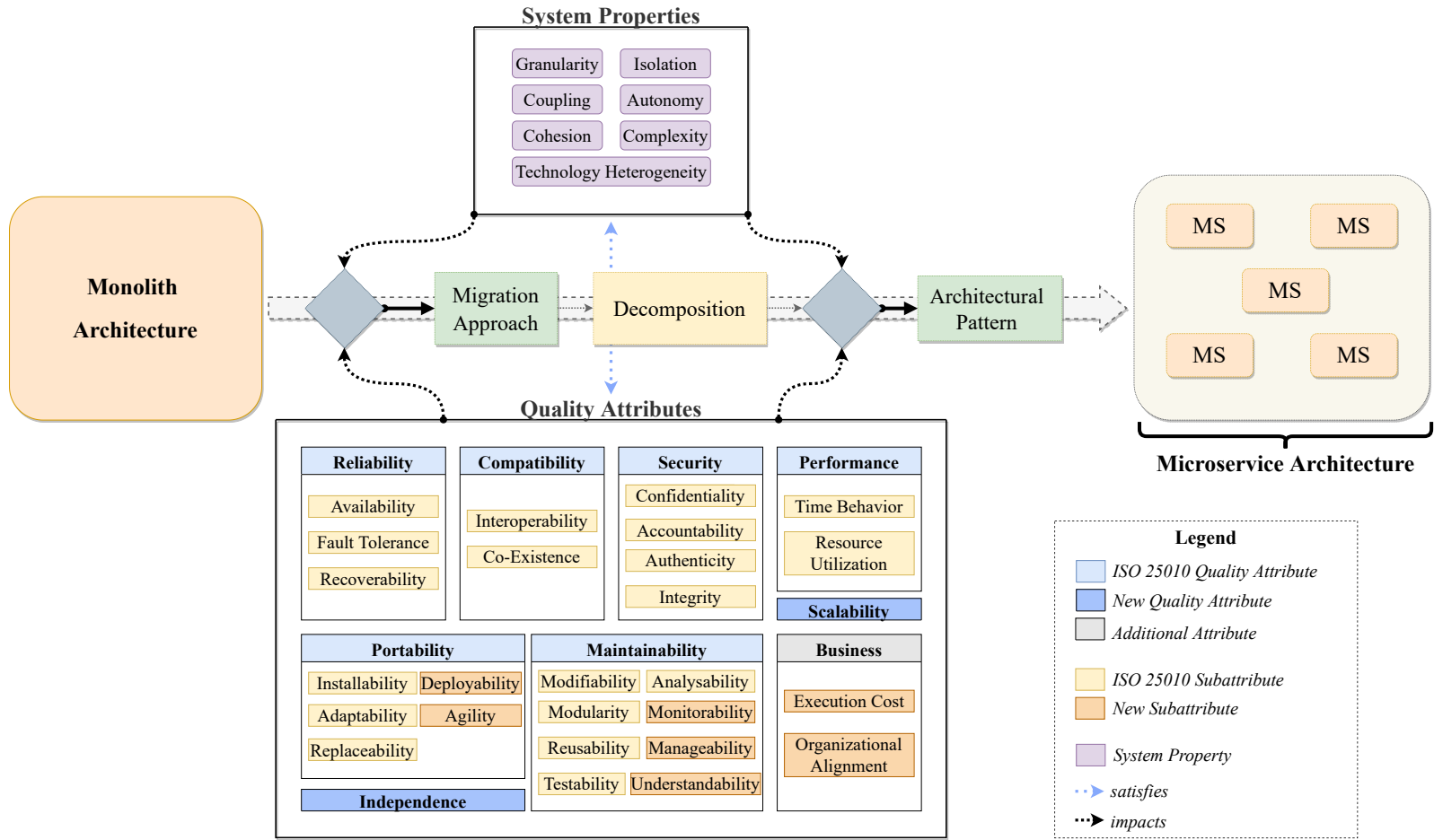


Figure 4.4: Quality Attributes and System Properties in the Migration Process to Microservice Architecture.

4.4.1 From Quality Attributes to Migration Approaches

The QAs, subattributes and SPs identified in Section 4.1.2 can provide an overview of attributes and properties relevant to microservice architectures. However, there are a number of migration approaches that try to optimize various QAs or increase or decrease the degree of certain properties of an architecture in this context. Some even, not only attempt to optimize them, but obtain evidence of successful improvement through the migration approach through proven methods such as comparison of results of measurements with the help of metrics, experiments or interviews.

For this purpose, using the methodology from **Stage 2.2: Mapping of the Quality Attributes to the migration approaches**, the QAs and their subattributes are assigned to the various migration approaches using Table 4.5, which enable or improve them through the migration to the microservice architecture. The result shows that the *Performance* (19 approaches), the grouped *Scalability* (16 approaches) and the *Independence* (12 approaches) are optimized by most of the approaches. However, it must also be mentioned that if the aspect of the subattributes is taken into account, the group of *Maintainability* in particular stands out. If the levels of the QA and the level of the subattributes are considered together, 31 approaches can contribute to an optimization in the group of *Maintainability*. If the subattributes are also considered, there are 22 approaches (+3) in the group of *Performance*. The group of *Reliability* (11 approaches) and *Portability* with 16 approaches also gain in significance as a result. *Security* (4 approaches), *Compatibility* (3 approaches) and the *Business attributes* (5 approaches) are the groups that are optimized by the fewest approaches.

There are also QAs and subattributes that are not improved by any of the 90 migration approaches. These include *Recoverability (Reliability)*, *Accountability (Security)*, *Installability*, *Adaptability*, *Replaceability*, *Agility* (all four from *Portability*), *Analysability (Maintainability)* and *Business*, as well as its subattribute *Organizational Alignment*. These are therefore without any mention.

Only attributes and subattributes that are explicitly mentioned are accepted and included for the results. No assumptions or conclusions are made based on the context of the migration approach or based on statements. However, there are occasionally mentioned attributes that are not present in the existing set of QAs, but which represent a synonym for an attribute contained in the set or are a subset of an attribute contained. One of the cases mentioned represents *Robustness*. As stated by IEEE Standard Glossary of Software Engineering Terminology [70], the definition of *Robustness* is: “The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions”. Moreover, it is directly related to *Fault Tolerance* according to the same reference. Therefore, in this context, *Robustness* is evaluated as a subset of *Reliability* and thus those approaches are assigned to *Reliability*. Since the attribute *Flexibility* forms a synonym for *Adaptability* [70], a classification in *Portability* is made when this attribute occurs. In addition, some approaches claim that they help to reduce *Network Overhead*. Taking into consideration the definitions of *Overhead Time* and *Performance* from [70], this is subordinate to *Performance*, since a reduction in *Network Overhead* improves *Performance*. Another important aspect is that the approaches presented often describe a general improvement in certain attributes or QAs through the migration to a microservice architecture. However, this listing is about the explicit allocation of individual migration approaches. To that extent, these mentions were not included.

As an example, the work by Abdullah et al. [3] can be used to illustrate how the results from the various papers are extracted. Here, an experiment was conducted comparing the *Scalability* and *Performance* on a public cloud infrastructure of a microservice architecture from manual

decomposition of an application and a microservice architecture from automatic decomposition of the same application using the presented approach. It could be shown that “the proposed solution yields a significant improvement of the application performance and scalability compared to the manually created microservices while reducing the overall operational cost” [3]. Based on the statements made therein about the results of the experiment, an improvement of the attributes *Performance*, *Scalability* and *Execution Cost* can be concluded.

Quality Attribute	Subattribute	Reference
<i>Reliability</i>		[16], [73], [60], [90], [135], [133]
	<i>Availability</i>	[47], [83], [9], [135], [129], [133]
	<i>Health Management</i>	[147], [83]
	<i>Recoverability</i>	-
<i>Compatibility</i>		[62]
	<i>Interoperability</i>	[85]
	<i>Co-Existence</i>	[89]
<i>Security</i>		[41], [5], [133]
	<i>Confidentiality</i>	[41], [42]
	<i>Accountability</i>	-
	<i>Authenticity</i>	[133]
	<i>Integrity</i>	[41]
<i>Performance</i>		[16], [47], [44], [11], [12], [28], [9], [152], [35], [34], [90], [108], [3], [8], [63], [104], [117], [86], [133]
	<i>Time Behavior</i>	[47], [60], [108], [135]
	<i>Resource Utilization</i>	[152], [55], [3]
<i>Scalability</i>		[55], [47], [28], [9], [73], [152], [14], [60], [3], [43], [8], [63], [117], [86], [5], [133]
<i>Portability</i>		[55], [83], [67], [63], [74], [133]
	<i>Installability</i>	-
	<i>Adaptability</i>	-
	<i>Replaceability</i>	-
	<i>Deployability</i>	[115], [28], [9], [73], [98], [108], [132], [66], [129], [74], [89]
	<i>Agility</i>	-
<i>Independence</i>		[130], [55], [6], [73], [100], [149], [26], [75], [24], [76], [77], [89]
<i>Maintainability</i>		[16], [17], [43], [110], [133]
	<i>Modifiability</i>	[115], [43]
	<i>Modularity</i>	[47], [44], [11], [12], [9], [48], [26], [35], [34], [96], [75], [51], [63], [50]
	<i>Reusability</i>	[16], [55], [11], [35], [34], [108], [24], [110]
	<i>Testability</i>	[123]
	<i>Analysability</i>	-
	<i>Monitorability</i>	[147], [66], [49]

Quality Attribute	Subattribute	Reference
	<i>Manageability</i>	[42], [46]
	<i>Understandability</i>	[83], [36], [50]
<i>Business</i>		-
	<i>Execution Cost</i>	[54], [60], [3], [24], [133]
	<i>Organizational Alignment</i>	-

Table 4.4: Quality Attributes optimized by Migration Approaches.

In addition to the QAs and subattributes, the SPs introduced earlier could also be detected in the majority of the approaches. As results the properties are assigned to the approaches, by which they were considered or treated in any form. This can be an exclusively neutral consideration of the SPs during the migration process, or the properties are increased or decreased by the process towards the target architecture. The results for this are shown in Table 4.5. Next to the references to the approaches, (+) means an increase, (-) means a decrease, and no symbol means a consideration without statements about directed impacts. In this case, as with QAs, assumptions are not made based on context or statements, and only the specified attributes are included in the results. An exception is made for *Technology Heterogeneity*. According to Alshuqayran et al. [7] the *Technology Heterogeneity* is equivalent to the ability to have the free choice of different programming languages or technologies for the microservice architecture. Accordingly, all approaches to the results of *Technology Heterogeneity* are included as long as they present the mentioned freedom of choice in any form. If the reduction of *Complexity* is assumed, the counterpart to *Complexity*, namely *Simplicity*, is thus also dealt with. This would be equivalent to the reduction of *Complexity*. In none of the approaches, however, statements are made about *Simplicity*.

Based on the results in Table 4.5, it can be seen that a total of 16 migration approaches address the *Granularity* of the resulting microservice architecture, while two cause an increase in *Granularity* and one approach causes a decrease in *Granularity*. In the case of the property of *Coupling*, there are a total of 34 approaches that deal with this. The reduction of the property through the migration process is brought about by 19 approaches. An increase in *Coupling* is not caused by any approach. The remaining references reflect those that make no directional statement of *Coupling*. The *Cohesion* is treated by a total of 33 approaches. Thereby, 19 of them explicitly state that an increase is achieved here, whereas 14 do not make a directional statement. The SP of *Isolation* is not directly considered by any of the approaches. On the other hand, *Autonomy* is examined by five approaches, with two of the approaches directly suggesting an increase in the target architecture. One approach describes an increase in *Complexity*, while another describes the reduction in *Complexity* through migration. Two approaches deal with the *Complexity* during the process without further indication of impacts. The SP of *Technology Heterogeneity* is taken into account by a total of seven migration approaches. In this case there are three non-directional and four increases in property in the resulting architecture.

4.4.2 From Quality Goals to Design Patterns and Best Practices

For the design of a microservice architecture, there are several principles and patterns that can be followed or applied to achieve desired quality goals. For this purpose, it must be possible to map the quality objectives to the individual patterns and best practices. In the following, the collection of the findings is presented in each case, followed by an overview of the results.

System Property	Reference
<i>Granularity</i>	[28], [41], [9], [54], [78] (+), [122], [147], [40], [132], [43], [10] (+), [18], [36] (-), [66], [125], [62]
<i>Coupling</i>	[16] (-), [115] (-), [91] (-), [11], [12], [41] (-), [48] (-), [100] (-), [147] (-), [149] (-), [154], [35] (-), [40], [34], [96], [132], [153] (-), [92], [121] (-), [140] (-), [10] (-), [8] (-), [51], [63] (-), [76] (-), [110], [117], [131], [143] (-), [36] (-), [50], [62], [141] (-), [120]
<i>Cohesion</i>	[16] (+), [130] (+), [91] (+), [46] (+), [11], [12], [41] (+), [48] (+), [78] (+), [147] (+), [45], [26] (+), [35] (+), [40], [34], [132], [153] (+), [92], [111] (+), [121] (+), [10] (+), [8] (+), [76] (+), [110], [117], [131], [143] (+), [18], [36] (+), [50], [62], [141] (+), [120]
<i>Isolation</i>	-
<i>Autonomy</i>	[115] (+), [132], [77], [131], [66] (+)
<i>Complexity</i>	[115] (+), [147] (-), [45], [110]
<i>Technology Heterogeneity</i>	[67] (+), [132], [36] (+), [66] (+), [125], [129], [89] (+)

Table 4.5: System Properties considered by Migration Approaches.

Design Patterns

Similar to the migration approaches, different QAs and subattributes can be assigned to the individual patterns. Accordingly, these QAs are optimized or improved towards the target architecture by applying the patterns. In order to map the QAs, literature reviews or systematic reviews that have already been conducted were used and the results were combined, which map the various architectural patterns to the individual QAs. For this purpose, only the patterns that are already categorized in Section 4.3 are treated. There were differing indications by the literature and overlaps of the indications. Therefore, all mentions per pattern were merged. The result of the mapping is shown in Table 4.6 and contains all references to QAs that are treated in at least one of the reference statements. The results of the overall mapping of microservices architectural patterns to the QAs improved by them are from Li et al. [93], Márquez and Astudillo [99], Richardson [118], Taibi et al. [138], Valdivia et al. [144], Valdivia et al. [145], and Vale et al. [146].

Based on the results, it can be seen that the patterns of microservice architectures mainly optimize *Maintainability*. A total of 33 of the 51 patterns are intended to increase *Maintainability*. Likewise by many patterns the *Reliability* is optimized by the application of the patterns on the architecture. Of the 51 patterns, 23 can improve *Reliability*. *Security* (14 patterns), *Performance* (14 patterns) and *Scalability* (eleven patterns) are in the midfield of the results, while *Compatibility* (five patterns) is further down the list. There is the least optimization on the architecture pattern side with regard to *Portability* and *Independence* within the architecture, with one pattern each improving this.

In the scope of the subattributes it stands out that these find a clearly smaller mention in the literature in general. Most optimization is mapped by seven patterns on the *Availability*, whereas *Monitorability* with three patterns can be assigned to the second most patterns. *Reusability*, *Understandability* and *Testability* are, according to the literature, each improved by one pattern towards the architecture.

All other attributes and subattributes not mentioned are not optimized by any of the patterns.

Pattern	Reference	Quality Attributes & Subattributes
<i>Ambassador</i>	[145, 146]	Reliability, Maintainability, Scalability, Security; Availability, Monitorability
<i>Anti-Corruption Layer</i>	[145, 146]	Performance, Compatibility
<i>API gateway</i>	[99, 138, 144, 145]	Maintainability, Reliability, Security, Compatibility; Availability
<i>Asynchronous completion token</i>	[144, 145]	Reliability
<i>Asynchronous messaging</i>	[118, 145]	Reliability, Performance, Maintainability
<i>Asynchronous query</i>	[144, 145]	Reliability
<i>Auth-service</i>	[93, 144, 145]	Reliability, Security
<i>Backend for frontend</i>	[99, 144, 145, 146]	Compatibility, Performance, Maintainability, Scalability
<i>Change code dependency to service call</i>	[144, 145]	Compatibility
<i>Circuit breaker</i>	[93, 144, 145]	Maintainability, Reliability; Availability
<i>Competing consumers</i>	[144, 145]	Maintainability
<i>Container</i>	[93, 99, 144, 145]	Maintainability, Reliability, Scalability
<i>Correlation ID</i>	[144, 145]	Maintainability
<i>CQRS</i>	[118, 145, 146]	Scalability, Performance, Maintainability, Reliability, Security; Availability
<i>DB is the service</i>	[99, 145]	Maintainability, Reliability, Scalability; Availability
<i>DB per Service</i>	[118, 138]	Scalability, Independence, Security
<i>Deploy cluster and orchestrate containers</i>	[144, 145]	Maintainability, Reliability
<i>Edge server</i>	[144, 145]	Maintainability, Reliability
<i>Enable continuous integration</i>	[99, 145]	Maintainability
<i>Event notification</i>	[144, 145]	Reliability
<i>Event sourcing</i>	[118, 145]	Security, Performance
<i>External load balancer</i>	[144, 145]	Maintainability, Reliability
<i>Externalized configuration</i>	[118, 144, 145, 146]	Security, Maintainability, Scalability; Availability
<i>Gatekeeper</i>	[144, 145]	Security
<i>Gateway Aggregation</i>	[145, 146]	Performance, Maintainability
<i>Gateway Offloading</i>	[145, 146]	Security, Maintainability, Performance, Reliability; Monitorability

Pattern	Reference	Quality Attributes & Subattributes
<i>Health check</i>	[118, 145]	Maintainability
<i>Internal load balancer</i>	[144, 145]	Maintainability, Reliability
<i>Key-Value Store</i>	[144, 145]	Reliability, Security
<i>Load balancer/load-balancing</i>	[93, 144, 145]	Performance, Reliability, Maintainability
<i>Local database proxy</i>	[144, 145]	Maintainability, Reliability
<i>Local sharing-based router</i>	[144, 145]	Maintainability, Performance
<i>Log aggregator</i>	[99, 144, 145]	Maintainability, Performance
<i>Microservice DevOps</i>	[144, 145]	Maintainability
<i>Mulitple service per host</i>	[118, 138]	Scalability, Performance
<i>Page cache</i>	[99, 144, 145]	Performance
<i>Pipes and filters</i>	[144, 145, 146]	Maintainability, Reliability, Performance, Scalability; Reusability
<i>Priority queue</i>	[144, 145]	Maintainability
<i>REST integration</i>	[144, 145]	Maintainability, Compatibility
<i>Result cache</i>	[99, 144, 145]	Performance
<i>Scalable Store</i>	[99, 145]	Maintainability, Reliability, Scalability; Availability
<i>Secure channel</i>	[144, 145]	Security
<i>Self-containment of services</i>	[144, 145]	Maintainability
<i>Service discovery</i>	[144, 145]	Security, Maintainability
<i>Service locator</i>	[144, 145]	Security, Maintainability
<i>Service registry</i>	[93, 138, 144, 145]	Portability, Maintainability, Reliability; Understandability, Availability
<i>Service registry client</i>	[144, 145]	Maintainability, Reliability
<i>Shared DB server</i>	[118, 138]	
<i>Sidecar</i>	[145, 146]	Maintainability, Scalability, Security; Monitorability, Testability
<i>Single Service per host</i>	[118, 138]	
<i>Strangler</i>	[145, 146]	Scalability, Maintainability, Compatibility

Table 4.6: Quality Attributes optimized by Architectural Patterns.

In contrast to the information given in the section on Section 4.4.1 the SPs introduced are treated exclusively by two architectural patterns. In the first case, *Isolation* is improved from the *Single Service per host* pattern, since according to Taibi et al. [138], “the complete isolation of services, reducing the possibility of conflicting resources” is stated as an advantage. In the second case, it is *Coupling*, where the degree is reduced. This reduction is achieved with the pattern called *Anti-Corruption Layer*, as indicated by four out of nine participants in a semi-structured interview by Vale et al. [146].

Best Practices

In the context of best practices, high-level principles or best practices for a microservice architecture were already given as an overview in Section 4.3. However, there are also more specific best practices, which can contribute to certain quality goals if they are followed. First of all, the work of Lichtenthaler and Wirtz [94] could be used as a basis, in which a figure is presented, which maps product factors to QAs. Therefore, these product factors were used as a basis for the best practices and the mapping to them. As an extension of the best practices the work of Pulnil and Senivongse [114] can be referred to, which shows anti-patterns and the corresponding contrary best practices, which are listed as design properties that stand for a good design. Best practices contained therein can be partially validated by the anti-patterns and smells of Schirgi and Brenner [127], since in this case the opposite of these are considered as best practices. From the resulting list and the mappings to the QAs contained therein, an overall view can then be given in Table 4.7.

Best Practice	Reference	Quality Attributes & Subattributes
<i>Access restriction</i>	[94]	<i>Security; Integrity</i>
<i>Account Separation</i>	[94]	<i>Security; Accountability</i>
<i>Acyclic Calls</i>	[114, 127]	<i>Independence, Scalability, Reliability; Understandability, Reusability, Modifiability, Modularity, Deployability</i>
<i>API-based communication</i>	[94]	<i>Maintainability, Compatibility; Testability, Interoperability</i>
<i>Appropriate Service Relationship</i>	[114]	<i>Modularity, Modifiability, Confidentiality, Fault tolerance</i>
<i>Authentication delegation</i>	[94]	<i>Security; Authenticity</i>
<i>Automated infrastructure</i>	[94]	<i>Maintainability, Reliability; Modifiability, Recoverability</i>
<i>Automated monitoring</i>	[94]	<i>Reliability, Maintainability; Recoverability, Analysability</i>
<i>Automated restarts</i>	[94]	<i>Reliability; Recoverability</i>
<i>Autonomous fault handling</i>	[94]	<i>Reliability; Fault tolerance</i>
<i>Built-in autoscaling</i>	[94]	<i>Performance; Capability, Resource Utilization</i>
<i>Cloud vendor abstraction</i>	[94]	<i>Portability; Adaptability</i>
<i>Coarse-Grained Microservices</i>	[114, 127]	<i>Scalability; Modularity, Modifiability, Understandability, Reusability</i>
<i>Communication indirection</i>	[94]	<i>Compatibility, Maintainability; Modularity, Interoperability</i>
<i>Configuration management</i>	[94]	<i>Reliability, Portability; Availability, Adaptability</i>
<i>Cost variability</i>	[94]	<i>Performance; Resource Utilization</i>
<i>Data encryption in transit</i>	[94]	<i>Security; Confidentiality</i>

Best Practice	Reference	Quality Attributes & Subattributes
<i>Dynamic scheduling</i>	[94]	<i>Maintainability, Reliability, Performance; Recoverability, Modifiability, Resource Utilization</i>
<i>Guarded ingress</i>	[94]	<i>Reliability; Availability</i>
<i>Immutable artifacts</i>	[94]	<i>Portability; Replaceability</i>
<i>Infrastructure abstraction</i>	[94]	<i>Portability, Maintainability; Adaptability, Modifiability</i>
<i>Isolated state</i>	[94]	<i>Portability, Performance, Maintainability; Replaceability, Capability, Resource Utilization, Modularity</i>
<i>Loose Coupling</i>	[94]	<i>Maintainability; Modularity</i>
<i>Manageable Connections</i>	[114, 127]	<i>Understandability, Modifiability</i>
<i>Manageable Standards</i>	[114, 127]	<i>Understandability, Modifiability</i>
<i>Non-ESB Microservices</i>	[114]	<i>Scalability; Understandability, Modifiability, Reusability, Fault tolerance</i>
<i>Operation outsourcing</i>	[94]	<i>Maintainability, Performance; Resource Utilization</i>
<i>Persistent Communication</i>	[94]	<i>Reliability; Recoverability, Modularity</i>
<i>Replication</i>	[94]	<i>Performance; Time behavior</i>
<i>Resolved Endpoints</i>	[114, 127]	<i>Scalability; Modifiability</i>
<i>Right Cuts</i>	[114, 127]	<i>Modularity, Modifiability</i>
<i>Seamless upgrades</i>	[94]	<i>Reliability; Availability</i>
<i>Secrets management</i>	[94]	<i>Security; Confidentiality</i>
<i>Separate Persistency</i>	[114, 127]	<i>Modularity, Modifiability, Confidentiality, Fault tolerance</i>
<i>Separation by gateways</i>	[94]	<i>Maintainability, Reliability, Security; Reusability, Availability, Integrity</i>
<i>Seperate Libraries</i>	[114, 127]	<i>Independence; Modularity, Modifiability, Fault tolerance</i>
<i>Service Independence</i>	[94]	<i>Maintainability, Compatibility; Modifiability, Co-Existence</i>
<i>Service-orientation</i>	[94]	<i>Maintainability; Modularity</i>
<i>Sparsity</i>	[94]	<i>Maintainability</i>
<i>Standardization</i>	[94]	<i>Maintainability; Reusability</i>
<i>Standardized deployment unit</i>	[94]	<i>Portability; Installability</i>
<i>Use infrastructure as code</i>	[94]	<i>Portability, Maintainability; Installability, Modifiability</i>
<i>Versioned APIs</i>	[114]	<i>Understandability, Modifiability</i>

Table 4.7: Quality Attributes optimized by Best Practices.

Based on the results, it is apparent that *Maintainability* (15 best practices) and the subordinate subattributes with *Modifiability* (16 best practices) and *Modularity* (eleven best practices) in particular can be improved by most best practices towards the target architecture. Likewise, *Reliability* (eleven best practices) is optimized by several best practices. *Security* (six best practices), *Performance* (six best practices), *Portability* (seven best practices) and the associated subattributes are in the midfield. The least number of best practices improve *Compatibility* (three best practices), *Scalability* (four best practices) and *Independence* with two best practices. The newly introduced attribute *Business* and its subordinate attributes are explicitly not mentioned in any context with a best practice. The subattributes *Agility*, *Monitorability* and *Manageability* are also not mentioned in this context.

For the SPs, only the *Coupling* and *Complexity* properties of best practices are treated. The information on *Complexity* in the work of Lichtenthaler and Wirtz [94] results from the mapping of best practices to the property of simplicity. This means that best practices mapped to simplicity in this work result in a reduction of the *Complexity* of the architecture.

For the best practice named *Acyclic Calls*, there is an influence in the form that the system has “lower coupling and lower complexity” [114] due to the usage. In the case of *Manageable Connections*, the effect is that this does not make the system too complex. The same is true for using the practice of *Non-ESB Microservices*, whereby not having an ESB Microservices leads to the reduction in *Complexity* [114]. According to Lichtenthaler and Wirtz [94], the *Persistent Communication* and the *Standardized deployment unit* are mapped to the simplicity attribute and thus have positive effects in reducing the *Complexity* of the system.

4.4.3 Patterns Trade-Offs

In addition to the advantages that architectural patterns can entail, there are also trade-offs or pains. Here, the patterns not only have positive effects on the QAs of an architecture, but can also have negative effects on the qualities and SPs of the architecture through their application. To cover these trade-offs, the works [146] and [138] are referred to. Thereby, the work of Vale et al. [146] explicitly deals with the trade-offs of patterns in microservice architecture. For this purpose, a semi-structured interview was conducted here, whereby the effects of the applied patterns on the QAs within an architecture are investigated. In the work of Taibi et al. [138] a systematic mapping study is presented, which first identifies the patterns and then establishes the advantages and disadvantages in each case.

For the QAs and their subattributes, the systematic mapping study by Taibi et al. [138] results in a trade-off for the *Service registry* pattern with respect to *Reusability*. The patterns *Shared DB server*, *Single Service per host* and *API Gateway* have a trade-off of the QA *Scalability*. Furthermore, the *Single Service per host* pattern has an additional trade-off of *Performance* due to the application. The results of the interview of Vale et al. [146] show that when the *Pipes and filters* pattern is applied, *Monitorability* emerges as a drawback. For *Gateway Aggregation*, *Scalability* and *Fault tolerance* are listed as trade-offs, as this may cause a single point of failure to be present. The same problem can be caused by the *Externalized configuration*, whereby *Fault tolerance* can also be a disadvantage. According to the interview participants, *Performance* is affected by the use of the *Anti-Corruption Layer* pattern. The use of the *Strangler* pattern can cause trade-offs in terms of *Maintainability* and *Compatibility* (backwards) according to the results of the interviews.

In the context of SPs, there are also drawbacks that can occur when the patterns are applied to the architecture. Accordingly, the *Complexity* of the architecture is undesirably increased by the *Service registry* and *API gateway* patterns [138]. *Isolation* may not be present to a high degree by applying shared *DB server* [138]. According to Vale et al. [146], in the area of SPs, it is mainly *Complexity* that is undesirably increased by certain patterns, respectively. These include *Pipes and filter*, *Ambassador*, *Sidecar*, *CQRS*, *Anti-corruption layer*, *Externalized configuration*, and *Backend for frontend*. Only the architecture pattern of *Gateway Aggregation* has a negative impact on the *Coupling* of the architecture since the degree of this attribute is increased as a result.

In summary, this results in the list of trade-offs found for the architectural patterns discussed in this thesis, which are illustrated in Table 4.8.

Pattern	QA trade-offs	SP trade-offs
<i>Ambassador</i>		<i>Complexity (+)</i> [146]
<i>Anti-Corruption Layer</i>	<i>Performance</i> [146]	<i>Complexity (+)</i> [146]
<i>API gateway</i>	<i>Scalability</i> [138]	<i>Complexity (+)</i> [138]
<i>Backend for frontend</i>		<i>Complexity (+)</i> [146]
<i>CQRS</i>		<i>Complexity (+)</i> [146]
<i>Externalized configuration</i>	<i>Fault tolerance</i> [146]	<i>Complexity (+)</i> [146]
<i>Gateway Aggregation</i>	<i>Scalability; Fault tolerance</i> [146]	<i>Coupling (+)</i> [146]
<i>Pipes and filters</i>	<i>Monitorability</i> [146]	<i>Complexity (+)</i> [146]
<i>Service registry</i>	<i>Reusability</i> [138]	<i>Complexity (+)</i> [138]
<i>Shared DB server</i>	<i>Scalability</i> [138]	<i>Isolation (-)</i> [138]
<i>Sidecar</i>		<i>Complexity (+)</i> [146]
<i>Single Service per host</i>	<i>Scalability, Performance</i> [138]	
<i>Strangler</i>	<i>Maintainability, Compatibility</i> [146]	

Table 4.8: Architectural Patterns Trade-offs Overview.

4.5 Quality Model

For the first and second research question, efforts have already been made to collect extensive results on how QAs can be incorporated into the migration process in order to propose suitable decomposition methods, patterns and best practices towards the target architecture. The third research question deals with how the existing tool ‘Architecture Refactoring Helper’ can be extended with a quality model that implements exactly these results and makes them accessible to users of the tool. For this purpose, the individual previous results must first be combined in order to create a model on the basis of them, which depicts the relations of QAs and other influences with regard to migration approaches, patterns and best practices.

4.5.1 Design and Overview

To provide a holistic overview of the individual components that address the quality aspect in the migration from a monolithic architecture to a microservice architecture, these components must be composed in a structured manner. In doing so, the intermediate steps and dependencies between the various components should be considered and linked in a meaningful way so that they become apparent and result in an overall model. In addition to the relationships to each other, the model can also represent a possible sequence or process of the migration process in terms of quality.

For the development of the model, which is roughly outlined in Figure 4.5, a top-down approach is initially chosen. In this approach, the QAs and SPs (from Section 4.1.2) are at the top level of the model after a Quality Assessment and act as a starting point, since they are intended to support the selection of the appropriate strategy and architecture. With the help of the QAs and the SPs, suitable migration approaches can then be assigned with the mapping from Section 4.4.1. On the other hand, the same results from the Quality Assessment level can be used to identify suitable patterns and/or best practices in the Architecture level with the help of the mapping from Section 4.4.2. For this purpose, the model can be used to make the appropriate choice in the level of Strategy finding and Architecture design, one after the other or independently of each other. The model thus results from the composition of the previous results of this section.

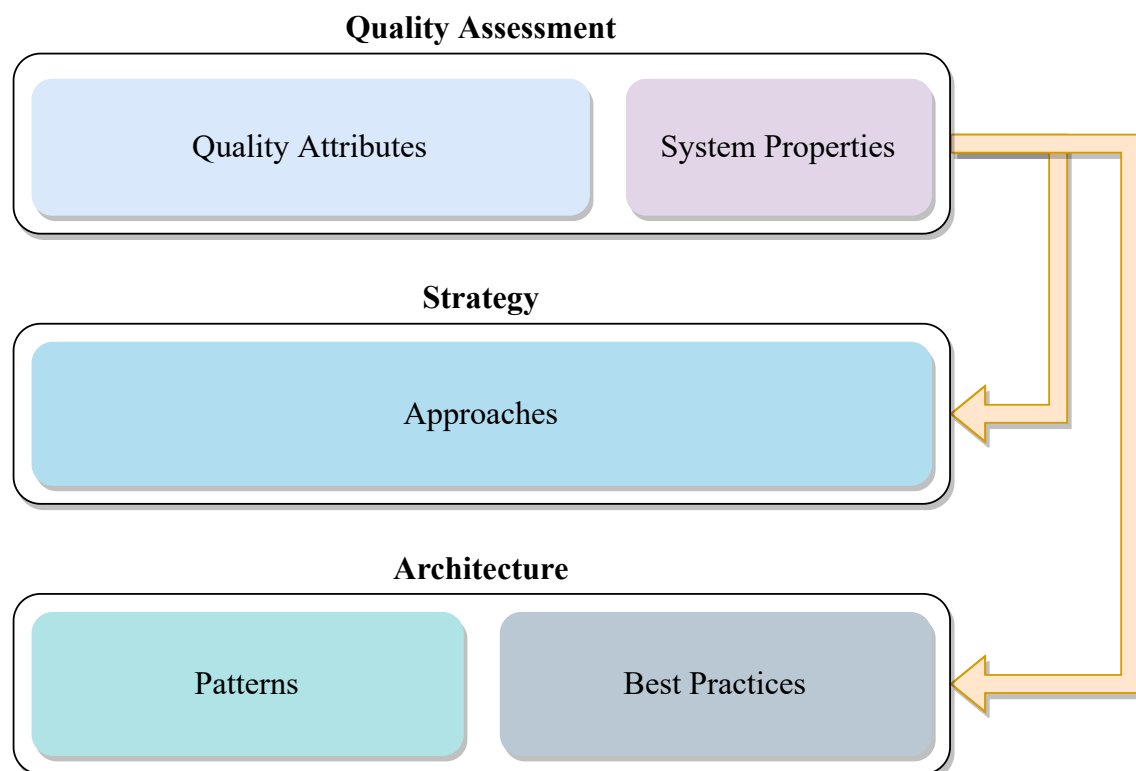


Figure 4.5: Quality Model Overview.

4.5.2 Model in Detail

In order to get a detailed view of the overall model, it must be divided into two parts. On the one hand, this results in a detailed view of the relationships between the components of the Quality Assessment and the Strategy level (Figure 4.6) and, on the other hand, the relationships to the architecture design level (Figure 4.7). The reason for this separation is that otherwise there would be too many visual overlaps between the relationships. For better readability and comprehensibility, the detailed view contains placeholders for exemplary components. Thus, an exemplary model is created without actual mapping - this mapping emerges from the results of Section 4.4.

Since the components of the Quality Assessment Level are the same for both detailed views, they will first be explained in detail. All QAs and associated subattributes are summarized under the group of QAs. The subattributes are always grouped to a superordinate QA, but can have independent relationships to other subattributes and entire QA groups. To this end, the QAs and subattributes may have potential trade-offs to other attributes (according to Section 4.1.4) that must be considered in this context. SPs represent another group in the same level. Here, there are no subattributes or identified trade-offs among them. There is a mutual relationship between the mentioned QAs/subattributes and the SPs. According to Section 4.1.3 it must be noted in this context that increasing (+) or decreasing (-) the degree of the respective SPs can reduce (↓) or improve (↑) a QA. On the other hand, a QA can be fulfilled or not fulfilled, provided that a certain degree of the SPs is present. This relationship to each other must be considered in this level and thus these groups are not independent of each other in the model. The goal of this level of the model is to select one or more QAs and SPs required for the target architecture, which are needed in the further process, taking into account the relationships to each other and the potential trade-offs.

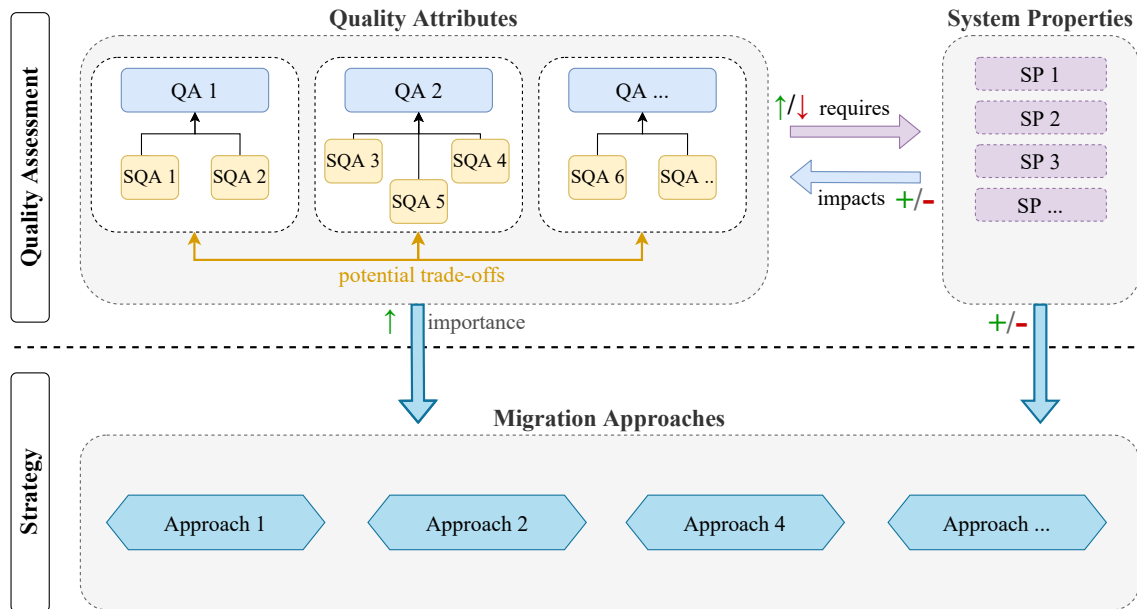


Figure 4.6: Quality Model on Migration Strategy Level.

After the components of the Quality Assessment level have been declared, an explanation of the components of the Strategy level and the relationships can be given. The group of migration approaches contains all migration approaches treated in the context of this work and the QAs

and SPs mapped to them, as stated in Section 4.4.1. Accordingly, with the help of the selection of the QAs, an improvement of these can be achieved by the usage of suitable approaches. Thereby, there is a weighting or an importance for each QA and subattribute, since in the selection of the QAs, one attribute may be of greater importance for the target architecture than another. For this purpose, an importance scale similar to the ATAM can be used. This would imply that the importance factors are chosen as follows:

- Importance A - Factor 3
- Importance B - Factor 2
- Importance C - Factor 1

When choosing the SPs, not only the property itself is mapped to the approaches, but additionally the degree to it, which means more precisely whether the degree of the SP should be increased or decreased by the migration approach. The combination of both groups results in a set of suitable migration approaches.

For the detailed view of the model in the context of the Architecture level, the basis of the relations to the level of the Quality Assessment is the same as for the migration approaches. Nevertheless, there are differences in the further selection procedure of the model. Namely, the selection by QAs and SPs does not target only one group, but maps to the design patterns and the best practices, which are identified in Section 4.4.2, and in which qualities and properties can be assigned. Another difference is that in the case of the patterns, in addition to the trade-offs between the QAs, there are trade-offs that occur as a result of the application of the patterns. This means, that for example, QA 1 and QA 2 are optimized by pattern 1, but QA 3 cannot be fulfilled by the application of the pattern because there is a trade-off between pattern 1 and QA 3.

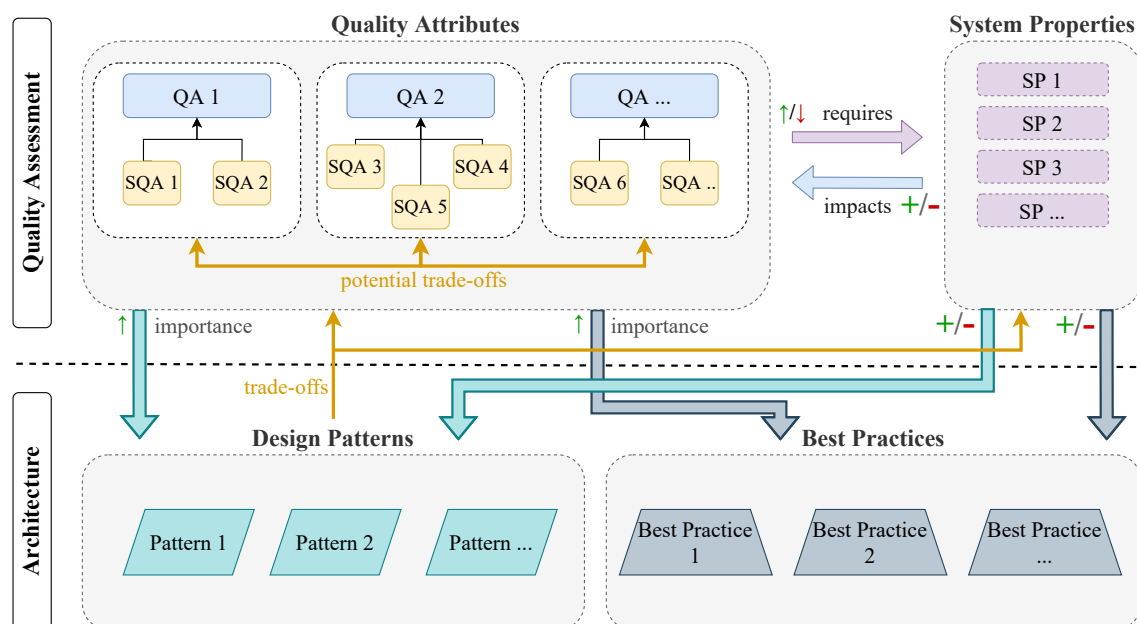


Figure 4.7: Quality Model on Architectural Design Level.

4.5.3 Quality Model acting as Guide

To better illustrate the background to the idea of the model, the role of the model as a guide in the migration process is explained as an example. Therefore, two sample scenarios and the respective steps are explained, whereby recommendations for strategies and architectural patterns can be given through the quality model. For this purpose, inputs to the model are provided and outputs resulting from the model are presented.

A simple case is a scenario in which *Security* should be optimized by migrating to microservice architecture, which results from the Quality Assessment. *Security* is in this case of paramount importance. Therefore, the attribute *Security* with the importance A is selected from the QAs. Likewise, the *Coupling* should be as low as possible and the *Cohesion* as a result should be particularly high. This results in the following inputs to the model:

- **QAs:** *Security (Importance: A)*
- **SPs:** *Coupling (-), Cohesion (+)*

In this case, according to the mappings of the model, no effects of the SPs on the QAs need to be considered. In addition, no trade-offs and weights need to be included since only one attribute is chosen.

From the choice of attributes and properties, in the strategy domain, the closest match to the migration approach by Daoud et al. [41] is obtained, since it optimizes *Security* after the mapping and also reduces *Coupling* and increases *Cohesion*.

In the architecture domain, some candidates emerge on the patterns side due to the *Security* attribute. The pattern *Secure Channel* can be mentioned here as an example. Another example would be the *Anti-corruption layer* pattern, which could be considered, but the trade-offs of *Performance* impairment and increased *Complexity* in the architecture would also have to be addressed in the further course. There are also some best practices that can contribute to *Security*, including the best practices of *Access restriction* and *Data encryption in transit*. This results in the following sample outputs:

- **Strategy:** [41], ...
- **Patterns:** *Secure Channel*, ...
- **Best practices:** *Data encryption in transit*, ...

A more complex example would represent a scenario in which *Performance* towards the microservice target architecture has a very high importance, while *Maintainability* and *Availability* should also be considered and have a medium importance. In addition, the microservices should have the lowest possible degree of *Coupling*. However, there is a potential trade-off between the QA *Maintainability* and *Availability* (according to Section 4.1.4). This must be considered during the further course, whereby the requirements may have to be adapted to the target architecture. Likewise, according to Section 4.1.3 a low *Coupling* leads to an increase in *Maintainability*. In this respect, the optimization of the attribute *Maintainability* is further reinforced. This results in the following inputs to the model:

- **QAs:** *Performance (Importance: A), Maintainability (Importance: B), Availability (Importance: B)*
- **SPs:** *Coupling (-)*
- **Trade-offs:** *Maintainability ↔ Availability*
- **Impacts:** *Coupling (-) → Maintainability (+)*

In order to be able to give suitable recommendations in the strategy and in the architecture domain, the weightings have to be taken into account. For the strategy, the weightings result in two promising recommendations. Overall, the approach by Bajaj et al. [16] fulfills 3/3 of the selected attributes and deals with 1/1 SPs. Provided the weighting is included, the following calculation results in an overall score:

$$3(\text{Performance}[A]) + 2(\text{Maintainability}[B]) + 2(\text{Availability}[B]) + 1(\text{Coupling}(-)) = 8$$

Since all criteria match, the result is a suitability of $8/8 = 100\%$.

Another approach that can be recommended, namely that of Alwis et al. [8], results in a suitability of $6/8 = 75\%$ after the same calculation. Since this approach does not fulfill the attribute *Maintainability*, which is evaluated with an importance of B (factor 2). According to the same principle, the pattern *Backend for frontend* can be recommended in the architecture, for example, with a suitability of $5/8 = 62.5\%$. In this case, *Availability* and *Coupling* are not covered by the pattern. For the best practices, *Dynamic scheduling* can be proposed as an example, which fulfills all QAs, except for the SP coupling, and thus achieves a suitability value of $7/8 = 87.5\%$. This results in the following sample outputs:

- **Strategy:** [16], [8], ...
- **Patterns:** *Backend for frontend*, ...
- **Best practices:** *Dynamic Scheduling*, ...

It should be noted that in the case of the *Backend for frontend* pattern, the application of this pattern increases the *Complexity* of the architecture (see Table 4.8).

Thus, the quality model can be used as a guide for the migration from monoliths to microservices, acting as a process and suggesting appropriate decomposition approaches and architectural design decisions based on QAs and SPs.

5 Implementation

The third research question *RQ 3* of this work deals with how the ‘Architecture Refactoring Helper’ tool [65] can be extended by the previous findings with respect to the quality aspects and the resulting quality model. This can be answered by considering how the individual steps of the implementation were accomplished. The implementation realizes an extension to the prerequisites and the existing functionalities of the tool and ensures that the implemented quality model can act as a guide for the user. For this purpose, the given prerequisites from the implementation of Haller [64] are first explained, and the further development process based on established requirements is described in detail. Subsequently, an evaluation is performed to examine the result of the implementation with regard to the usability and usefulness of the integrated quality model.

5.1 Prerequisites

In advance, some prerequisites regarding the technology stack and the deployment were already given for the extension. These result from the original implementation of Haller [64], since in the context of this work an extension is realized, which builds therefore on the used technologies and prerequisites.

Starting in the backend, the object-oriented programming language C# and the platform .NET 6 are used for the implementation. In general, the backend provides a Representational State Transfer (REST) API, which is adapted to further endpoints in the context of this work. In addition, a database and a Database Management System (DBMS) are required in the background for storing the data. A SQLite database was chosen, because it is particularly lightweight and well suited for many-to-many relationships due to its nature as a relational database [64]. For the decision of a suitable Object-relational mapping (ORM) between the database and the C# objects of the backend, the choice was made to use the Entity Framework Core (EF Core) from Microsoft for this purpose. With the help of Swashbuckle, which provides swagger support, an OpenAPI specification can be generated, which is stored in a JavaScript Object Notation (JSON) file. It can also be used to test the different provided interfaces of the API via a user interface (SwaggerUI) and to display them as an overview. The generated JSON file is especially important for the connection of the frontend to the backend, since through the conform OpenAPI specification to the backend suitable endpoints and models in the frontend can be generated automatically in the further process.

For the frontend on the other hand, Angular (version 13) was chosen among the known frontend frameworks. This decision was made mainly by personal preferences and experience [64]. It must be said that Angular has a clear separation of templates (Hypertext Markup Language (HTML)), styling (Sassy Cascading Style Sheets (SCSS)) and controllers (Typescript) for each component, which allows a straightforward structuring of the application. So that the controls used in the frontend remain consistent, the components of Angular Material from Google LLC were used.

This is a library of pre-built components with frequently requested functions, which can be easily integrated into an Angular application. To be able to design the layout of the individual components, an API called Angular Flex-Layout was used. As already mentioned with the backend, the required models and services can be generated with the help of the JSON file conforming to the OpenAPI specification by the Node Package Manager (NPM) module ng-openapi-gen.

Since the tool will be hosted after the extension, building and deploying the application is essential. It was therefore decided that this should be implemented using a Docker architecture in combination with an NGINX reverse proxy. For this purpose, the backend and the frontend each have their own Dockerfile, so that Docker containers can be created. The three containers of the defined services are then all running in a multi-container application using the Docker Compose tool.

The given prerequisites have enabled an implementation without introducing any new technologies. This means that only named technologies and libraries had to be included to ensure functionality. In addition, a hosting without customization was made possible by the predefined build and deployment from Haller [64].

5.2 Requirements

The requirements for the extension of the tool were developed in several sessions of discussions [59] with Fritzsich, the designer of the 'Architectural Refactoring Framework'. These were continuously adjusted until the final requirements were created. In this respect, the requirements are results of possible use cases that a user would like to perform in the tool in order to be guided in the migration process in terms of quality aspects. It was also considered during the creation of the requirements that the results of the quality model should be included in the implementation. Based on this, there are the following requirements for the implementation in the backend and frontend. Thereby there are subordinate requirements to a superordinate one. If all subordinate requirements are fulfilled, the superordinate requirement is also fulfilled.

5.2.1 Backend Requirements

For the backend, there are a number of requirements that have to be fulfilled so that the tool can be extended to include the quality model. One requirement that extends across all requirements is that the models and functionality in the backend should be designed to be as extensible as possible, so that they can be adapted without great effort. In addition, the structure and style of the original implementation of the tool should be continued. Furthermore, the following specific requirements for the backend could be established:

Req. Backend-1

QAs and SPs should be created in the database and accessible via API.

Req. Backend-1.1: QAs, subattributes and SPs should be created in the database with relationship to each other. Subattributes are subordinated to QAs.

Req. Backend-1.2: Data configuration of quality QAs, subattributes and SPs. At the initial start of the application, the attributes and properties of the quality model should be stored in the database.

Req. Backend-1.3: Create, Read, Update and Delete (CRUD) methods for QAs, subattributes and SPs with API. This allows them to be read, created, edited or deleted in the database.

Req. Backend-2

All models required by the ATAM for scenario-based mode should be created in the database and accessible via API. All functions needed for this like add/update/delete shall be implemented in the backend.

Req. Backend-2.1: Scenario model should be created. Model should be based on ATAM with the fields name, description, importance and difficulty.

Req. Backend-2.2: QAs should be assignable and editable to scenarios. Scenarios are related to a list of QAs and/or subattributes that are important in the context of the scenarios.

Req. Backend-2.3: CRUD methods for scenarios and relationships with API. This allows them to be read, created, edited or deleted in the database.

Req. Backend-3

QAs/subattributes and SPs should be able to be mapped to migration approaches/patterns/best practices.

Req. Backend-3.1: In addition to the existing models and data for the migration approaches, models for patterns and best practices should be added, and the known data for these should be imported into the database.

Req. Backend-3.2: Known QAs and SPs of the model should already be added with the approaches/patterns/best practices via the data seeding method at the initial start of the application.

Req. Backend-4

The recommendation service shall be extended to include the quality model so that recommendations for migration approaches/patterns/best practices can be given based on the input information.

Req. Backend-4.1: QAs/subattributes and SPs shall be included in the calculation of suitability so that these are involved as matches, mismatches, and neutrals.

Req. Backend-4.2: Weighted Suitability should be able to be calculated, insofar that importance of scenarios are transferred to the weight of QAs in scenario-based mode.

5.2.2 Frontend Requirements

In addition to the general requirements for the backend, there are also requirements that should be adhered to throughout the implementation of the frontend. It is important that the user interface is designed consistently, which means that the components of the application are implemented in the same presentation form, resulting in an overall uniform appearance. In addition, the navigation in each component should be arranged in such a way that a simple navigation between the components is possible and if necessary, steps can be skipped. In addition to the general requirements, there are specific requirements for the frontend, which are listed below:

Req. Frontend-1

In phase one, it should be possible to create scenarios in the third step of the phase, to which QAs are assigned. All control elements should be designed to be intuitively usable.

Req. Frontend-1.1: Scenarios should be able to be created with the fields name, description, importance and difficulty. Furthermore, it should be possible to edit and delete them.

Req. Frontend-1.2: It should be possible to assign QAs to the selected scenario. Also subattributes should be selectable. The selection should be adjustable at any time.

Req. Frontend-2

In phase two, it should be possible to decide between the existing manual configuration and the scenario-based search for suitable migration approaches.

Req. Frontend-2.1: The manual configuration of the inputs shall be restructured with respect to the quality model, so that the attributes and properties are added and selectable.

Req. Frontend-2.2: In the scenario-based mode the QAs of the scenarios shall be transferred and a more detailed configuration with additional already existing input options shall be possible.

Req. Frontend-3

In phase two, after the configuration, the recommendations for suitable migration approaches shall be presented in an overview.

Req. Frontend-3.1: In addition to the already existing view, further columns shall be available, which shall display the suitability exclusively in relation to the QAs and SPs.

Req. Frontend-3.2: In the detailed view of the individual approaches, it shall be evident which quality aspects are fulfilled by the respective approach.

Req. Frontend-4

In phase three, it should be possible to decide between the existing manual configuration and the scenario-based search for suitable patterns/best practices. For this purpose, the configuration from phase two is to be adopted and thus refers to **Req. Frontend-2**.

Req. Frontend-5

In phase three, after the configuration, the recommendations for suitable patterns/best practices are to be displayed in an overview. For this purpose, **Req. Frontend-3** is referenced, since the same functionalities should be implemented only for patterns/best practices instead of the migration approaches.

Req. Frontend-6

During the configuration for phase two and phase three and during the presentation of the recommendation results, an overview of all inputs made shall be given.

5.3 Development Process

The extension of the tool was developed exclusively by one person, who used the Git system to manage the individual steps of the process. A public repository on GitHub was used for this purpose [57], which was a fork of the original tool created by Haller [65].

To track progress and ensure the implementation of the quality model, three milestones were created, each corresponding to one of the three phases of the ‘Architecture Refactoring Helper’ tool. All issues related to these phases were assigned to the appropriate milestone, allowing for clear tracking of progress.

Each issue represented a specific feature of the tool, and included the implementation of both the frontend and backend requirements. This was necessary because the models and endpoints of the API were often related. In some cases, an issue was divided into subissues, which were resolved collectively to fulfill the parent issue. To allow for independent implementation of each issue, a new branch was created for each parent issue. Once an issue was successfully resolved, the branch was merged into the main branch.

Throughout the development process, issues were added that were not part of the original requirements but were identified as potential enhancements or additions to the tool. These issues were tagged with the ‘wontfix’ label, indicating that they were not part of the current development scope but could be addressed in future versions of the tool.

5.4 Backend Design

The established requirements for the backend from Section 5.2.1 require the implementation of the needed functionality to fulfill them. In this regard, the implementation steps deemed most important for an understanding can be discussed and illustrated with actual examples from the code.

Quality Attributes and System Properties

To fulfill **Req. Backend-1** of the backend, the data models of the QAs/subattributes and the SPs must be integrated into the database. First of all, it must be noted that the initial situation of the tool provided that qualities can be requirements on the one hand and metrics on the other hand, which could be distinguished by an enum with the help of the field `category`. In order to adapt this subdivision to the model, the enum value 'Requirement' was renamed to 'Attribute' and the value 'Metric' to 'SystemProperty', but the structure was kept in order to not negatively affect the functions of the tool. Since, according to the quality model, there are associated subattributes in addition to the QAs, a new model `QualitySublevel` was added. At this point, the model contains the same fields as a quality, but without the classification of the category. Here, the assignment of the relationship to each other is crucial. The relationship is realized in such a way that a quality can have none, one or more subattributes. For this, an `ICollection<QualitySublevel>?` must be present in the quality model, and a quality object and the name of the parent quality must be present in the `QualitySublevel` (the subattributes) as foreign key, so that these can be assigned. This allows navigation in both directions afterwards - i.e. from QA to all subattributes as well as from the subattribute to the parent QA. In the database configuration for the qualities, this relationship can then be defined as illustrated in Listing 5.1. The configuration is applied when the database is created and thus the actual data of the model can be put into the database and relationships can be defined.

Listing 5.1 Configuration of Relationship of Quality Attributes and Subattributes (C#).

```
builder
    .HasMany(q => q.QualitySublevels)
    .WithOne(qu => qu.Quality)
    .HasForeignKey(q => q.QualityName);
```

The configuration of the QAs and the SPs can be shown exemplary in Listing 5.2, where the distinction of the categories becomes clear. A similar configuration is given to the subattribute, in which the foreign key to the QA is additionally set by the quality name. Thus, if the attributes and subattributes would change in the future, a slight adaptation of the quality model is possible through the configuration. Moreover, it is possible to create subproperties of the SPs, which are currently not yet provided.

It must be paid attention here that if data models are changed, added or removed, and/or that entities or characteristics are adapted, then an EF Core migration must be added, so that the data base schemata and the configuration data matches with those of the application.

After the models and configurations have been implemented or customized, the CRUD methods for the `QualitySublevel` objects have been added analogous to those of the `Quality` objects. Furthermore, the API was adapted so that the `QualitySublevel` objects are included in the queries of the `Quality` objects and are thus added as a child element of these in order to read them.

Listing 5.2 Configuration of Quality Attributes and System Properties (C#).

```
builder.ToTable(Constants.TableNameApproachProcessQuality);
builder.Property(q => q.Category)
    .HasConversion < string > ();

builder.HasData(
    new Quality {
        Name = "Reliability",
        Description = "...",
        Category = QualityCategory.Attribute
    },
    ...
    new Quality {
        Name = "Cohesion",
        Description = "...",
        Category = QualityCategory.SystemProperty
    }
    ...
);
```

Scenarios

Similar to the QAs and SPs, the data models of the scenarios are created first. In addition to a unique ID, the data model from Listing 5.3 contains the fields name, description, difficulty and importance. Likewise, each scenario can be related to QAs and subattributes assigned to the respective scenario. Due to the design of the model, it is possible to assign subattributes without a parent QA, which provides more flexibility when the assignment needs to be very specific. However, it is also possible that a scenario has no relationships to named attributes. A scenario requires the name field in addition to the key - all other fields are optional. The configuration of the scenarios is similar to the one from Listing 5.1 except that in this case no manual foreign key is required, since the attributes are not assigned via the configuration but are assigned only by creating or changing the relationships.

For the CRUD operations a new service `ScenarioService` and the corresponding controller `ScenarioController` were created. The controller provides the interface and paths of the API and uses the associated methods of the service that directly access the database context. An example can be given on the basis of the Listing 5.4 from the controller, with which it is possible over a GET request to receive all QAs of a scenario. In this case, only the ID of the scenario is required and passed in the request. The controller then calls the corresponding methods of the service.

In the service, the method `GetQualities` then returns all qualities of a scenario (see Listing 5.5). The GET/PUT/POST/DELETE requests can subsequently be made via the controller, which can be read/changed/created/deleted in the database with the help of the methods from the service. In this

Listing 5.3 Data Model of a Scenario (C#).

```

Table(Constants.TableNameProjectScenario)]
public class Scenario
{
    [Key]
    [JsonPropertyName("scenarioId")]
    public int ScenarioId { get; set; }
    [Required]
    [JsonPropertyName("name")]
    public string Name { get; set; }
    [JsonPropertyName("description")]
    public string? Description { get; set; }
    [JsonPropertyName("difficulty")]
    [JsonConverter(typeof(JsonStringEnumConverter))]
    public RatingLevel? Difficulty { get; set; }
    [JsonPropertyName("importance")]
    [JsonConverter(typeof(JsonStringEnumConverter))]
    public RatingLevel? Importance { get; set; }
    [JsonPropertyName("qualities")]
    public ICollection<Quality>? Qualities { get; set; }
    [JsonPropertyName("qualitySublevels")]
    public ICollection<QualitySublevel>? QualitySublevels { get; set; }
    ...
}

```

Listing 5.4 Get all Qualities of a Scenario via Controller (C#).

```

[HttpGet("{id:int}/" + Constants.ApiSubPathQualities, Name = "GetQualities")]
public ActionResult<IEnumerable<Scenario>> GetQualities(int id)
{
    return Ok(_scenarioService.GetQualities(id));
}

```

process, the relationships of the QAs and subattributes are added when they are created. When editing, not only the fields but also the relationships are updated by removing them beforehand and then adding the changed relationships again. When deleting, only the scenario itself and the relations are removed. When adding relations of attributes to scenarios, it is important that the qualities for the operation are attached, since these entities already exist and only relations are changed. The same is true for changing the relations. However, before adding them to the scenario, the previous entities of the qualities are set to the `EntityState.Detached` state, which means that they are no longer tracked. In this regard, **Req. Backend-2** and all associated requirements could be fulfilled.

Listing 5.5 Get all Qualities of a Scenario via Service Method (C#).

```
public IEnumerable<Quality> GetQualities(int scenarioId)
{
    var db = new RefactoringApproachContext();
    var query = db.Scenarios
        .Where(e => e.ScenarioId == scenarioId).Select(e => e.Qualities).ToList();
    var result = query.FirstOrDefault();
    ...
    return result;
}
```

Mapping of Attributes and Properties to Approaches/Patterns/Best Practices

Data seeding is already anchored in the basic functions of the tool. Using a JSON with predefined or pre-exported data sets, it is possible to insert data and relationships to each other directly into the database when the application is initially started or when the database is created. In this respect, QAs (or requirements) and SPs (or metrics) could already be assigned to migration approaches before the tool was extended. In addition to the already existing migration approaches, however, there are still patterns and best practices for which models should be created and initial data inserted. In this respect, a new model `ArchitecturalDesign` was added analogously, which is decisive above all for the relationships and navigation and in which a distinction can be made between pattern and best practice with the aid of an enum. In addition, analogous to the model of the `ApproachSource`, the `ArchitecturalDesignSource` was added, which then contains the actual data, such as name and description, that can then be uniquely assigned to an architectural design entity. The division was chosen to remain consistent with the models of the migration approaches. The existing controller and service for migration approaches were used as a basis, so that these can also be implemented for the architectural design entities after adjustments. This enables CRUD methods and the addition of relationships to, for example, the QAs and results in the functionality for the architectural designs and migration approaches services being almost identical. An example of a pattern within the JSON to be seeded can be seen in Listing 5.6. For better understanding, it is necessary to mention that the initial situation of the tool consists in the fact that the relations of the approaches are divided into different categories. The QAs are thereby located in the ‘ApproachProcess’ category, which is continued in the same way for the patterns and best practices to prevent duplicated functionality.

This allows the QAs, subattributes, and SPs identified in this work to be added to the migration approaches and architectural design patterns and best practices. The data seeder reads the JSON and calls the create or add method of the respective service for each approach or pattern/best practice, if they do not already exist in the database. In this respect, **Req. Backend-3** could be fulfilled with all aspects.

Recommendation Service

For the migration approaches, a recommendation service was already available in the tool, which performs the calculations of the configuration inputs and generates suitable proposals for migration approaches by assigning a suitability. The procedure can be explained roughly in the sense that first an `ApproachRecommendationRequest` is received, which contains all necessary information. This means

Listing 5.6 Pattern Example for JSON Data Seeding.

```

{
  "architecturalDesignId":15,
  "identifier":"15",
  "architecturalDesignSource":{
    "architecturalDesignId":15,
    "name":"CQRS",
    "description":"...",
    "source":"Microservices.io by Chris Richardson",
    "link":"https://microservices.io/patterns/data/cqrs.html"
  },
  "category":"Pattern",
  "approachProcess":{
    "qualities":[
      {
        "name":"Scalability",
        "category":"Attribute"
      },
      ...
    ],
    "qualitySublevels":[
      {
        "name":"Availability",
        "qualityName":"Reliability"
      }
    ]
  }
}

```

in particular that the information is contained about which attributes should be included or excluded in the calculation. Neutral attributes are not included. The service `SimpleRecommendationService`, which implements the interface `IRecommendationService`, processes this request by establishing the matches and mismatches of the attributes of the request with the migration approaches. Thus it performs the evaluation of the attributes for each approach and also assigns an overall suitability value to this approach based on the request.

In order to now integrate the quality model established in the context of this work, adjustments and extensions had to be made here. First, the possibility had to be given that subattributes are also included in the evaluation. In this respect, the implementation was carried out analogously to the other attributes. However, there is a special feature with the subattributes, which was not available for any attribute in the former `SimpleRecommendationService`. Namely, a subattribute should also match with a migration approach, provided that the father QA is matched. Accordingly, if a match to a QA occurs, all subattributes are potentially assigned as a match if they can be mapped to the migration approach. For the quality model, the next step is that there is a separate suitability, which is specified based on the number of matches in relation to the specified QAs and subattributes. The same is true for the SPs, which are given their own suitability. Since the suitability in the quality domain is given by the number of matches, the total suitability for all attributes was no longer given as a percentage, but was also changed to the relation of

Algorithm 5.1 Weighted Quality and System Property Score Calculation.

```

function CALCULATETOTALWEIGHTEDSCORE(recommendation, request, weightedMatched)
  totalWeightedScore ← 0
  totalWeight ← 0
  for all quality in request.qualityInformation do
    if quality.category == Attribute and quality.selection == Include then
      totalWeight ← totalWeight + getTotalWeight(quality)
    end if
  end for
  for all qualitySub in request.qualitySublevelInformation do
    if qualitySub.selection == Include then
      totalWeight ← totalWeight + getTotalWeight(qualitySub)
    end if
  end for
  totalWeightedScore ← ROUND( $\frac{\text{weightedMatched} + \text{recommendation.spScore.matched}}{\text{totalWeight} + \text{recommendation.spScore.total}} \times 100$ )
  return totalWeightedScore
end function

```

matches to total included attributes. For this purpose, the fields `TotalIncludeCount`, `MatchesCount`, `QualityScore`, `SystemPropertiesScore`, `WeightedScore` and `TotalScore` were added to the model of the resulting `ApproachRecommendation` in addition to the `SuitabilityScore` field. The new total suitability is indicated by `MatchesCount/TotalIncludeCount`. The `QualityScore` concerns only the QAs and subattributes and holds an object containing the fields relevant for the calculation and specifies how many matches there are in relation to included QAs and subattributes. The same is true for the `SystemPropertiesScore`. The `TotalScore` field of the `ApproachRecommendation` is the combined result of QAs, subattributes and SPs. The `WeightedScore` is a special feature, because in case of additional weighting factors of the QAs, it includes them in the calculation. This can be seen in the algorithm from Algorithm 5.1, which is especially decisive for the scenario-based configuration. These weightings result from the importance of the scenarios, otherwise a weighting of the factor 1 is calculated. Based on this, the total weighting is calculated for each possible migration approach recommendation (here: `recommendation`) for the QAs and subattributes, if these are included. The total score is then calculated from the relation of actually matched QAs (`weightedCount`) and the matched SPs to the total number of both. This produces a percentage result that can be used later for tendencies.

The whole previous explanation is based on the `RecommendationService` for the migration approaches. For the patterns and best practices, the identical procedure and steps are performed to calculate the suitability of all attributes and separately of the qualities and SPs based on the attributes of the request. In both cases, a `POST` request is sent and the recommendations are sent as a response. In this respect, the `RecommendationService` could be extended with regard to the **Req. Backend-4** and thus the entire requirement could be fulfilled.

5.5 Frontend Design

To keep all models and services in sync with the backend, the `npm run openapi-gen` command of the associated NPM module must be called after each model and API customization. This keeps all models and services in the Angular application up-to-date based on the OpenApi 3 specification, using the `swagger.json` file of the repository folder as input. To meet the requirements of Section 5.2.2, an implementation of the user interface with functionalities was done in increments from phase one to phase three of the tool.

Scenarios and Quality Attributes

To fulfill **Req. Frontend-1**, a new component was first added to phase one, step three of the tool. This replaces the former placeholder for the scenario declaration. Figure 5.1 shows the created user interface, which contains a list of scenario objects that can be re-sorted via drag-and-drop. A new scenario can be added by using the green button with the plus and removed using the trash can on the right side of a scenario. On the right side, for each selected scenario, there is a selection of QAs and subattributes in relation to it, which can be selected via a checkbox. The QAs can be expanded to allow a more specific selection of the subattributes via the checkboxes. Changes to the list of scenarios are only sent to the backend to be created or updated in the database if the changes are saved. To get a list of all scenarios, the asynchronous method `requestScenarios` (see Listing 5.7) can be called via the project service, which manages all data relevant for a project within the application. Except for the scenarios, there is no functionality yet regarding project management. This method expects the Hypertext Transfer Protocol (HTTP) response via the API of the GET request of the method `listScenario` which was automatically added by the `openapi-gen`. In a similar way, after extending this, the QAs and subattributes can be read via the `AttributeService`.

Listing 5.7 Request Scenarios for Scenario Component (Typescript).

```
async requestScenarios(): Promise <void> {
  try {
    this.scenarios.next(
      await lastValueFrom(this.scenarioService.listScenario())
    );
  } catch (err) {
    console.log(err);
    this.utilService.callSnackBar(
      'Error! Scenarios could not be retrieved.'
    );
  }
}
```

Basically, this component works with several lists of scenario objects, so that one list contains original scenarios from the database, one list contains the changed scenarios that should be updated, one list for new scenarios and one list for scenarios that should be deleted. This can ensure that by saving the changes, all the desired operations are performed, and the correct services are called. For example, the `createAll()` method iterates through all scenarios in the list of scenarios to be created, sending a POST request to the backend so that they are created as a result in the database.

5 Implementation

In this section, scenarios can be created based on the [ATAM method](#). For this purpose, quality attributes you want to optimize can be assigned to the scenarios. For importance, a scale of prioritization is applied, where **A** means 'important', **B** means 'medium' and **C** means 'less important'. For difficulty, **A** means 'very difficult', **B** means 'normal' and **C** means 'easy'.

The screenshot displays the user interface for creating scenarios. At the top, there are navigation buttons: 'Back', 'Next', 'Save Changes', 'Skip to Phase 2', and 'Skip to Phase 3'. The main area is divided into two panels. The left panel, titled 'Scenarios', features a green plus icon and contains two scenario cards. Each card has a 'Name *' field, 'Importance' and 'Difficulty' dropdown menus, and a 'Description' text area. The first card is 'Example Scenario' with Importance 'A' and Difficulty 'B', and a description 'This is an example scenario.'. The second card is 'Example Scenario 2' with Importance 'B' and Difficulty 'C', and a description 'This is another example scenario.'. The right panel, titled 'Quality Attributes', lists various attributes with checkboxes and dropdown menus: Business, Compatibility, Independence, Maintainability, Performance, Portability, Reliability, Scalability (checked), and Security.

Figure 5.1: User Interface of Scenario Component.

Scenario-based and Manual Configuration

In phase two of the tool, it is possible to provide several attributes as input, so that a request can be made to the backend and recommendations for suitable migration approaches are provided as a result. In order to integrate the quality model into this phase and fulfill **Req. Frontend-2**, a new separate step was defined exclusively for the QAs, subattributes and SPs in the different steps of the configuration. In the state before the extension, the QAs (there still requirements) and SPs (at that time still metrics) were included in another section. In the section of the qualities as part of the configuration, the QAs, subattributes and SPs can either be included, excluded, or kept neutral with the respective radio buttons.

Basically, the component was redesigned, so that it differentiates between two modes. On the one hand, there is the already existing manual configuration with the individually selectable attributes or selectable presets and, on the other hand, there is the scenario-based configuration. A decision between the two modes can be made in advance. The selection of the desired mode is passed with the help of a parameter during routing. The passing of parameters can be declared in the `RoutingModule` of the Angular application. In this case, the mode parameter is specified via `:mode` at the end of the path:

```
path: 'phase/2/recommendation/configure/:mode',  
component: RecommendationConfiguratorComponent
```

As it can be seen in Listing 5.8, the mode of the component is set to scenario-based during initialization by using a simple `Boolean`, if this mode is selected. The two modes differ in that the manual configuration is not necessary in scenario-based mode, but that the quality aspects can be

transferred from the defined scenarios in phase one. However, in addition to the simple transfer of the attributes, it is also possible to make further adjustments to the configuration, which leads to the familiar manual configuration. Here, the QAs and subattributes no longer appear, but all other attributes are displayed. Thus, for example, SPs can be specified in addition to the qualities from the scenarios. It is also essential that the user is informed that the importance of the scenarios is transferred to the weightings of the assigned QAs in the calculation.

Listing 5.8 Set Scenario-based Mode by Routing Parameter (Typescript).

```

this.sub = this.route.params.subscribe((params) => {
  if (params["mode"] == MODES.modeScenario) {
    this.scenarioBased = true;
  }
});

```

This is made possible by the fact that different ng-containers can be shown in the HTML template of the component through the Boolean of the scenario-based mode set in advance, so that the contents shown differ and thus a different routing is made possible. Thus, only one component is used for both modes.

By starting the search process, a request is created by the `ApproachRecommendationService` with the configuration made and sent to the backend so that the input can be processed and the generated recommendations are set to the value of the `recommendations` field of the `ApproachRecommendationService` and are thus accessible to all components.

Approach Recommendations Results

In order to fulfill **Req. Frontend-3**, quality-specific columns should be added to the already existing columns in the table overview of the results for the recommendations. These hold the information about matches of QA and SP inputs to the respective migration approaches. Also in this case, a general distinction is made between the scenario-based mode and the manual mode or preset mode based on the parameters of the path, as described in Section 5.5.

The tabular representation shows the list of migration approaches and respective matches. The respective first column shows the total matches of all input attributes, which are fulfilled by the migration approach. Besides the basic information of the approaches, there is the quality section, which shows the matches for QAs and subattributes, as well as the matches of the SPs of each approach. A tendency for suitability based on the qualities is shown with the help of an arrow next to the two match scores, which can give the user information about the quality score. It is important to note that the tendency is based on the overall evaluation of the QAs and the SPs, and therefore the weightings by importance are also included in the scenario-based mode. The recommendations and scores can be taken from the response from the `ApproachRecommendationRequest` to the backend from Section 5.5. The calculation of this in detail can be made comprehensible with the help of Section 5.4.

When the individual entries of the table are expanded, a detailed view for the matches and mismatches of the input attributes for each category appears. Via the button with the eye as symbol, a complete detailed view of the individual migration approaches and its matches/mismatches can be accessed.

This function was already available before the extension and has only been adapted in that the QAs, subattributes and SPs are displayed in a restructured way. In addition, all columns can be sorted by the values in ascending and descending order.

Figure 5.2 shows an example output in scenario-based mode, where the quality inputs include *Performance* (importance B), *Scalability* (importance B) and the SP *Coupling*. Likewise, microservices are desired as an output, so this was given as an input attribute for the recommendation request. Since the third approach of the sorted list from Figure 5.2 of Bajaj et al. [16] satisfies all of the given input attributes except *Scalability*, the approach has a total of 5/6 matches. This results in 3/4 matches for the QAs and subattributes and 1/1 matches for the SPs. It should be noted at this point that the two subattributes of *Performance* (*Resource Utilization* and *Time Behavior*) are also included in the match statistics, increasing the number to four quality aspects. As a result, the tendency with regard to the qualities and SPs reaches a certain percentage and is classified as very positive (a green arrow pointing upwards). The following entry by Ivanov and Tasheva [73], fulfills only one of four QAs and no SPs and thus has a rather negative tendency (yellow arrow pointing downwards). In this respect, there are the levels light green arrow up ($\geq 75\%$), green arrow inclined up ($> 50\%$), black arrow horizontal ($> 30\%$), yellow arrow inclined down ($> 15\%$) and red arrow down ($\leq 15\%$). The values for the levels can be adjusted at any time in the constants of the Angular application.

Information				Qualities			Actions	
Matches	ID	Title	Authors	QA Matches	↓ SP Matches	Tendency		
5 / 6	13	Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic	De Alwis, Adambarage Anuruddha Chathuranga; Barros, Alistair; Fidge, Colin; Polyvyanyy, Artem	4 / 4	1 / 1	↑	👁	⬆
5 / 6	20	Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning	Dehghani, MohammadHadi; Kolahdouz-Rahimi, Shekoufeh; Tisi, Massimo; Tamzalit, Dalila	4 / 4	0 / 1	↑	👁	⬆
5 / 6	19	GreenMicro: Identifying Microservices from Use Cases in Greenfield Development	Bajaj, Deepali; Goel, Anita; Gupta, S. C.	3 / 4	1 / 1	↑	👁	⬆
2 / 6	21	A Hot Decomposition Procedure : Operational Monolith System to Microservices	Ivanov, Nikolay; Tasheva, Antoniya	1 / 4	0 / 1	↓	👁	⬆
2 / 6	1	Functionality-oriented Microservice Extraction Based on Execution Trace Clustering	W. Jin, T. Liu, Q. Zheng, D. Cui and Y. Cai	0 / 4	1 / 1	↓	👁	⬆

Figure 5.2: Overview of Results of Recommendations for Migration Approaches.

Since in the result of the scenario-based mode the tendency may differ based on the weights, in the HTML template of the component the output of the correct tendency icon for the table cell is determined based on the Boolean for the mode. This can be seen using the code snippet Listing 5.9.

In this snippet, the class for styling is set up by using the `getScoreIconStyle` method, which returns the correct SCSS selector based on the tendency level. Provided that the manual mode is active, the calculation is performed based on the `totalScore` (without weights). If the scenario-based mode is active, the `ng-template #scenarioScore` replaces the content of the cell, using the same method, but passing the weighted score for calculation.

Listing 5.9 Quality Tendency Table Cell of Recommendation Results (HTML).

```

<td mat-cell class='approach-cell' *matCellDef='let recommendation'>
  <mat-icon *ngIf="!scenarioBased; else scenarioScore"
    [class]='getScoreIconStyle(recommendation.totalScore, columnData.columnDef)'>
    arrow_circle_up</mat-icon>
  <ng-template #scenarioScore>
    <mat-icon
      [class]='getScoreIconStyle(recommendation.weightedScore, columnData.columnDef)'>
      arrow_circle_up</mat-icon>
    </ng-template>
  </td>

```

Patterns and Best Practices Configuration and Results

The functionality needed to satisfy the **Req. Frontend-4** and **Req. Frontend-5** in terms of patterns and best practice in phase three is nearly identical to the previous implementation of the configuration and recommendation results of the migration approaches. Therefore, the functionality is not explained repetitively. The difference is that an overview for the recommendations for patterns and best practices are available in the same component. Thereby, a toggle button between patterns and best practices can be used to decide which category should be displayed. In addition, the components for the view of all available patterns and best practices, also with a switch toggle, were implemented in the same way as the view of the migration approach exploration. Likewise, the detailed view is implemented similarly to the detailed view of the approaches, which is already available in advance.

In both cases, the configuration, whether manual or scenario-based, is transferred from phase two to phase three and applied to the recommendations for patterns and best practices. Preset support, as in phase two, was not implemented due to the exclusive focus on quality aspects.

Configuration Overview

To satisfy the last requirement **Req. Frontend-6** for the frontend, an overview of all input attributes must be provided during configuration and result display. To enable this and not reduce the rest of the view, a sidepanel has been added that can be inserted and pushes the rest of the user interface to aside, so that all elements are still visible.

The overview can be opened and closed in any desired component, so that the current input of attributes is always visible. The example in Figure 5.3 shows a scenario-based configuration in which the transferred QAs and the associated scenarios with their importance are displayed. The individual categories can be expanded and collapsed. For the manual configuration, the overview differs in that the scenarios do not appear and the QAs/subattributes and SPs are combined in one category.

To open, close and set the current mode of the overview, the methods required for this are implemented via the `UtilService`. The sidepanel is implemented by using a customized Angular Material Sidenav, which is set in the `UtilService` after the first initialization of the main view of the application, so that it can be controlled by any component.

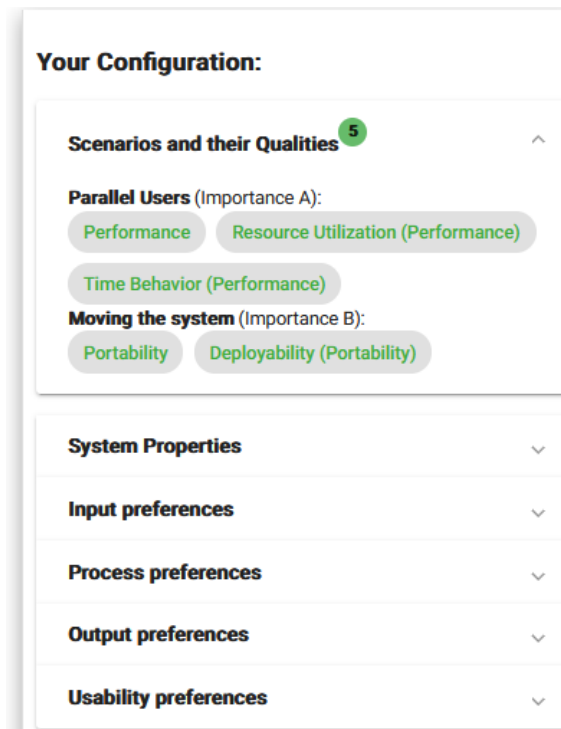


Figure 5.3: Configuration Overview in Sidepanel.

Through the steps listed for the implementation, the tool could be extended by means of the explained functional elements in the user interface and the functionality in the backend based on the requirements, so that the quality model is an essential part of the 'Architecture Refactoring Helper' tool [65]. The detailed history of the entire implementation can be found in the repository of Fritsch [57].

6 Evaluation of Usability

During the implementation, emphasis was placed on ensuring that the usability for future users of the tool is high by making the tool's controls as intuitive as possible and providing the necessary information regarding the functionality. In order to evaluate the tool in terms of usability, a survey was conducted that uses specific tasks for the participants to examine all functionalities and user interfaces implemented as part of this work.

For the design and execution of the survey, the seven-step process of Kasunic [81] was used as a guideline. First, the research objectives and the target audience are identified and a design sampling plan is created. Subsequently, the questionnaire is developed and a pilot test questionnaire is conducted. Finally, the questionnaire is distributed and the results are analyzed after all participants have completed the survey.

The **research objective** here is to investigate how usable the extension of the tool is in relation to the integrated quality model. This also includes how comprehensible the implementation of the results obtained in this work is perceived. The idea of the survey is to collect qualitative answers to statements concerning the usability of the extension. Thus, direct ambiguities and suggestions for improvement can be derived after evaluating the results.

Since the tool is primarily directed at the target group of software architects and developers who are significantly involved in the migration from a monolithic architecture to a microservice architecture, four suitable participants could be drawn on via industry contacts of the Empirical Software Engineering (ESE) group of the Institute for Software Engineering (ISTE) at the University of Stuttgart. The requirement for the selection of participants was that they have a technical role and significant professional experience. In addition, they should have knowledge in the topic of microservices migration, whereby these correspond to the **target audience**. As a **sampling plan** it is envisaged that three to five participants are representative for the survey. This is stated in an article by Nielsen [106], where this is defined as the optimal size for analyzing the usability. Further the author indicates that after the fifth participant resources are used unnecessarily and nothing new is learned, because after that the number of repetitions increases.

6.1 Survey Design

In order to ensure that the survey covers all parts of the extension of the tool equally and that a distinction can be made to the functionalities existing before the extension, a rough structure of the survey was first worked out. This structure includes an introduction that explains the background, the idea of the extension and the area that is covered by the survey. The entire survey can be seen in Appendix A.

Following the introduction, a statement was asked with two options, as to whether and in what form the information and answers provided may be used. Here, both options were for anonymous use of the data, with one having the restriction that no verbatim information may be used.

The first data collected for the evaluation is personal information related to professional experience, microservices experience, and current job description.

As a next step, two tasks are described to guide users in exploring the functionalities of the implemented extension and to cover all areas within the scope of this work. This means that first the scenario definition in phase one and the assignment of the QAs shall be performed. Then, using the scenario-based mode after additional configuration of a SP, the recommendations for migration approaches (phase two) and patterns/best practices (phase three) shall be presented. After that, as a second task, the manual configuration with the newly integrated quality model will be investigated. Following the two tasks, the user will have the opportunity to explore the tool further.

After the participants have been able to explore the tool for the extended functionality, the actual evaluation of usability starts. For this purpose, statements were given for each of the three phases regarding the understanding and usability of the newly implemented elements. Likewise, such statements were also given for the respective instructions. The participants can now agree or disagree with these statements and give a reason for their decision in each case in textual form. Subsequently, statements given for the entire extension and a field for further suggestions and proposals are provided. In this way, the results can be collected for the individual areas. Thereby all answers of the evaluation survey were optional.

For the **test pilot questionnaire**, no member of the target group was consulted, instead only self-performed test runs were used to identify and fix bugs. The **distribution** was then done by sending mails to the potential participants. So that the tool can be investigated by the participants of the survey, it was hosted so that they can access it. After all participants have completed the survey, the results are summarized and presented in narrative form, as this allows the qualitative data to be structured and presented in context.

6.2 Survey Results

This section summarizes and compares the results of the evaluation survey. For this purpose, each area or phase is considered individually and subsequently the general statements about the tool or extension are collected.

Personal Information

First, the personal information of the participants can be clarified. Three software architects take part in the survey, with another participant having the job description of a lead consultant. Among them, two have professional experience of over ten years and two have experience of over 20 years. For microservices experience, two indicated five years and one indicated two and a half years, with one participant having no professional experience with microservices. The data for experience with migration to microservices is somewhat similar to this, with five years given twice, one year given once, and no experience on this given once.

Phase one - Scenarios and Quality Attributes

For the evaluation from phase one, the statements that it was easy and intuitive to create/edit scenarios and it was easy and intuitive to assign QAs to these scenarios were assessed first. In this regard, the feedback for both statements was positive, with one participant indicating that there was ambiguity regarding how to define scenarios specifically. Another participant gave the suggestion, “Consider moving ‘Quality Attributes’ component into each scenario for clarity”. Another stated, “Yes it is easy to assign a quality attribute to a scenario”. Also assessed in relation to the scenarios was the statement that the instructions were helpful in performing the operations. This statement was agreed to by three participants, with one not providing any information. Additionally, one participant again stated that a scenario needed to be stated more generically than the participant intended. The final part of phase one was an assessment of the user interface’s structure. The goal was to determine if it was easy to understand and useful. Overall, there were three agreements and one disagreement. The disagreement was based on the fact that the sketches of the components to follow in the future in phase one had an unclear reason and that he was “Not sure if the wizard navigation in phase 1 and the offer to skip to other phases helps at all”. As a suggestion, breadcrumbs were given, so that even across phase one, the current page can be assigned to a phase. One participant who agreed with the statement had as a suggestion, that detailed information such as priority should only be displayed when a scenario is selected directly.

Phase two - Configuration and Migration Approaches Recommendation

The statements for phase two start with the newly introduced overview of the configuration on the right-hand side. This is considered as useful by all participants, with two also agreeing that it is easy to understand. The other two participants stated that it only becomes clear during configuration that it is needed for this purpose. When asked if the various selection modes were straightforward, there was one agreement and for three participants the modes were unclear or misunderstood. For example, one participant stated that “I did not understand the modes well”. Another indication did not directly relate to the statement, but suggested replacing the radio buttons with tri-state switches during configuration. Whether the configuration of additional filters is helpful was answered in the affirmative by three participants, with one participant providing no indication. The statement, ‘The result view, especially in terms of qualities helps to find suitable approaches’ received agreement from three participants and one non-directed statement. The non-directed statement suggests not displaying the ID column. Another participant, in addition to agreeing, indicated that more context on the connection between the scenarios and each recommendation might additionally help with the overview. For the display of the tendency for the recommendation suitability based on qualities, one participant indicated that this display was useful, although for two participants this display was not clear. A fourth respondent indicated that they did not need the tendency display, but did not reject the statement. The statement, ‘It is comprehensible how the rankings of the migration approaches are generated’ received three approvals and one disapproval. One participant gave an additional comment to his agreement, describing that only after a short thought it became clear what the abbreviations QA and SP stand for. Also positively assessed was the chosen way of presenting the results. There were four participants who agreed with this, whereby there was an individual comment that would once again find it helpful to relate the results to the scenarios.

Phase three - Patterns/Best Practices Recommendation

Only three participants took part in the evaluation of phase three. The section of the survey on phase three contains the same statements as in phase two, with the exception of the statement about the configuration overview on the right-hand side. In this respect, the statements about the selection of the modes being straightforward, that the additional filters are helpful and that the results overview helps to find the appropriate recommendations with regard to the qualities, were either referred to the answers from phase two or the same answer was given. Only one participant pointed out that an explanation of the meaning of matches, tendency and actions in the results overview would have been helpful. For the statements regarding stated tendency, comprehensibility of rankings, and manner of presentation, the same responses were given as in phase two, only in relation to patterns and best practices.

General Statements

The last section of the survey contained general statements or free text fields for further input. Here, there were also responses from only three of the four participants. The statement ‘The operation and navigation are intuitive and easy to follow’ was agreed to by one participant, with one not providing a statement and another participant only providing a suggestion, with no directed opinion. This one includes that “Considering to drop the nested wizards, I would prefer a search engine that allows to modify the filters next to the result set”. On the general statement of whether the extension can help to support migration in terms of quality goals, there are two participants who agree with this. One could not give an answer to this. One explanation for agreeing was that it specifically offers the appropriate approaches, patterns and best practices, which makes the tool very useful in this regard. For further comments, there was only one comment from a participant about the tool, describing that it has not become clear how in the scenario-based mode better results can be achieved than with the manual configuration. However, the same participant concludes by describing that “It is a nice tool and can be useful in practice”.

By evaluating the results, we can create an overview of which parts of the extension or tool have achieved a high level of usability and which still have room for improvement. This can be determined by looking at the suggestions and disagreements to certain statements.

6.3 Discussion of Survey Results

This section discusses the summarized results of the evaluation conducted in the form of a survey. For this purpose, the answers given are combined and interpreted in a directional manner. If it is determined that the usability is limited in certain areas or if there are explicit suggestions for improvement from the participants, these are analyzed and future improvements are suggested.

From the results of phase one, it can be inferred that the CRUD methods are designed to be intuitive and simple in the user interface regarding the scenarios. The instructions are also helpful in this respect, as they can ensure better understanding. In this regard, this should be retained but still expanded to provide additional information via the textual descriptions. However, it is desired that for a scenario possibly more specific information can be provided. In this respect, groups

or subordinate scenarios could be introduced, which can be divided into more specific parts for a scenario. However, if more specific information for a scenario in the form of descriptions or other fields is meant, then additional helpful information elements for a scenario should be added. For this purpose, a scenario should be expandable and collapsible to display or hide information due to space limitations. This contradicts with another statement. Here it was suggested to add the QA assignment component to the scenarios individually so that this is clearer. This would probably take up too much space and result in a cluttered presentation. Thus, one suggestion would be to rather use a pointer to assign the component to the particular selected entry in the scenario list. Additionally, the actual assigned attributes could be displayed by using small tags within the scenario so that this does not take up the entire space of the entry.

The basic structure and design of this phase receives mostly support, thus no major adjustments are needed. However, it was stated that the sketches in this phase, which show screenshots for the intended interface, can be confusing. However, it must be said that these are crucial for further understanding of the tool, as the intention of the future implementation is depicted here. In addition, descriptive texts could lead to less confusion about the placeholders. This rather concerns the general usability and is not aimed at the extension in the scope of this work.

The interpretation of the results of phase two and phase three are summarized here, since they have an almost identical user interface and functionality and the result data on both phases coincide. It was commented that the two modes or the difference between them was not completely clear. There should be more instructions on this or a short introductory tutorial could help to understand the functionality and differences. It was also stated that in the scenario-based mode, additional filters are helpful and understandable, so the function should be retained. Likewise, the quality-based results display is rated as good in usability, with one suggestion indicating that the columns displayed could be optionally selectable so that only the information desired by the user is displayed. Since the modes, and in particular the scenario-based mode, have caused some confusion, more context on how each scenario targets the recommendations is also desired. Scenario matches for each recommendation entry could help with this so that this can be clearly mapped. The presentation of results was generally perceived to be good, implying that the tabular presentation with expandable elements was found to be useful. However, it must be said that this type of presentation was already present in the previous implementation and, besides minor adjustments, was only extended to include the quality elements. The element of the tendency based on the qualities was not clear to everyone here, whereby a small description of the meaning, for example with a tooltip, could help. This also applies to the other elements, so that clarity is provided for the user.

The results for the cross-phase elements of the user interface are positive in terms of usability. Here, the configuration overview on the right-hand side is perceived as good and comprehensible, so that this should be retained. However, adjustments can be made to this so that the elements are explained as part of an introductory tutorial. One suggestion was to move the entire configuration to this overview, which would possibly result in a cluttered configuration due to space limitations. Likewise, the selection could be done using tri-state toggles rather than radio buttons. This is actually already a proposed future improvement and an identified issue in the project. So that in the future it is clear in which phase and in which step the user is located, the currently used 'steppers' could be replaced by cross-phase displayed breadcrumbs.

In the general statements, it was also stated that the extension to include the quality model can support migration, since suitable approaches to migration and architectural design decisions can thus be recommended on the basis of the defined QAs. This can be supported by two statements of the participants. In this respect, according to the participants, the main intention of the extension can be fulfilled.

As a conclusion of the results, the usability of the prototype in the context of the extension is thus guaranteed, in which the structure and control elements are comprehensible and intuitively designed. Only minor adjustments in the form of more textual instructions and an introduction tutorial to the controls and elements would address or implement most of the concerns or suggestions. This may be part of future implementation work, as some of this is not within the scope of the extension, but addresses the entire tool. In addition, it is apparent from the general statements that the tool can help with migration in terms of quality goals and it can be useful in practice.

7 Discussion

In order to address the results of the objectives of the research and the evaluation of the implementation can be critically addressed, the research questions of this thesis must be examined. For this purpose, the key findings of the research on research questions *RQ 1* and *RQ 2* are considered first. It is also discussed to what extent the quality model could be implemented as an extension of the already existing tool by Haller [65], thus addressing *RQ 3*.

RQ 1: Incorporated Quality Goals in Migration to Approaches

QAs can play a role in several phases of migration. For example, they can be used for the assessment that follows the migration, or they can be included at the beginning of the process so that quality goals can be set for the target architecture. In the context of this work, the consideration in this phase is examined above all, since this serves to include the quality goals already at the beginning in the migration, so that suitable decisions can be made, in order to reach these. There are few references in the literature for the role of QAs in this phase of migration in the context of microservices. This can possibly be attributed to the fact that architecture assessment based on quality objectives is not microservice specific, but is equally possible for all architectures. Therefore, the generalized ATAM was used, which allows an architecture assessment for all software architectures. However, for this purpose, a method that explicitly considers certain quality goals and requirements for an architecture in the design phase for microservices would help in the future, since this type of architecture has specific characteristics that differ from others in this respect. It is important to note that the quality goals may be reassessed and incorporated during the migration process. In this respect, the goals should be readjusted in the choice of strategy for the migration and the subsequent choice of patterns and best practices. This can be depicted by the quality model, through which attributes and properties are explicitly mapped in each phase, thus allowing for new or changed quality goals during the process.

Besides the identification of QAs, SPs could be recognized. However, it must be said that the identified SPs are often recognized as QAs. The recommendation from the results of this work is to regard these separately, since these affect the qualities of an architecture based on their degree and do not represent a quality themselves.

So that quality goals can be linked to migration approaches, each of the approaches provided by the Institute for Software Engineering (ISTE) was examined to determine which QAs and SPs are optimized as a result. In this way, quality goals can be mapped directly to suitable approaches, which optimize the target architecture through their methods. However, the approaches often do not consider the identified trade-offs among each other or the influences of the SPs on the QAs. This can result from the fact that in the presented approaches mostly the quality optimizations of the research are emphasized and possible downsides do not represent a relevant part of the elaboration.

Accordingly, this can be addressed and included during the process under consideration of the quality model resulting from this work, so that the stated optimizations of the approaches do not contradict each other.

RQ 2: Quality Goals to Architectural Patterns and Best Practices

In the migration approaches, trade-offs or possible negative influences were mostly not considered. At least for the patterns this is not the case. There is sufficient literature dealing with the trade-offs of the individual architectural patterns. In the case of the best practices, on the other hand, such indications could not be found. Likewise, the direct references to the optimization of the QAs and the treatment of the SPs can give direct assignments to the identified patterns and best practices. Similarly, the quality model can incorporate interdependencies that may not have been considered among themselves.

Another aspect is that the distinction between architectural patterns and best practices is often not completely clear, respectively the difference between the opposite anti-patterns and smells [127]. Therefore, efforts have been made to provide a distinction so that quality goals can be mapped to patterns and adherence to certain best practices can lead to a quality optimization. This can support practitioners in architectural design of a microservice architecture.

Across *RQ 1* and *RQ 2*, it was found that the newly introduced business attributes have little appeal and are hardly mentioned in the literature of approaches and patterns/best practices. In addition to the quality goals of the product, the associated business aspects should also be taken into account here, since these are at least as crucial for practitioners in industry.

RQ 3: Extension of Tool by Quality Model

The tool developed by Haller [65] could be extended by the quality model by performing the implementation steps described in Chapter 5. In addition to the required models and data, the functionalities were realized, which enabled the inclusion of QAs in each phase of the framework. Furthermore, in addition to the newly structured QAs and SPs, the scenario-based configuration and quality-related result view could also be added. Here, each component was implemented with the condition that the original functionality was not restricted or significantly changed. In this respect, the existing structures and prerequisites could be retained, so that future adaptations and extensions are still possible.

In the quality model, the SPs have positive or negative effects on the QAs, whereby these mutual influences are documented exclusively and are not part of the extension, to first of all ensure simplicity in usability. The same applies to the trade-offs to the QAs. These are not shown to the users, because this data is not yet available in the configuration of the database and no graphical elements show them yet. However, the implemented model of the QAs/subattributes or the SPs already provides relationships in the form of trade-offs, which would only have to be added to the configuration. Since this is a prototypical implementation, sample data was used in this work. This means in detail that not every mapping of attributes and properties was assigned to the migration approaches, patterns and best practices, since these are not relevant for the evaluation of usability and for prototypical testing. In this respect, the examples of approaches, patterns and best practices

already available in the database were extended by the corresponding correct mapping of the quality model or data sets were added completely new, so that an unrestricted functionality could be realized.

As a result, the three research questions formulated within the scope of this work could be addressed and answered, while taking into account the aspects discussed previously. From these aspects, future improvements and adaptations can be derived, which can be contemplated in further research projects.

8 Threats to Validity

With regard to this work, sufficient results could be obtained for the literature review. However, limitations and possible threats to validity of the results have to be considered. The same is true for the evaluation conducted for the extension of the tool. Here there are also threats to the validity based on the implementation or the interpretation of the results.

Literature Review

For the identified QAs and SPs, it cannot be guaranteed that all attributes and properties relevant to microservices could be detected. Therefore, the decision to conduct a Rapid Review [32] should ensure that these threats to validity are minimized. This also applies to the trade-offs found and the influences of the properties on the attributes. There is a possibility that additional trade-offs and influences exist that could not be extracted from the literature. Moreover, these often lack direct evidence in the form of evaluations of the claims. Another aspect in this respect can be recognized in the mapping of QAs and SPs to be optimized. Here, the literature of the approaches claims that these approaches lead to optimization. However, there is a large number of the approaches which can prove an actual optimization only conditionally or not at all. The same applies to the patterns and best practices, where the mapping of the literature are indications that would have to be proven individually. Moreover, in many cases of the migration approaches and also in some cases of the patterns and best practices, attention is not paid in each case to possible complications and newly emerging constraints due to the combination of these. For example, the choice of a migration strategy could have an impact on feasible architecture patterns and limit the choice or prevent the optimization of QAs by a pattern. This could not be addressed as it is outside the scope of this work.

Evaluation

In the evaluation, on the other hand, different aspects arise that should be taken into account. In this context, there were participants who had not gained professional experience in the area of microservices, but were able to gain knowledge in this area through other opportunities. This could affect the validity of these participants' answer. In addition, there is the question of whether all statements in the survey were set up in an understandable way with respect to the context, and whether these cover all areas of the extension. In principle, the evaluation should focus exclusively on the extension. Since this is directly linked to the original implementation, it is possible that the participants may have the wrong perception in the form of mixing new and already existing elements. Since a prototypical implementation was realized, it is possible that future functionalities or adaptations could be interpreted as missing, even though they are not intended to be part of this work. To prevent these effects, this was taken care of, by explicitly explaining the goals and scope of the evaluation to the participants within the survey.

9 Conclusion and Future Work

For the migration from a monolith to a microservice architecture, there are many different approaches that address this corresponding process. In addition, the respective approaches have different techniques and methods as well as different results as output. By applying the migration approaches, one of the results is the optimization of certain Quality Attributes (QAs). For this purpose, efforts have already been made in the literature to assign optimized QAs to the individual approaches. In addition, there is a doctoral research project by Fritzsich [56] within the Empirical Software Engineering (ESE) group of the Institute for Software Engineering (ISTE) at the University of Stuttgart, which deals with the organizational and architectural aspects of the migration from a monolith to a microservice architecture. Within this project a number of migration approaches could be collected, which were then provided for this thesis. Since the mappings from the literature do not fully cover the collected approaches, a comprehensive mapping of QAs to migration approaches was created by examining each approach individually. For microservices, however, not only the familiar QAs from ISO 25010 [71] are of great importance here. Relevant QAs could be identified for a microservice architecture and thus a specific set of attributes could be collected. An interesting finding here is that, in addition to QAs, System Properties (SPs) could be identified which also play a decisive role in the migration process. This is because the degree to which these SPs are present has a great impact on the target architecture as well as the QAs.

Evidence was gathered from the literature on which architectural design decisions lead to the fulfillment of certain quality aspects. To this end, the results could be merged, so that an overview of the findings could be given and thus quality goals can be reached by the suitable choice of patterns and best practices for the architecture. In addition, an overview of the trade-offs between QAs for microservices and architectural trade-offs already addressed in the literature was provided.

What could be identified as a fundamental gap in previous research is the combination of the assignments of the QAs to migration approaches and the subsequent assignment of architectural design patterns and best practices, taking into account the interdependencies and trade-offs. This work was able to close this gap in that a comprehensive quality model was developed on the basis of the quality aspects. This model brings together all of the preceding aspects in the migration process in each phase. The design was chosen in such a way that it can act as a guide for practitioners, as it maps the QAs and SPs specifically to the migration approaches, design patterns and best practices. Thus, the model can recommend suitable migration approaches and architectural design decisions based on quality goals. This holistic consideration makes a significant contribution to further research, as it brings together several aspects in the quality-related context.

A tool called 'Architecture Refactoring Helper' [65] was developed in the context of a master thesis by Haller [64], so that practitioners and software architects can be supported in the migration to microservices. However, the quality aspects were not fully addressed. To this end, this tool has

been extended and implemented with the aforementioned quality model so that it can guide the migration process in terms of quality goals and make it accessible in the form of an actual practical application.

For this extension, an evaluation of usability in the form of a survey was conducted as part of this work. The results of the survey of four practitioners from industry showed that the integrated quality model can be useful in practice and that the user interface is predominantly designed to be intuitive and understandable.

Future Work

The results of the literature review could gather evidence about the connection of QAs to migration approaches, architectural patterns, and best practices. Further research should investigate the possible impact of combinations of multiple architectural patterns and best practices in migration on QAs and SPs. It is also becoming apparent that the choice of migration approach has a significant impact on the possible choice of architecture patterns. In this respect, it might be possible in the future to investigate which approach is eventually compatible with which pattern. In the case of the identified QAs for microservice architectures, research could address the newly introduced *Business* attributes, which are little covered in the literature but are a crucial factor for practitioners in the migration process.

Another aspect, in addition to the findings of the trade-offs, is that positive dependencies between the attributes and properties can be dealt with, as there is already sufficient evidence for this in the literature. The quality model can be expanded to include these aspects, so that this can be incorporated into future research. This also indicates that the implementation of the 'Architecture Refactoring Helper' tool extension [57] in particular can be complemented by the aspects addressed, so that additional information is available to users to help them in the migration process. With regard to the extension, for instance, indications could be displayed if certain quality goals are in trade-off or if they influence each other positively. Furthermore, the influences between the SPs and the QAs could be integrated into the tool. In this way, the tool could be extended to include automatic pre-selection in the configuration based on the dependencies. This would result in more specific recommendations regarding migration approaches and architectural design decisions.

Bibliography

- [1] M. Abdellatif, A. Shatnawi, H. Mili, N. Moha, G.E. Boussaidi, G. Hecht, J. Privat, Y.G. Guéhéneuc. “A taxonomy of service identification approaches for legacy software systems modernization”. In: *Journal of Systems and Software* 173 (Mar. 2021). ISSN: 01641212. DOI: [10.1016/j.jss.2020.110868](https://doi.org/10.1016/j.jss.2020.110868) (cit. on pp. 12, 40, 41).
- [2] M. Abdellatif, R. Tighilt, N. Moha, H. Mili, G. El Boussaidi, J. Privat, Y.-G. Guéhéneuc. “A type-sensitive service identification approach for legacy-to-SOA migration”. In: *International Conference on Service-Oriented Computing*. Springer. 2020, pp. 476–491 (cit. on p. 41).
- [3] M. Abdullah, W. Iqbal, A. Erradi. “Unsupervised learning approach for web application auto-decomposition into microservices”. In: *Journal of Systems and Software* 151 (2019), pp. 243–257 (cit. on pp. 41, 47–49).
- [4] M. Ahmadvand, A. Ibrahim. “Requirements reconciliation for scalable and secure microservice (de) composition”. In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. IEEE. 2016, pp. 68–73 (cit. on p. 39).
- [5] M. Ahmadvand, A. Ibrahim. “Requirements reconciliation for scalable and secure microservice (de) composition”. In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. IEEE. 2016, pp. 68–73 (cit. on pp. 41, 48).
- [6] A. Alaasam, G. Radchenko, A. Tchernykh. “Refactoring the Monolith Workflow into Independent Micro-Workflows to Support Stream Processing”. In: *Programming and Computer Software* 47.8 (2021), pp. 591–600 (cit. on pp. 41, 48).
- [7] N. Alshuqayran, N. Ali, R. Evans. “A systematic mapping study in microservice architecture”. In: Institute of Electrical and Electronics Engineers Inc., Dec. 2016, pp. 44–51. ISBN: 9781509047819. DOI: [10.1109/SOCA.2016.15](https://doi.org/10.1109/SOCA.2016.15) (cit. on pp. 29, 31, 32, 34, 36, 37, 49).
- [8] A. A. C. D. Alwis, A. Barros, C. Fidge, A. Polyvyanyy. “Discovering microservices in enterprise systems using a business object containment heuristic”. In: *OTM Confederated International Conferences “On the Move to Meaningful Internet Systems”*. Springer. 2018, pp. 60–79 (cit. on pp. 41, 48, 50, 61).
- [9] A. A. C. D. Alwis, A. Barros, C. Fidge, A. Polyvyanyy. “Microservice Remodularisation of Monolithic Enterprise Systems for Embedding in Industrial IoT Networks”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2021, pp. 432–448 (cit. on pp. 41, 48, 50).
- [10] M. J. Amiri. “Object-aware identification of microservices”. In: *2018 IEEE International Conference on Services Computing (SCC)*. IEEE. 2018, pp. 253–256 (cit. on pp. 41, 50).

- [11] W. K. Assunção, T. E. Colanzi, L. Carvalho, A. Garcia, J. A. Pereira, M. J. de Lima, C. Lucena. “Analysis of a many-objective optimization approach for identifying microservices from legacy systems”. In: *Empirical Software Engineering* 27.2 (2022), pp. 1–31 (cit. on pp. 41, 48, 50).
- [12] W. K. Assunção, T. E. Colanzi, L. Carvalho, J. A. Pereira, A. Garcia, M. J. de Lima, C. Lucena. “A multi-criteria strategy for redesigning legacy features as microservices: An industrial case study”. In: *2021 IEEE International conference on software analysis, evolution and reengineering (SANER)*. IEEE. 2021, pp. 377–387 (cit. on pp. 41, 48, 50).
- [13] F. Auer, V. Lenarduzzi, M. Felderer, D. Taibi. “From Monolithic Systems to Microservices: An Assessment Framework”. In: (Sept. 2019). DOI: [10.1016/j.infsof.2021.106600](https://doi.org/10.1016/j.infsof.2021.106600) (cit. on pp. 29, 32–36, 39).
- [14] D. Bajaj, U. Bharti, A. Goel, S. Gupta. “Partial migration for re-architecting a cloud native monolithic application into microservices and faas”. In: *International Conference on Information, Communication and Computing Technology*. Springer. 2020, pp. 111–124 (cit. on pp. 41, 48).
- [15] D. Bajaj, U. Bharti, A. Goel, S. C. Gupta. “A Prescriptive Model for Migration to Microservices Based on SDLC Artifacts”. In: *Journal of Web Engineering* (2021), pp. 817–852 (cit. on pp. 12, 15).
- [16] D. Bajaj, A. Goel, S. Gupta. “GreenMicro: Identifying Microservices From Use Cases in Greenfield Development”. In: *IEEE Access* 10 (2022), pp. 67008–67018 (cit. on pp. 41, 48, 50, 61, 76).
- [17] M. H. G. Barbosa, P. H. M. Maia. “Towards identifying microservice candidates from business rules implemented in stored procedures”. In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2020, pp. 41–48 (cit. on pp. 41, 48).
- [18] L. Baresi, M. Garriga, A. D. Renzis. “Microservices identification through interface analysis”. In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2017, pp. 19–33 (cit. on pp. 41, 50).
- [19] L. Bass, P. Clements, R. Kazman. *Software Architecture in Practice: Software Architect Practice*. 3rd ed. Addison-Wesley, 2012 (cit. on pp. 33–35).
- [20] G. Blinowski, A. Ojdowska, A. Przybyłek. “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation”. In: *IEEE Access* 10 (2022), pp. 20357–20374. ISSN: 21693536. DOI: [10.1109/ACCESS.2022.3152803](https://doi.org/10.1109/ACCESS.2022.3152803) (cit. on pp. 14, 15).
- [21] J. Bogner, J. Fritsch, S. Wagner, A. Zimmermann. “Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality”. In: Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 187–195. ISBN: 9781728118765. DOI: [10.1109/ICSA-C.2019.00041](https://doi.org/10.1109/ICSA-C.2019.00041) (cit. on pp. 12, 27, 29, 32, 34).
- [22] J. Bogner, S. Wagner, A. Zimmermann. “Automatically measuring the maintainability of service- and microservice-based systems - a literature review”. In: vol. Part F131936. Association for Computing Machinery, Oct. 2017, pp. 107–115. ISBN: 9781450348539. DOI: [10.1145/3143434.3143443](https://doi.org/10.1145/3143434.3143443) (cit. on p. 37).

- [23] J. Bogner, S. Wagner, A. Zimmermann. “Towards a practical maintainability quality model for service and microservice-based systems”. In: vol. Part F130530. Association for Computing Machinery, Sept. 2017, pp. 195–198. ISBN: 9781450352178. DOI: [10.1145/3129790.3129816](https://doi.org/10.1145/3129790.3129816) (cit. on pp. 36–38).
- [24] M. Borges, E. Barros, P. H. Maia. “Cloud restriction solver: A refactoring-based approach to migrate applications to the cloud”. In: *Information and Software Technology* 95 (2018), pp. 346–365 (cit. on pp. 41, 48, 49).
- [25] P. Bourque, R. E. Fairley, IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge*. Version 3.0. IEEE Computer Society, 2014. ISBN: 0769551661. URL: www.swebok.org (cit. on p. 34).
- [26] M. Brito, J. Cunha, J. Saraiva. “Identification of microservices from monolithic applications through topic modelling”. In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 2021, pp. 1409–1418 (cit. on pp. 41, 48, 50).
- [27] A. Bucchiarone, K. Soysal, C. Guidi. “A model-driven approach towards automatic migration to microservices”. In: *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Springer. 2019, pp. 15–36 (cit. on p. 41).
- [28] M. Camilli, C. Colarusso, B. Russo, E. Zimeo. “Domain metric driven decomposition of data-intensive applications”. In: *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2020, pp. 189–196 (cit. on pp. 41, 48, 50).
- [29] R. Capuano. “Enhancing System Quality Attributes via Microservices Adoption”. In: 2021 (cit. on p. 29).
- [30] R. Capuano, H. Muccini. “A Systematic Literature Review on Migration to Microservices: a Quality Attributes perspective”. In: IEEE, Mar. 2022, pp. 120–123. ISBN: 978-1-6654-9493-9. DOI: [10.1109/ICSA-C54293.2022.00030](https://doi.org/10.1109/ICSA-C54293.2022.00030) (cit. on pp. 12, 15, 29).
- [31] B. Cartaxo, G. Pinto, B. Fonseca, M. Ribeiro, P. Pinheiro, M. T. Baldassarre, S. Soares. “Software engineering research community viewpoints on rapid reviews”. In: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2019, pp. 1–12 (cit. on p. 19).
- [32] B. Cartaxo, G. Pinto, S. Soares. “Rapid reviews in software engineering”. In: *Contemporary Empirical Methods in Software Engineering*. Springer, 2020, pp. 357–384. DOI: [10.1007/978-3-030-32489-6_13](https://doi.org/10.1007/978-3-030-32489-6_13) (cit. on pp. 13, 19–21, 23, 25, 88).
- [33] B. Cartaxo, G. Pinto, S. Soares. “The role of rapid reviews in supporting decision-making in software engineering practice”. In: vol. Part F137700. Association for Computing Machinery, June 2018. ISBN: 9781450364034. DOI: [10.1145/3210459.3210462](https://doi.org/10.1145/3210459.3210462) (cit. on p. 19).
- [34] L. Carvalho, A. Garcia, T. E. Colanzi, W. K. Assunção, M. J. Lima, B. Fonseca, M. Ribeiro, C. Lucena. “Search-based many-criteria identification of microservices from legacy systems”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 2020, pp. 305–306 (cit. on pp. 41, 48, 50).
- [35] L. Carvalho, A. Garcia, T. E. Colanzi, W. K. Assunção, J. A. Pereira, B. Fonseca, M. Ribeiro, M. J. de Lima, C. Lucena. “On the performance and adoption of search-based microservice identification with tomicroservices”. In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2020, pp. 569–580 (cit. on pp. 41, 48, 50).

- [36] R. Chen, S. Li, Z. Li. “From monolith to microservices: A dataflow-driven approach”. In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2017, pp. 466–475 (cit. on pp. 41, 49, 50).
- [37] M. D. Cojocaru, A. Oprescu, A. Uta. “Attributes assessing the quality of microservices automatically decomposed from monolithic applications”. In: Institute of Electrical and Electronics Engineers Inc., June 2019, pp. 84–93. ISBN: 9781728138008. DOI: [10.1109/ISPDC.2019.00021](https://doi.org/10.1109/ISPDC.2019.00021) (cit. on pp. 16, 29, 31–38).
- [38] M. Conway. *Conway’s Law*. URL: https://www.melconway.com/Home/Conways_Law.html (cit. on p. 36).
- [39] M. Daghighzadeh, S. M. Babamir. “A model driven and clustering method for service identification directed by metrics”. In: *Software: Practice and Experience* 51.2 (2021), pp. 449–484 (cit. on p. 37).
- [40] M. Daghighzadeh, S. M. Babamir. “A model driven and clustering method for service identification directed by metrics”. In: *Software: Practice and Experience* 51.2 (2021), pp. 449–484 (cit. on pp. 41, 50).
- [41] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, A. El Fazziki. “A multi-model based microservices identification approach”. In: *Journal of Systems Architecture* 118 (2021), p. 102200 (cit. on pp. 41, 48, 50, 60).
- [42] M. Daoud, A. E. Mezouari, N. Faci, D. Benslimane, Z. Maamar, A. E. Fazziki. “Automatic microservices identification from a set of business processes”. In: *International Conference on Smart Applications and Data Analysis*. Springer. 2020, pp. 299–315 (cit. on pp. 41, 48, 49).
- [43] L. De Lauretis. “From monolithic architecture to microservices architecture”. In: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2019, pp. 93–96 (cit. on pp. 41, 48, 50).
- [44] O. Al-Debagy, P. Martinek. “Dependencies-based microservices decomposition method”. In: *International Journal of Computers and Applications* (2021), pp. 1–8 (cit. on pp. 41, 48).
- [45] O. Al-Debagy, P. Martinek. “Extracting microservices’ candidates from monolithic applications: interface analysis and evaluation metrics approach”. In: *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*. IEEE. 2020, pp. 289–294 (cit. on pp. 41, 50).
- [46] O. Al-Debagy, P. Martinek. “A microservice decomposition method through using distributed representation of source code”. In: *Scalable Computing: Practice and Experience* 22.1 (2021), pp. 39–52 (cit. on pp. 41, 49, 50).
- [47] M. Dehghani, S. Kolahdouz-Rahimi, M. Tisi, D. Tamzalit. “Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning”. In: *Software and Systems Modeling* 21.3 (2022), pp. 1115–1133 (cit. on pp. 41, 48).
- [48] U. Desai, S. Bandyopadhyay, S. Tamilselvam. “Graph neural network to dilute outliers for refactoring monolith application”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 1. 2021, pp. 72–80 (cit. on pp. 41, 48, 50).

- [49] H. Dinh-Tuan, F. Beierle. “MS2M: A message-based approach for live stateful microservices migration”. In: *2022 5th Conference on Cloud and Internet of Things (CIoT)*. IEEE. 2022, pp. 100–107 (cit. on pp. 41, 48).
- [50] D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, R. Casallas. “Towards the understanding and evolution of monolithic applications as microservices”. In: *2016 XLII Latin American computing conference (CLEI)*. IEEE. 2016, pp. 1–11 (cit. on pp. 37, 41, 48–50).
- [51] 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*. 2018, pp. 1–6 (cit. on pp. 41, 48, 50).
- [52] M. Fowler, J. Lewis. *Microservices - a definition of this new architectural term*. Mar. 2014. URL: <https://martinfowler.com/articles/microservices.html> (cit. on p. 14).
- [53] P. D. Francesco, I. Malavolta, P. Lago. “Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption”. In: Institute of Electrical and Electronics Engineers Inc., May 2017, pp. 21–30. ISBN: 9781509057290. DOI: [10.1109/ICSA.2017.24](https://doi.org/10.1109/ICSA.2017.24) (cit. on pp. 29, 30, 39).
- [54] A. F. A. Freire, A. F. Sampaio, L. H. L. Carvalho, O. Medeiros, N. C. Mendonça. “Migrating production monolithic systems to microservices using aspect oriented programming”. In: *Software: Practice and Experience* 51.6 (2021), pp. 1280–1307 (cit. on pp. 41, 49, 50).
- [55] F. Freitas, A. Ferreira, J. Cunha. “Refactoring Java Monoliths into Executable Microservice-Based Applications”. In: *25th Brazilian Symposium on Programming Languages*. 2021, pp. 100–107 (cit. on pp. 41, 48).
- [56] J. Fritzs. *Application Modernization: Refactoring to Microservice-based Systems, Doctoral project*. <https://www.iste.uni-stuttgart.de/ese/research/>. Institute of Software Engineering, University of Stuttgart (cit. on pp. 24, 25, 89).
- [57] J. Fritzs. *Fork of the Repository “Architecture Refactoring Helper” tool*. 2022. URL: <https://github.com/jfr609/architecture-refactoring-helper> (cit. on pp. 66, 78, 90).
- [58] J. Fritzs, J. Bogner, A. Zimmermann, S. Wagner. *From Monolith to Microservices: A Classification of Refactoring Approaches* (cit. on pp. 12, 40, 41).
- [59] J. Fritzs, D. Koch. *Discussions about the requirements for the implementation of a quality model into the “Architecture Refactoring Helper” tool*. Institute of Software Engineering, University of Stuttgart (cit. on p. 63).
- [60] M. Gao, M. Chen, A. Liu, W. H. Ip, K. L. Yung. “Optimization of microservice composition based on artificial immune algorithm considering fuzziness and user preference”. In: *Ieee Access* 8 (2020), pp. 26385–26404 (cit. on pp. 41, 48, 49).
- [61] A. Glen. *Microservices Priorities and Trends*. July 2018. URL: <https://dzone.com/articles/dzone-research-microservices-priorities-and-trends> (cit. on p. 27).
- [62] M. Gysel, L. Kölbener, W. Giersche, O. Zimmermann. “Service cutter: A systematic approach to service decomposition”. In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2016, pp. 185–200 (cit. on pp. 41, 48, 50).

- [63] S. Habibullah, X. Liu, Z. Tan. “An approach to evolving legacy enterprise system to microservice-based architecture through feature-driven evolution rules”. In: *International Journal of Computer Theory and Engineering* 10.5 (2018) (cit. on pp. 41, 48, 50).
- [64] T. Haller. “Design, implementation and evaluation of an application for guiding architectural refactoring to microservices”. MA thesis. 2022. DOI: [10.18419/opus-12272](https://doi.org/10.18419/opus-12272) (cit. on pp. 13, 17, 18, 62, 63, 89).
- [65] T. Haller. *Repository for the “Architecture Refactoring Helper” tool*. 2022. URL: <https://github.com/T-Haller/architecture-refactoring-helper> (cit. on pp. 13, 17, 18, 62, 66, 78, 85, 86, 89).
- [66] S. Hassan, N. Ali, R. Bahsoon. “Microservice ambients: An architectural meta-modelling approach for microservice granularity”. In: *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2017, pp. 1–10 (cit. on pp. 41, 48, 50).
- [67] S. G. Haugeland, P. H. Nguyen, H. Song, F. Chauvel. “Migrating Monoliths to Microservices-based Customizable Multi-tenant Cloud-native Apps”. In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2021, pp. 170–177 (cit. on pp. 41, 48, 50).
- [68] A. Henry, Y. Ridene. “Migrating to microservices”. In: *Microservices*. Springer, 2020, pp. 45–72 (cit. on p. 15).
- [69] R. C. A. Hutapea, A. P. Wahyudi. *Design Quality Measurement for Service Oriented Software on Service Computing System: a Systematic Literature Review; Design Quality Measurement for Service Oriented Software on Service Computing System: a Systematic Literature Review*. 2018. ISBN: 978-1-5386-5692-1 (cit. on pp. 29, 36).
- [70] “IEEE Standard Glossary of Software Engineering Terminology”. In: *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: [10.1109/IEEESTD.1990.101064](https://doi.org/10.1109/IEEESTD.1990.101064) (cit. on p. 47).
- [71] International Organization for Standardization/International Electrotechnical Commission. *ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. 2011 (cit. on pp. 22, 23, 29, 31–35, 89).
- [72] International Organization for Standardization/International Electrotechnical Commission/Institute of Electrical and Electronics Engineers. *ISO/IEC/IEEE 24765 - Systems and software engineering - Vocabulary - Second edition*. Sept. 2017 (cit. on pp. 35, 36).
- [73] N. Ivanov, A. Tasheva. “A Hot Decomposition Procedure: Operational Monolith System to Microservices”. In: *2021 International Conference Automatics and Informatics (ICAI)*. IEEE, pp. 182–187 (cit. on pp. 41, 48, 76).
- [74] P. Jamshidi, C. Pahl, N. C. Mendonça. “Pattern-based multi-cloud architecture migration”. In: *Software: Practice and Experience* 47.9 (2017), pp. 1159–1184 (cit. on pp. 41, 48).
- [75] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, Q. Zheng. “Service candidate identification from monolithic systems based on execution traces”. In: *IEEE Transactions on Software Engineering* 47.5 (2019), pp. 987–1007 (cit. on pp. 41, 48).
- [76] W. Jin, T. Liu, Q. Zheng, D. Cui, Y. Cai. “Functionality-oriented microservice extraction based on execution trace clustering”. In: *2018 IEEE International Conference on Web Services (ICWS)*. IEEE. 2018, pp. 211–218 (cit. on pp. 41, 48, 50).

- [77] M. I. Josélyne, D. Tuheirwe-Mukasa, B. Kanagwa, J. Balikuddembe. “Partitioning microservices: a domain engineering approach”. In: *Proceedings of the 2018 International Conference on Software Engineering in Africa*. 2018, pp. 43–49 (cit. on pp. 41, 48, 50).
- [78] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, D. Banerjee. “Mono2micro: a practical and effective tool for decomposing monolithic java applications to microservices”. In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2021, pp. 1214–1224 (cit. on pp. 41, 50).
- [79] M. Kalske, N. Mäkitalo, T. Mikkonen. “Challenges When Moving from Monolith to Microservice Architecture”. In: vol. 10544 LNCS. Springer Verlag, 2018, pp. 32–47. ISBN: 9783319744322. DOI: [10.1007/978-3-319-74433-9_3](https://doi.org/10.1007/978-3-319-74433-9_3) (cit. on pp. 14, 15).
- [80] M. Kamimura, K. Yano, T. Hatano, A. Matsuo. “Extracting candidates of microservices from monolithic application code”. In: *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2018, pp. 571–580 (cit. on p. 41).
- [81] M. Kasunic. *Designing an effective survey*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2005 (cit. on p. 79).
- [82] J. Kazanavičius, D. Mažeika. “Migrating legacy software to microservices architecture”. In: *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*. IEEE. 2019, pp. 1–5 (cit. on p. 14).
- [83] J. Kazanavičius, D. Mažeika, D. Kalibatienė. “An Approach to Migrate a Monolith Database into Multi-Model Polyglot Persistence Based on Microservice Architecture: A Case Study for Mainframe Database”. In: *Applied Sciences* 12.12 (2022), p. 6189 (cit. on pp. 41, 48, 49).
- [84] R. Kazman, M. Klein, P. Clements. *ATAM: Method for Architecture Evaluation*. Tech. rep. CMU/SEI-2000-TR-004. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000 (cit. on pp. 12, 28).
- [85] G. Kecskemeti, A. C. Marosi, A. Kertesz. “The ENTICE approach to decompose monolithic services into microservices”. In: *2016 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 2016, pp. 591–596 (cit. on pp. 41, 48).
- [86] S. Klock, J. M. E. Van Der Werf, J. P. Guelen, S. Jansen. “Workload-based clustering of coherent feature sets in microservice architectures”. In: *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2017, pp. 11–20 (cit. on pp. 39, 41, 48).
- [87] H. Knoche. “Sustaining runtime performance while incrementally modernizing transactional monolithic software towards microservices”. In: *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. 2016, pp. 121–124 (cit. on p. 41).
- [88] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, D. Kröger. “Microservice Decomposition via Static and Dynamic Analysis of the Monolith”. In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. 2020, pp. 9–16 (cit. on p. 41).
- [89] A. Levcovitz, R. Terra, M. Valente. “Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems”. In: (May 2016) (cit. on pp. 41, 48, 50).
- [90] 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. 41, 48).

- [91] J. Li, H. Xu, X. Xu, Z. Wang. “A Novel Method for Identifying Microservices by Considering Quality Expectations and Deployment Constraints”. In: *International Journal of Software Engineering and Knowledge Engineering* 32.03 (2022), pp. 417–437 (cit. on pp. 41, 50).
- [92] S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, J. Ge, Z. Shan. “A dataflow-driven approach to identifying microservices from monolithic applications”. In: *Journal of Systems and Software* 157 (2019), p. 110380 (cit. on pp. 41, 50).
- [93] S. Li, H. Zhang, Z. Jia, C. Zhong, C. Zhang, Z. Shan, J. Shen, M. A. Babar. “Understanding and addressing quality attributes of microservices architecture: A Systematic literature review”. In: *Information and Software Technology* 131 (Mar. 2021), p. 106449. issn: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2020.106449> (cit. on pp. 12, 16, 27, 29, 32, 34, 35, 37, 39, 42, 43, 50–52).
- [94] R. Lichtenthäler, G. Wirtz. “Towards a Quality Model for Cloud-native Applications”. In: vol. 13226 LNCS. Springer Science and Business Media Deutschland GmbH, 2022, pp. 109–117. ISBN: 9783031047176. DOI: [10.1007/978-3-031-04718-3_7](https://doi.org/10.1007/978-3-031-04718-3_7) (cit. on pp. 12, 16, 25, 26, 29, 32, 43, 53–55).
- [95] J. Lin, L. C. Lin, S. Huang. “Migrating web applications to clouds with microservice architectures”. In: *2016 International conference on applied system innovation (ICASI)*. IEEE. 2016, pp. 1–4 (cit. on p. 41).
- [96] J. Löhnertz, A.-M. Oprescu. “Steinmetz: Toward Automatic Decomposition of Monolithic Software Into Microservices.” In: *SATToSE*. 2020 (cit. on pp. 41, 48, 50).
- [97] S.-P. Ma, C.-Y. Fan, Y. Chuang, W.-T. Lee, S.-J. Lee, N.-L. Hsueh. “Using service dependency graph to analyze and test microservices”. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 2. IEEE. 2018, pp. 81–86 (cit. on p. 36).
- [98] S. A. Maisto, B. D. Martino, S. Nacchia. “From monolith to cloud architecture using semi-automated microservices modernization”. In: *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. Springer. 2019, pp. 638–647 (cit. on pp. 41, 48).
- [99] G. Márquez, H. Astudillo. “Actual use of architectural patterns in microservices-based open source projects”. In: *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. Ieee. 2018, pp. 31–40 (cit. on pp. 42, 43, 50–52).
- [100] A. Mathai, S. Bandyopadhyay, U. Desai, S. Tamilselvam. “Monolith to Microservices: Representing Application Software through Heterogeneous GNN”. In: *arXiv preprint arXiv:2112.01317* (2021) (cit. on pp. 41, 48, 50).
- [101] B. Mayer, R. Weinreich. “An approach to extract the architecture of microservice-based software systems”. In: *2018 IEEE symposium on service-oriented system engineering (SOSE)*. IEEE. 2018, pp. 21–30 (cit. on p. 41).
- [102] A. Messina, R. Rizzo, P. Storniolo, M. Tripiciano, A. Urso. “The database-is-the-service pattern for microservice architectures”. In: *International Conference on Information Technology in Bio-and Medical Informatics*. Springer. 2016, pp. 223–233 (cit. on p. 43).
- [103] M. Milić, D. Makajić-Nikolić. “Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures”. In: *Symmetry* 14.9 (2022), p. 1824 (cit. on pp. 29, 32–36, 39).

- [104] R. Nakazawa, T. Ueda, M. Enoki, H. Horii. “Visualization tool for designing microservices with the monolith-first approach”. In: *2018 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE. 2018, pp. 32–42 (cit. on pp. 41, 48).
- [105] S. Newman. *Building Microservices: Designing Fine-Grained Systems*. O’Reilly Media, Inc., 2015. ISBN: 978-1-491-95035-7 (cit. on p. 14).
- [106] J. Nielsen. *Why You Only Need to Test with 5 Users*. 2000. URL: <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/> (cit. on p. 79).
- [107] L. Nunes, N. Santos, A. Rito Silva. “From a monolith to a microservices architecture: An approach based on transactional contexts”. In: *European Conference on Software Architecture*. Springer. 2019, pp. 37–52 (cit. on p. 41).
- [108] J. Park, D. Kim, K. Yeom. “An approach for reconstructing applications to develop container-based microservices”. In: *Mobile Information Systems 2020 (2020)* (cit. on pp. 41, 48).
- [109] M. Perepletchikov, C. Ryan, K. Frampton. “Cohesion metrics for predicting maintainability of service-oriented software”. In: *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE. 2007, pp. 328–335 (cit. on p. 37).
- [110] K. Perera, I. Perera. “A rule-based system for automated generation of serverless-microservices architecture”. In: *2018 IEEE International Systems Engineering Symposium (ISSE)*. IEEE. 2018, pp. 1–8 (cit. on pp. 41, 48, 50).
- [111] I. Pigazzini, F. Arcelli Fontana, A. Maggioni. “Tool support for the migration to microservice architecture: An industrial case study”. In: *European Conference on Software Architecture*. Springer. 2019, pp. 247–263 (cit. on pp. 41, 50).
- [112] F. Ponce, G. Márquez, H. Astudillo. “Migrating from monolithic architecture to microservices: A Rapid Review”. In: *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE. 2019, pp. 1–7 (cit. on pp. 12, 40, 41).
- [113] G. Procaccianti, B. van der Sanden, G. Li, Z. Luo, B. Hertzberger, M. Kopershoek, A. Oprescu. “Towards a MicroServices Architecture for Clouds”. In: *VU University Amsterdam (2016)* (cit. on p. 41).
- [114] S. Pulnil, T. Senivongse. “A Microservices Quality Model Based on Microservices Anti-patterns”. In: IEEE, June 2022, pp. 1–6. ISBN: 978-1-6654-8510-4. DOI: [10.1109/JCSSE54890.2022.9836297](https://doi.org/10.1109/JCSSE54890.2022.9836297) (cit. on pp. 29, 43, 53–55).
- [115] V. Raj, S. Ravichandra. “A service graph based extraction of microservices from monolith services of service-oriented architecture”. In: *Software: Practice and Experience (2022)* (cit. on pp. 41, 48, 50).
- [116] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, T. Huang. “Migrating web applications from monolithic structure to microservices architecture”. In: Association for Computing Machinery, Sept. 2018. ISBN: 9781450365901. DOI: [10.1145/3275219.3275230](https://doi.org/10.1145/3275219.3275230) (cit. on p. 14).
- [117] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, T. Huang. “Migrating web applications from monolithic structure to microservices architecture”. In: *Proceedings of the tenth asia-pacific symposium on internetware*. 2018, pp. 1–10 (cit. on pp. 41, 48, 50).
- [118] C. Richardson. *A pattern language for microservices*. URL: <https://microservices.io/patterns/index.html> (cit. on pp. 42, 43, 50–52).

- [119] C. Richardson. *Introduction to Microservices*. May 2015. URL: <https://www.nginx.com/blog/introduction-to-microservices/> (cit. on p. 30).
- [120] N. F. Rodrigues, L. S. Barbosa. “Component identification through program slicing”. In: *Electronic Notes in Theoretical Computer Science* 160 (2006), pp. 291–304 (cit. on pp. 41, 50).
- [121] I. Saidani, A. Ouni, M. W. Mkaouer, A. Saied. “Towards automated microservices extraction using multi-objective evolutionary search”. In: *International Conference on Service-Oriented Computing*. Springer. 2019, pp. 58–63 (cit. on pp. 41, 50).
- [122] A. Santos, H. Paula. “Microservice decomposition and evaluation using dependency graph and silhouette coefficient”. In: *15th Brazilian Symposium on Software Components, Architectures, and Reuse*. 2021, pp. 51–60 (cit. on pp. 41, 50).
- [123] N. Santos, C. E. Salgado, F. Morais, M. Melo, S. Silva, R. Martins, M. Pereira, H. Rodrigues, R. J. Machado, N. Ferreira, et al. “A logical architecture design method for microservices architectures”. In: *Proceedings of the 13th European Conference on Software Architecture-Volume 2*. 2019, pp. 145–151 (cit. on pp. 41, 48).
- [124] Sarita, S. Sebastian. “Transform Monolith into Microservices using Docker”. In: Institute of Electrical and Electronics Engineers Inc., Sept. 2018. ISBN: 9781538640081. DOI: [10.1109/ICCUBEA.2017.8463820](https://doi.org/10.1109/ICCUBEA.2017.8463820) (cit. on p. 14).
- [125] A. Sayara, M. S. Towhid, M. S. Hossain. “A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling”. In: *2017 20th International Conference of Computer and Information Technology (ICCIIT)*. IEEE. 2017, pp. 1–6 (cit. on pp. 41, 50).
- [126] T. Schirgi. *Architectural Quality Attributes for the Microservices of CaRE*. 2021 (cit. on p. 29).
- [127] T. Schirgi, E. Brenner. “Quality assurance for microservice architectures”. In: *2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE. 2021, pp. 76–80 (cit. on pp. 43, 53, 54, 86).
- [128] C. Schroer, S. Wittfoth, J. M. Gomez. “A Process Model for Microservices Design and Identification”. In: Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 38–45. ISBN: 9781665439107. DOI: [10.1109/ICSA-C52384.2021.00013](https://doi.org/10.1109/ICSA-C52384.2021.00013) (cit. on pp. 27, 28).
- [129] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, L. Veiga. “Practical service placement approach for microservices architecture”. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2017, pp. 401–410 (cit. on pp. 41, 48, 50).
- [130] K. Sellami, M. A. Saied, A. Ouni. “A hierarchical DBSCAN method for extracting microservices from monolithic applications”. In: *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*. 2022, pp. 201–210 (cit. on pp. 41, 48, 50).
- [131] A. Selmadji, A.-D. Seriai, H. L. Bouziane, C. Dony, R. O. Mahamane. “Re-architecting oo software into microservices”. In: *European Conference on Service-Oriented and Cloud Computing*. Springer. 2018, pp. 65–73 (cit. on pp. 41, 50).

- [132] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. O. Mahamane, P. Zaragoza, C. Dony. “From monolithic architecture style to microservice one based on a semi-automatic approach”. In: *2020 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2020, pp. 157–168 (cit. on pp. 41, 48, 50).
- [133] A. Shashwat, D. Kumar. “A service identification model for service oriented architecture”. In: *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*. IEEE. 2017, pp. 1–5 (cit. on pp. 41, 48, 49).
- [134] A. Shimoda, T. Sunada. “Priority order determination method for extracting services stepwise from monolithic system”. In: *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*. IEEE. 2018, pp. 805–810 (cit. on p. 41).
- [135] N. Singhal, U. Sakthivel, P. Raj. “Efficient Hybrid Research for QoS-Aware Microservice Compostion”. In: *International Journal of Recent Technology and Engineering* 8.2 (2019), pp. 5251–5255 (cit. on pp. 41, 48).
- [136] G. Starke. *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. 3rd ed. Carl Hanser Verlag München, 2008. ISBN: 978-3-446-41215-6 (cit. on pp. 28, 29).
- [137] S. X. Sun, J. Zhao. “A decomposition-based approach for service composition with global QoS guarantees”. In: *Information Sciences* 199 (2012), pp. 138–153 (cit. on pp. 33, 35).
- [138] D. Taibi, V. Lenarduzzi, C. Pahl. “Architectural patterns for microservices: a systematic mapping study”. In: *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*. SciTePress. 2018 (cit. on pp. 42, 43, 50–52, 55, 56).
- [139] D. Taibi, V. Lenarduzzi, C. Pahl. “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation”. In: *IEEE Cloud Computing* 4 (5 Sept. 2017), pp. 22–32. ISSN: 23256095. DOI: [10.1109/MCC.2017.4250931](https://doi.org/10.1109/MCC.2017.4250931) (cit. on pp. 12, 15).
- [140] D. Taibi, K. Systä. “From monolithic systems to microservices: A decomposition framework based on process mining”. In: (2019) (cit. on pp. 41, 50).
- [141] R. Terra, M. T. Valente, R. S. Bigonha. “An approach for extracting modules from monolithic software architectures”. In: *IX Workshop de Manutenção de Software Moderna (WMSWM)*. 2012, pp. 1–18 (cit. on pp. 41, 50).
- [142] M. Tusjunt, W. Vatanawood. “Refactoring orchestrated web services into microservices using decomposition pattern”. In: *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*. IEEE. 2018, pp. 609–613 (cit. on p. 41).
- [143] S. Tyszberowicz, R. Heinrich, B. Liu, Z. Liu. “Identifying microservices using functional decomposition”. In: *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer. 2018, pp. 50–65 (cit. on pp. 41, 50).
- [144] J. A. Valdivia, X. Limon, K. Cortes-Verdin. “Quality attributes in patterns related to microservice architecture: A Systematic Literature Review”. In: Institute of Electrical and Electronics Engineers Inc., Oct. 2019, pp. 181–190. ISBN: 9781728125244. DOI: [10.1109/CONISOFT.2019.00034](https://doi.org/10.1109/CONISOFT.2019.00034) (cit. on pp. 12, 16, 27, 29, 32, 42, 43, 50–52).

- [145] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, J. O. Ocharán-Hernández. “Patterns related to microservice architecture: a multivocal literature review”. In: *Programming and Computer Software* 46.8 (2020), pp. 594–608 (cit. on pp. 12, 42, 43, 50–52).
- [146] G. Vale, F. F. Correia, E. M. Guerra, T. de Oliveira Rosa, J. Fritzsich, J. Bogner. “Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs”. In: (Jan. 2022). URL: <http://arxiv.org/abs/2201.03598> (cit. on pp. 16, 42, 43, 50–52, 55, 56).
- [147] F. H. Vera-Rivera, E. Puerto, H. Astudillo, C. M. Gaona. “Microservices Backlog—A Genetic Programming Technique for Identification and Evaluation of Microservices From User Stories”. In: *IEEE Access* 9 (2021), pp. 117178–117203 (cit. on pp. 41, 48, 50).
- [148] M. Waseem, P. Liang, G. Márquez, M. Shahin, A. A. Khan, A. Ahmad. “A decision model for selecting patterns and strategies to decompose applications into microservices”. In: *International Conference on Service-Oriented Computing*. Springer. 2021, pp. 850–858 (cit. on p. 16).
- [149] Y. Wei, Y. Yu, M. Pan, T. Zhang. “A Feature Table approach to decomposing monolithic applications into microservices”. In: *12th Asia-Pacific Symposium on Internetware*. 2020, pp. 21–30 (cit. on pp. 41, 48, 50).
- [150] E. Wolff. *Das Microservices-Praxisbuch: Grundlagen, Konzepte und Rezepte*. dpunkt.verlag, 2018 (cit. on pp. 26, 43).
- [151] J. Yu, J. Ge, J. Sun. “Research on Quality Model and Measurement for Microservices”. In: *Proceedings of the 3rd International Workshop on Experience with SQuaRE Series and Its Future Direction (IWESQ 2021) co-located with 28th Asia-Pacific Software Engineering Conference (APSEC 2021), Virtual Event, Taipei, Taiwan, Dec 8, 2021*. Ed. by T. Nakajima, T. Komiyama. Vol. 3114. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pp. 5–10 (cit. on pp. 16, 33, 34, 36, 37).
- [152] P. Zaragoza, A.-D. Seriai, A. Seriai, H.-L. Bouziane, A. Shatnawi, M. Derras. “Refactoring Monolithic Object-Oriented Source Code to Materialize Microservice-oriented Architecture.” In: *ICSOF* 117 (2021), p. 126 (cit. on pp. 41, 48).
- [153] Y. Zhang, B. Liu, L. Dai, K. Chen, X. Cao. “Automated microservice identification in legacy systems with functional and non-functional metrics”. In: *2020 IEEE international conference on software architecture (ICSA)*. IEEE. 2020, pp. 135–145 (cit. on pp. 41, 50).
- [154] J. Zhao, K. Zhao. “Applying Microservice Refactoring to Object-oriented Legacy System”. In: *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE. 2021, pp. 467–473 (cit. on pp. 41, 50).
- [155] O. Zimmermann. “Architectural refactoring for the cloud: a decision-centric view on cloud migration”. In: *Computing* 99 (2 Feb. 2017), pp. 129–145. ISSN: 0010485X. DOI: [10.1007/s00607-016-0520-y](https://doi.org/10.1007/s00607-016-0520-y) (cit. on p. 28).

All links were last followed on December 19, 2022.

A Survey Material

A.1 Qualitative survey for quality model extension of the ‘Architecture Refactoring Helper’ tool - Description

Dear participant,

this qualitative survey aims to investigate the usability of an extension of the tool “Architecture Refactoring Helper” from the Institute of Software Engineering (University of Stuttgart) [<https://github.com/T-Haller/architecture-refactoring-helper>]. In general, the tool is intended to support practitioners in migrating from monolithic architectures to a microservice architecture. Different approaches for the migration based on individual and predefined inputs are recommended to the users. Part of the master’s thesis “Migrating Monolithic Architectures to Microservices: A Study on Software Quality Attributes” by Daniel Koch is to extend this already existing tool with a quality model. In particular, this means that different migration approaches and architectural patterns for a microservice architecture can be proposed based on the results from the research of the master thesis. In this respect, the extension can support the practitioner in making a suitable choice for migration approaches and/or architecture patterns and best practices by providing quality attributes when defining scenarios according to the ATAM method (<https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=5177>) or by manual configuration.

A.1.1 Privacy Policy

Before answering the questions, please agree that the information from this survey may be used anonymously in the master thesis “Migrating Monolithic Architectures to Microservices: A Study on Software Quality Attributes” and possibly used in future publications of the University of Stuttgart. To do so, please briefly answer whether you agree to the above conditions and whether the answers may be included in the thesis partially anonymized word by word.

1. I accept the conditions in the sense that

- The information I have provided may be used completely and anonymously.
- The information I have provided may be used anonymously, but no word-for-word information will be used.

A.1.2 Personal Information

2. Professional experience in years:
3. Experience with microservices in years:
4. Experience migrating to microservices in years:
5. Current job title/area of responsibility:

A.1.3 Exploration of the extension - Task description

For the exploration and subsequent evaluation, you will be given two tasks that cover the scope of the extension and help you discover its functionality. For this purpose, the individual steps and inputs are first described to guide you through the tool. After that, usability statements are presented, which you can then use to agree or disagree and give reasons for your decision. The tool can be accessed at: [LINK].

Task 1) - Using an example scenario to find suitable migration approaches and target architecture patterns

1. In this first task, you want to get scenario-based recommendations for your system. To do this, navigate to Phase 1: System Comprehension and create two example scenarios in step 3 using the ATAM method. As an example, you want a scenario 'Parallel Users', which handles a system with > 5000 parallel users with a low response time, so you want to optimize the performance. This scenario has a very high importance and a high difficulty. Another scenario ('Moving the system') describes that you may need to move the system more often, so a high Deployability (under Portability) is needed. The importance is medium and the difficulty is very high.
2. Please confirm the entries with 'Save Changes'.
3. After that, you want to get scenario based recommendations for migration approaches. For this, you can go to phase 2 manually or skip directly to phase 2 in phase 1. Select the scenario-based method here with 'Find scenario-based recommended approaches'.
4. It crosses your mind that you also want to consider the degree of cohesion in the migration, so you want to further configure the filters. To do this, press 'Configure Filters' and include cohesion in the configuration.
5. Press 'Search' to get the results.
6. Browse the overview of the results.
7. You would also like to get recommended patterns and best practices for your target architecture based on the inputs you have made. For this, you can skip directly to phase 3 with 'Show Patterns and Best Practices'.
8. Browse the overview of the results.

Task 2) - Manual configuration

1. You now want to perform a manual configuration of the quality attributes.

2. To do this, navigate to phase 2 and press 'Find recommended approaches'.
3. Via the 'Configure' button you can open the manual configuration.
4. You do not want to consider the applied input of 'Cohesion' in this case, but you would like to consider all approaches that optimize scalability and handle coupling.
5. Press 'Search' to get the results.
6. Explore the results and, if appropriate, patterns and best practices.

Feel free to explore the tool independently of the tasks. However, in the context of this work, the main focus is on the quality attributes and system properties in terms of recommendations for the migration approaches and for the target architecture patterns and best practices.

Usability Evaluation - Description and Statements

After you have explored the extension, we would like to ask you a few usability statements that you can agree or disagree with. In the best case you should give a reason for each of them. Please keep in mind that the current state of the tool is a prototype and that the data and number of approaches and patterns are not final, but should serve as examples. Please answer in German or English.

Phase 1

6. It was intuitive and easy to understand how I could create and edit scenarios.
7. It was intuitive and easy to understand how I could assign quality attributes to scenarios.
8. The instructions as text about the scenarios helped me to understand what I needed to do.
9. The structure of the user interface is useful and easy to understand.

Phase 2

10. The overview of the configuration on the right side is easy to understand and helpful.
11. The selection of the different modes is straightforward.
12. The configuration of additional filters is helpful.
13. The result view, especially in terms of qualities helps to find suitable approaches.
14. The indication of the tendency based on the qualities with the number of matches and the directed arrows is sensibly chosen.
15. It is comprehensible how the rankings of the migration approaches are generated.
16. The chosen type of the presentation of the results is generally helpful.

Phase 3

17. The selection of the different modes is straightforward.
18. The configuration of additional filters is helpful.
19. The result view, especially in terms of qualities helps to find suitable patterns and best practices.
20. The indication of the tendency based on the qualities with the number of matches and the directed arrows is sensibly chosen.
21. It is comprehensible how the rankings of the patterns and best practices are generated.
22. The chosen type of the presentation of the results is generally helpful.

General

23. The operation and navigation are intuitive and easy to follow.
23. The extension can support the migration in the context of quality goals.
25. Further comments - Do you have any further comments or suggestions for improvement which concern usability?

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