The present work was submitted to the Research Group Software Construction

of the Faculty of Mathematics, Computer Science, and Natural Sciences

Master Thesis

Investigating Quality Attributes and Best Practices of Microservices Architectures

presented by

Faizan Zafar

Aachen, November 7, 2022

Examiner

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisor

Alex Sabau, M.Sc.

Acknowledgment

First and foremost, I would like to offer my profound thanks to my esteemed supervisor, Alex Sabau, for guiding and supporting me consistently throughout the whole thesis process as well as for providing me with valuable feedback every step of the way.

I would also like to extend my kind regards to Prof. Dr. Horst Lichter and Prof. Dr. Bernhard Rumpe for examining this thesis and for providing insightful suggestions and feedback in the duration of the thesis.

Next, I would like to convey my sincere thanks to the evaluation interview participants who agreed to provide their precious time and helpful feedback that proved to be very important for the results of the thesis.

Last but not the least, I would like to express my immense gratitude to my dearest friends and family from near and far, especially my brother. I would like to appreciate their constant moral support and continuous encouragement in not only the period of the thesis but my entire masters study.

Faizan Zafar

Abstract

Software systems that are built in accordance with the Microservice Architecture pattern present viable solutions to modern problems in the rapidly evolving world of software engineering, particularly for industrial applications. This architectural pattern offers various benefits such as increased scalability, low coupling, high resilience, independent deployment, and greater agility compared to other architectural patterns. However, this highly distributed architectural paradigm poses novel challenges in essential software engineering activities such as quality assurance.

Due to the high complexity of building systems by following the Microservice Architecture pattern, design guidelines, including best practices, design patterns, and design principles, are employed by software architects and developers which help in constructing microservices applications in the most optimal ways. However, knowledge around such guidelines is widely dispersed and unorganised. As a result, there is a lack of a structured knowledge base of design guidelines that can be used as a reference guide for designing Microservices Architectures.

In addition, with respect to the quality assurance of microservices systems, there is a lack of comprehensiveness in the industry as well as academia concerning the quality of Microservices Architectures, including how it can be conceptualised and described. In relation to this, there is very limited information available on how the design guidelines affect the quality characteristics of a microservices application. As a result, there is a lack of standardised concepts that can enable targeted quality assurance of Microservices Architectures.

In this thesis, we create and structure a catalogue of 239 design guidelines, including best practices, design patterns, and design principles, that are used by software architects and developers to build Microservices Architectures. We also determine a set of quality characteristics that are related to each of the existing design guidelines. Additionally, we propose and evaluate a Quality Model which outlines and explains 70 quality characteristics that mutually and meaningfully describe the quality of microservices applications and architectures.

Contents

List of Tables

List of Figures

List of Source Codes

1. Introduction

This chapter presents the background and concepts of the current research that serve as the basis for understanding the approaches and results of the thesis. Lastly, it outlines the structure of this thesis document.

1.1. Background

Software systems are constructed to satisfy organisations' business goals, and the system *architecture* is a bridge between those business goals and the final resulting system [BCK12]. The *architecture* of a software system refers to the structure of its elements, their interrelationships, and the principal design decisions made during the system's development and any subsequent evolution [BCK12]. Especially when constructing large and complex systems, architects must build the entire software architecture using the most appropriate architectural paradigm in order to ensure smooth operations. *Microservices Architectures (MSA)* is a software architecture style that structures a software application into several individual *microservices* narrowly aligned to the business domain, instead of a single, strongly-coupled application utilising a monolithic architecture [Ric17]. *Microservices* are a suite of small services, each running on its own process and intercommunicating with lightweight mechanisms [OMA18]. Key characteristics of microservices include; small and focused on doing one thing well, autonomous, composability, technology heterogeneity, resilience, independent scaling, ease of deployment, and organisational alignment [Neu15].

In practice, microservices are usually developed in a decentralised manner and operated on by specialised teams from different organisational units [Ric17]. The individual microservices are loosely coupled and independently deployable often running on geographically distributed resources [Ric17]. MSA are deemed as an evolution of *Service-oriented Architectures (SOA)*. SOA is a non-centralised architectural paradigm based on the communication between services in order to offer functionality, including consuming third-party services [MT10]. However, SOA pose challenges in communication and level of granularity and MSA offer to mitigate those challenges [Neu15].

Architectural design plays an important role in the success of a software system in terms of customer satisfaction. With the MSA pattern, the development complexity of an individual component is decreased. However, the increasing amount of components and continuous changes in the microservices system lead to increase in system complexity. Another core aspect is that microservices applications are distributed systems by nature, thus operational complexity for managing a distributed system is added. In order to cope with this complexity, *design guidelines* have been researched upon in academia and employed by architects and developers in industry for several years.

Definition 1. *A* Design Guideline *is a fundamental directive which maintains certain rules and responsibilities, and impacts a specified scope of the system architecture.*

Such guidelines are not only followed and incorporated during the design phase, but also during the development phase of a microservices system. Design guidelines provide architects and developers with directions and advice for constructing MSA since following a particular guideline standardises the solution to a particular problem, and contributes to one or more aspects of the system quality. However, there exist numerous different advises labelled as design guidelines in industry and academia. In this work, we want to systematically collect the design guidelines and develop a structured catalogue for the same. For the purposes of this thesis, we refer to *design guidelines* as an umbrella term covering the *best practices*, *design patterns*, and *design principles* used to build MSA. These three individual concepts are defined and explained later in Chapter 6.1.

The concept of *quality* is often used in daily life and is perceived as self-explanatory and obvious [Bev95]. However, there are multi-faceted notions of the concept of quality from different views [GQ84]. According to ISO 9000 which is the European standard for quality management systems, quality is the degree to which a set of inherent characteristics of a product, software or service fulfills requirements [Sta15]. Similarly, ISO/IEC 25010, which is the international standard for system and software quality models, explains the quality of a system as the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value [ISO11]. Typically, software companies gain revenue by retaining existing customers and gaining more customers [MSM16]. Customer satisfaction plays a vital role in customer retention, and can be achieved by providing a service that creates value for the customer. This eventually leads the service provider to establish a loyal relationship with the customer. In order to achieve customer satisfaction, software companies continuously strive to understand and implement guidelines that would help them in delivering high quality services to their stakeholders. Hence, quality is of utmost importance in any kind of software system and should be incorporated from the initial design phases such as code architecture design and user interface design [Kas17].

Architectural drivers have a significant influence over the design of a microservices system. The need to improve system quality, and technical and business constraints constitute such drivers. The architectural decisions made during the design phase influence the *quality characteristics* of a microservices system.

Definition 2. *A* Quality Characteristic *is an intrinsic property of a product, software or service that gives it the ability to satisfy stakeholder requirements.*

Such characteristics are required to be achieved to a certain level in order to ensure system quality in a specific direction. The requirement to reach a certain level of a particular quality characteristic impacts the architecture and influences the architectural design decisions made in a microservices system [ISO11].

The ISO/IEC 25010 standard defines a *Quality Model* as a conceptual construct to describe the quality of a product, software or service as a decomposition into characteristics and sub-characteristics. A *Quality Model* enables to assess and predict the quality of a product, software or service by the operationalisation of its quality characteristics [Wag13]. The quality of services in microservices systems typically considers technical and behavioural aspects such as *reliability*, *maintainability* and *performance* in order to satisfy customer needs [BCK12], [Di 17], [Li17]. However, the understanding of quality in MSA is considered incomplete primarily due to lack of research around it, with very few known approaches partially tackling it. In this work, we want to formulate a *Quality Model (QM)* tailored to microservices applications by investigating, structuring and describing several quality characteristics relevant to MSA.

The decisions made around the architecture of a microservices system should realistically take into account important aspects such as the trade-offs for each design guideline, the balance between the different quality characteristics specified for the system, and the benefits offered by the design guideline under consideration. Since MSA are an emerging innovation, the relationship between the design guidelines and the quality characteristics is not clearly defined. This is a point of concern from a software engineering standpoint since such understanding is fundamental and crucial to correctly design applications using MSA. In this work, we want to determine the quality characteristics which are related to the MSA design guidelines.

1.2. Thesis Structure

This thesis is organised into eight chapters. Chapter 1 consists of the introduction and background of the research domain so that the problem and related work are clearly understood. Chapter 2 consists of the motivation put forward, the research questions put together, and the scope defined for the thesis. Chapter 3 consists of the prior work related to the research domain. Chapter 4 consists of the research methodology that was adopted in this thesis. Chapter 5 consists of the proposed Quality Model as a prime outcome of the thesis, including the evaluation method applied. Chapter 6 consists of the proposed Guidelines Catalogue as a major artifact of the thesis. Chapter 7 consists of a discussion on the thesis results, research questions, implications, and threats to validity. Chapter 8 consists of the summary of the current work and the outline of the future work.

2. Problem Statement

This chapter discusses the need and motivation for this research. It outlines the research questions that we define and tackle as part of this thesis. Lastly, it specifies the scope of this work.

2.1. Motivation

From their conception in 2014 to date, MSA have become the subject of increased industrial application and academic research in recent years [CZL21]. MSA, being a software architecture pattern, defines the rules, patterns and constraints respectively on how to build the software architecture. Even though MSA have been researched upon in the past, we realised that this domain has not been addressed to its full potential. Specifically, the understanding around the design and the concrete implementation of this architectural pattern is not clear and comprehensive enough. Additionally, we realised that there exist complex operations and hidden costs that need to be handled by architects and developers when designing, implementing and testing microservices applications. For these purposes, there are plenty of design guidelines, including best practices, design patterns, and design principles, in industry and academia to construct MSA. However, information around such design guidelines is mostly unstructured, non-standardised and scattered. Consequently, it is unclear how to adopt and exercise the guidelines. It is also ambiguous how the design guidelines concretely affect the quality of a microservices application as it is difficult to determine which specific aspect of quality is influenced by the application of a particular design guideline under consideration.

Lack of understanding and documentation on how to design and develop MSA can lead to negative consequences such as increased development costs, decreased productivity of the developers, and challenges in product management. We observed that several design guidelines seem to exclude each other (e.g. *Orchestration* vs *Choreography*), however a clear overview of this is missing. As such, we aimed to perform a study in order to discover existing design guidelines in the context of MSA.

Building MSA without adhering to certain design guidelines significantly increases the risk of the deterioration of system quality. Industry-experts and academic researchers dived into determining and describing the quality of software systems with various levels of granularity, and have proposed different Quality Models for the same. The ISO/IEC 25010 standard constitutes such a model. However, with this established QMs, there is a perceived lack of depth and specificity with respect to describing the quality of microservices applications and architectures, as also detailed in Chapter 4.4.2. For example, we studied the ISO/IEC 25010 QM and noticed that it does not cater to the *scalability* aspect of a software product. Meanwhile, we know from literature and practice that increased *scalability* is one of the essential benefits and propositions of the MSA pattern [BD19], [JC19], [KM19]. Hence, we were motivated to propose an extension of the ISO/IEC 25010 standard by a finer, more granular Quality Model. The proposed model describes additional sub-characteristics being relevant for MSA, and aligns them with the given classification of the quality characteristics of the ISO/IEC 25010 standard.

2.2. Research Questions

Keeping the motivation and vision in mind, we proposed the following *Research Questions (RQs)* that we aimed to answer with this thesis:

- **RQ1:** Which design guidelines exist for constructing meaningful Microservices Architectures?

The first research question delivers a structured catalogue of design guidelines, including best practices, design patterns, and design principles, that are or should be adopted when dealing with MSA. We addressed this question by conducting a *Systematic Literature Review (SLR)* in order to establish a well-rounded literary foundation.

- **RQ2:** Which quality characteristics adequately describe the quality of Microservices Architectures and how can they be meaningfully represented in the form of a Quality Model?

The second research question proposes a Quality Model which specifies a variety of quality characteristics that collectively aim to sufficiently describe the quality of microservices applications and architectures.

- **RQ3:** Which quality characteristics are affected by the existing design guidelines of Microservices Architectures?

Lastly, the third research question presents a mapping to identify the quality characteristics relevant to MSA which are related to and affected by each of the existing design guidelines around MSA.

2.3. Scope

For carrying out comprehensive research and in order to study the architectural information which is important for analysing the quality of microservices systems, we only considered Microservices Architectures for this study, thus the scope of this thesis is limited to such architectures.

3. Related Work

This chapter reveals insights into the current state of research and the existing knowledge related to the subject of interest. It presents the views of different authors regarding the research domain in existing literature.

3.1. Literature Research

Prior work reports the state-of-the-art approaches that are centered around investigating MSA. Researchers have tried to capture the state of the latest research on MSA pertaining to various use cases. In [JC19], the authors presented a detailed review of the scientific approaches that are targeted towards MSA, and proposed a multi-tiered taxonomy to classify the research work around MSA. The authors discussed various distributed computing paradigms that are utilising microservices, including the novel challenges in this domain. [DML17] applied a systematic mapping study to identify, categorise, and validate the latest findings on architecting microservices considering the publication trends, research focus, and the potential for adoption in industry. The authors proposed a categorisation scheme for classifying 71 selected research publications. [BNK20] explained a variety of opportunities and problems associated with adopting and implementing MSA. The research method of this work harnessed 19 interviews with software architects harbouring experiences in a range of areas including corporate systems, middleware, SOA and microservices.

As part of our research, we found that published secondary studies on collections of best practices, design patterns, and design principles around MSA are limited. [HW+18], [Bog+21], [BD19], [CZL21], [Lai+21], [MA18], [Bil+22], [Val+20], [WKR21], [VF21], [Abd+21] present secondary studies in the form of Systematic Literature Reviews or Systematic Mapping Studies which employed a mixture of mostly academic literature and sometimes grey literature sources for collecting and validating relevant information with respect to the research question under consideration. Additionally, studies in [Che+15], [HBK20], [MA18], [Oli+20], [Sou+17], [Val+20], [WKR21], [VF21] loosely grouped the MSA design guidelines they reported based on some qualitative measure. [WKR21] gathered and classified the best practices, related challenges, and current solutions used by industry practitioners who design and develop microservices applications for commercial purposes. The research methods of this work include 21 interviews with microservices practitioners and an online survey with 37 respondents. [MA18] investigated the architectural patterns that are used in 30 open source microservices systems via detailed code and design reviews. [Oli+20] performed a study to identify design patterns that impact architectural design decisions related to the service size, database sharing, and the level of coupling of services. Using this information, the authors proposed a method for architectural trade-off analysis. Lastly, [Abd+21] conducted a SLR which reported 41 service identification approaches. The authors proposed a taxonomy of such approaches and developed a multi-level classification scheme around them. The authors also evaluated the scope of the SLR with industry-based practitioners.

Several publications, including [HW+18], [Bog+21], [BD19], [PJZ18], [Val+20], mentioned or discussed the quality characteristics relevant to MSA and hinted towards the quality assurance of microservices systems. [HW+18] provided a general overview of design quality measurement in SOA and identified the most common quality attributes in service-oriented design via a SLR. [Bog+21] investigated how industry-based practitioners observed evolvability assurance for microservices, including the tools, metrics and patterns they apply, and the challenges they fear for the evolvability of their microservices systems. The research method of this work included semi-structured interviews with 17 microservices experts. Additionally, the authors performed a *Grey Literature Review (GLR)* to analyse 295 online sources. [Val+20] worked on a multivocal literature review for MSA design patterns where the authors categorised the patterns based on the benefits they offer. In this work, the authors made use of the ISO/IEC 25010 QM to evaluate the product quality of applications employing MSA. They also researched on finding the quality attributes related to the discovered design patterns. Lastly, [BD19] conducted a SLR of architectural approaches for the implementation of Continuous Delivery (CD) and *DevOps*. The authors discovered 17 architectural characteristics that are useful when adopting CD and DevOps. In addition, they identified 10 probable hindrances in embracing CD and DevOps for current software systems.

These works indicate that research on MSA, involving design guidelines and quality assurance, is an ongoing effort. The work of this thesis differentiates from current research in the sense that it consolidates and presents the existing design guidelines around MSA in the form of a catalogue. In addition, our work uniquely contributes to existing research by proposing a well-grounded classification scheme for the collected design guidelines. We also adequately address quality, including quality characteristics, in microservices applications and architectures by formulating a Quality Models which is a novel contribution in itself. Finally, we cater to the research gap around determining the relation of the collected design guidelines and the quality characteristics in existing research.

4. Research Approach

This chapter presents the overall research approach that was adopted for the purpose of this thesis. It also provides details of the individual research methods that were followed for carrying out the thesis. We explain our research approach in a sequential manner where we demonstrate how we worked in order to get to the results of the thesis.

4.1. Overall Approach

The state chart in Figure 4.1 presents the overall research process of the thesis, including the individual steps which we undertook to execute our research. Each of the steps are in correlation with the discussed research questions and serve as central methods around which this thesis was framed.

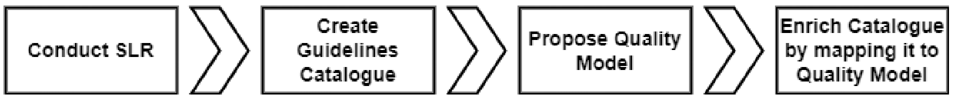
Conduct SLR
Create Guidelines Catalogue
Propose Quality Model
Enrich Catalogue by mapping it to Quality Model

Figure 4.1.: State Chart of the Overall Research Approach

As the first step, we perform a Systematic Literature Review with the primary purpose of researching design guidelines associated with constructing meaningful MSA. This method serves as the main research methodology adopted for the thesis. We provide and discuss all the SLR related aspects such as the strategy used to search for the information, the inclusion, exclusion and selection criteria meant to choose the relevant information, and the procedures used to ensure the application of the mentioned criteria. Details regarding this step and its respective actions and outcomes have been covered meticulously in Section 4.2.

Next, we analyse the literature sources identified in the SLR in order to develop and structure a catalogue of design guidelines used to build MSA. We provide and discuss the methods used to extract and classify the relevant information. Details regarding this step and its respective actions and outcomes have been discussed diligently in Section 4.3.

Having obtained sufficient information regarding research around MSA, we formulate and propose a Quality Model tailored to microservices applications and architectures. For this purpose, we identify quality characteristics which collectively describe the quality of microservices applications and architectures meaningfully. We provide and discuss all relevant aspects such as the reference Quality Model used in this research, the sources of the quality characteristics and the strategies used to extract and describe them, the techniques used to build the Quality Model, and the evaluation method applied to validate the QM. Details regarding this step and its respective actions and outcomes have been covered thoroughly in Section 4.4.

With the knowledge of the design guidelines and the quality characteristics relevant to MSA, we close our work by performing a mapping to identify which quality characteristics of microservices systems are related to and affected by each of the existing design guidelines. We provide and discuss the mapping related aspects such as the mapping strategies used. Details regarding this step and its respective actions and outcomes have been covered fully in Section 4.5.

4.2. Performing a Systematic Literature Review

4.2.1. SLR Method

In order to utilise a traceable method, we adopted the sophisticated SLR method proposed by [Kee+07]. A Systematic Literature Review is a method to identify, evaluate, and interpret all researches relevant to a specific research question or topic area [Kee+07]. This method is widely used for academic research in the software engineering domain. This method was adopted to unify the procedure of performing literature research and to deliver valuable results. [Kee+07] proposed a set of guidelines which helped in conducting the SLR. One of the aspects to evaluate theoretical work based on existing literature is to provide a clear and reusable process, hence we were motivated to follow this particular method. Additionally, this method focuses on a specific overview of publications on a given topic and allows for an in-depth analysis of the gathered material. It also fit with our need to solve a particular issue and search for relevant and precise information. When performing the SLR, the first step is to identify the need for a review. According to [Kee+07], the need for a fully systematic literature review arises from the requirement of researchers to summarise all existing information around a specific phenomenon in a thorough, exhaustive and unbiased manner. This might be done in order to draw more general conclusions about that phenomenon than is possible from individual studies, or to undertake as a prelude to future research activities.

In our work, the research phenomenon was centered around the discussed research questions for which a thorough and exhaustive SLR was not an essential requirement, given the scope. Hence, we adjusted the original SLR method from [Kee+07] based on our requirements. The adjustment included omitting the *Study Quality Assessment* and *Dissemination Strategy* steps, as explained later in Section 4.2.9. We frame the SLR as the main research method for building upon the research initiatives of this thesis.

We outlined the steps of the SLR process in the state chart shown in Figure 4.2. In the SLR, we conceptualised the search criteria, applied it on selected databases, and subsequently executed consecutive screening phases on the literature sources in order to obtain the final SLR corpus.

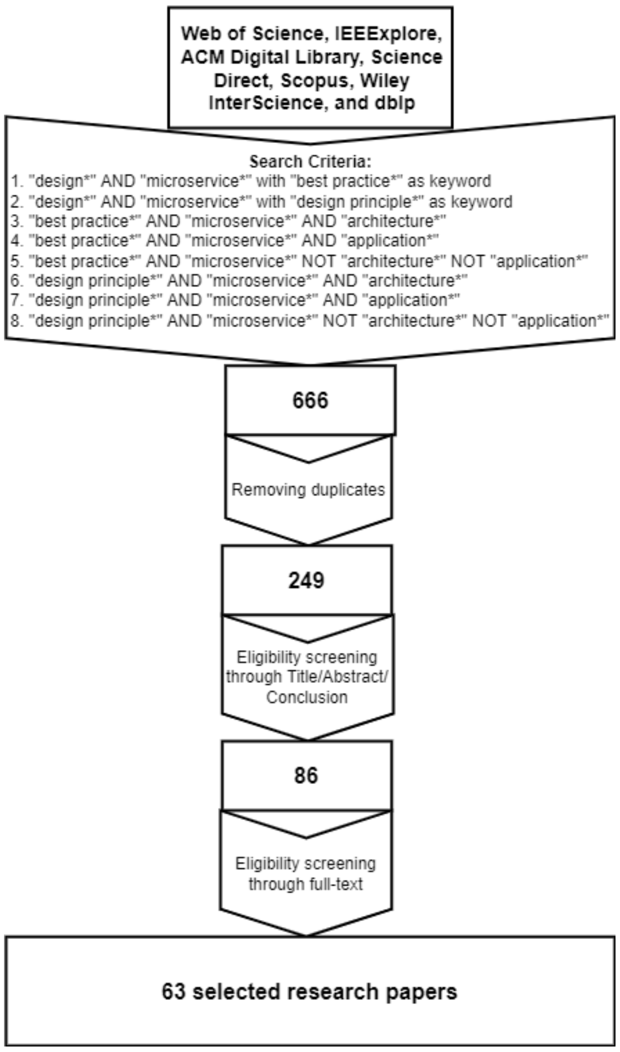
Web of Science, IEEExplore, ACM Digital Library, Science Direct, Scopus, Wiley InterScience, and dblp
Search Criteria:
1. "design*" AND "microservice*" with "best practice*" as keyword
2. "design*" AND "microservice*" with "design principle*" as keyword
3. "best practice*" AND "microservice*" AND "architecture*"
4. "best practice*" AND "microservice*" AND "application*"
5. "best practice*" AND "microservice*" NOT "architecture*" NOT "application*"
6. "design principle*" AND "microservice*" AND "architecture*"
7. "design principle*" AND "microservice*" AND "application*"
8. "design principle*" AND "microservice*" NOT "architecture*" NOT "application*"
666
Removing duplicates
249
Eligibility screening through Title/Abstract/ Conclusion
86
Eligibility screening through full-text
63 selected research papers

Figure 4.2.: State Chart of the SLR Process

4.2.2. Inclusion Criteria

We were interested in researching the design guidelines around MSA, so we decided to define search criteria that would be as specific as possible. Our focus was targeted towards the best practices and the design principles of building MSA. We wanted to search for them in academic literature, hence the main parts of the search criteria included the terms *best practice* and *design principle*. Given the context in which we wanted to find these terms, we decided to search for them in relation to *microservice architecture* and *microservice application*. We considered all possible combinations of these search terms in order to completely define the search criteria as described in Table 4.1 (where the wild card operator '*' denotes searching for the lexically related terms).

Inclusion Criteria (I#)	**Definition**
I1	Publications containing the terms ”design*” and ”microservice*”, with ”best practice*” as keyword filter
I2	Publications containing the terms ”design*” and ”microservice*”, with ”design principle*” as keyword filter
I3	Publications containing the terms ”best practice*” and ”microservice*” and ”architecture*”
I4	Publications containing the terms ”best practice*” and ”microservice*” and ”application*”
I5	Publications containing the terms containing ”best practice*” and ”microservice*” but omitting ”architecture*” and ”application*”
I6	Publications containing the terms ”design principle*” and ”microservice*” and ”architecture*”
I7	Publications containing the terms ”design principle*” and ”microservice*” and ”application*”
I8	Publications containing the terms ”design principle*” and ”microservice*” but omitting ”architecture*” and ”application*”

Table 4.1.: Search Criteria for the SLR

Each of the described search criteria helped us in forming its respective search string. We coupled the search terms via `AND`, `OR` and `NOT` logical operators in order to compose complete and valid search strings. From the search strings, we worked on coding the search queries tailored to the querying rules of each of the scientific databases that we considered in the SLR. We searched through the full-text and the meta-information fields of the publications, unless otherwise explicitly stated in the search query. This guarantees consistency and reproducibility of results. In the case of two publications having the same title but different abstracts and full-texts, we included both such studies in the SLR.

Although *Microservice Design Patterns* is a separate and extensive research area altogether, we included publications which discussed architectural design patterns in our work. We also included papers which discussed Microservice *API* Patterns. Additionally, we included papers which elaborated strategies and practices for the migration of software systems from monolithic architectures to MSA. Furthermore, we included papers which discussed architectural refactoring practices which, in essence, are inspired from the design guidelines of MSA. We understand that MSA is a subset of the SOA paradigm which could be considered as a more general implementation of a distributed service architecture [Neu15]. Hence, we included papers which discussed the design guidelines of SOA that were in common with MSA.

We encountered the situation of whether to include secondary studies which discussed design guidelines around MSA as part of the SLR or not. In order to gain clarity on this issue, we performed a pros-cons analysis keeping our research objectives in mind. We discovered that, in most cases, including secondary studies in systematic reviews reduces the likelihood of bias and serves as a more efficient pathway for identifying relevant literature, apart from systematic searching [Kee+07]. Later when conducting the trial searches (discussed in Section 4.2.5), we discovered that the identified secondary studies were in direct relation with our research question so including them actually pushed us in the right direction of what we were seeking to answer. Additionally, secondary studies are known to provide a basis for background reviews in prior research [HW+18], [BD19], [WKR21]. Since one of the objectives of the SLR was to identify existing design guidelines of MSA, and this SLR would serve as a basis and background study for all the research questions of the thesis, we considered the inclusion of secondary studies in the SLR.

4.2.3. Selection Criteria

For the scientific databases which offered the possibility to restrict search results based on the subject area, publication type, publication year, publication stage, and language, we defined the values for these filters according to our requirements for the SLR. We considered the default values for all other criteria that have not been explicitly mentioned as they appear in the scientific databases themselves. We selected English-language based publications completely published between January 2014 and April 2022. Table 4.2 enlists the restrictions applied when gathering the literature. The values are listed in no particular order. We included sub-types of the selected article types, if any. The starting year of the publication year range was determined by the year of the first microservices study known in the literature.

Filter	**Value**
Research Areas (Web of Science)	Computer Science
Publication Topics (IEEExplore)	Software Engineering
Subject Areas (Science Direct, Scopus, Wiley Inter-Science)	
Article Type	Conference Paper
Document Type	Conference Proceedings
Content Type (ACM Digital Library)	Research Article
	Journal Paper
Publication Year	Between 2014 and 2022
Publication Stage	Final
Language	English

Table 4.2.: Selection Criteria for the SLR

4.2.4. Exclusion Criteria

We excluded research papers possessing the criteria described in Table 4.3 from the SLR as they provided widely varying results and transcended the intended scope of this SLR.

Exclusion Criteria (E#)	Definition
E1	Is not relevant to microservices architectures, applications or designs in the computer science or software engineering fields
E2	Does not discuss best practices, design principles, design patterns or quality characteristics of MSA
E3	Only discusses general characteristics and features of MSA
E4	Identifies design principles of DevOps-driven architectures
E5	Researches on anti-patterns and architectural design issues or smells in microservices systems
E6	Provides modelling approaches for SOA, web services or mobile services

Table 4.3.: Exclusion Criteria for the SLR

We excluded publications which deviated from the point of view of design guidelines around MSA. We discarded articles which only discussed the implementation cycle of a microservices system solving a particular problem. For cases where different versions of the same publication appeared across one or more databases, we included the most recent version of the paper and discarded the prior ones. For cases where multiple instances of the same publication appeared across different databases, we ignored the duplicate ones so that only a single instance of the paper was included.

We realised that, in computer science research, the terms *cloud native architectures* and *Microservices Architectures* are used interchangeably. In reality, MSA have been around longer than cloud native computing. Moreover, in the realm of cloud native architectures, microservices form one artifact among various others such as *Continuous Integration/Continuous Delivery (CI/CD)*, DevOps and containers. According to [Kee+07], the inclusion and exclusion criteria should be based on the research question at hand, which in our case is specifically centered around MSA. Thus, we excluded papers on cloud native architectures from the SLR.

During our research, we came across publications which discussed the design principles of *RESTful* services. The design of RESTful *Application Programming Interfaces (APIs)* is a domain in itself and is only weakly connected and indirectly related to the domain of MSA. Additionally, diving into this finer level of granularity is out of the scope of this SLR. Thus, we excluded papers discussing the design principles of RESTful services from the SLR.

4.2.5. Study Selection Process

We utilised an electronic search method, harnessing the search queries devised in Section 4.2.2, through which we searched for design guidelines in the context of microservices applications and architectures respectively within major online scientific databases. We decided to use the databases proposed by [Kee+07] with the addition of *Web of Science*, resulting in a total of 12 databases as presented in Table 4.4. Direct access to *Inspec* was not available, however, indirect subscription through vendor platforms such as *Web of Science* was possible.

Trial Searches

First, we performed multiple trial searches on the 12 databases listed in Table 4.4. The trial searches were aimed at sharpening the search queries and assessing the volume of potentially relevant studies. This increased the accuracy of the search results and allowed to identify existing systematic reviews. The trial searches led us to exclude some of the databases from the SLR. We rejected the databases that offered either limited filtering possibilities, huge volumes of papers pertaining to irrelevant topics, or a very small (negligible) amount of research papers. This screening activity removed *SpringerLink*, *CiteSeerX*, *IET Digital Library* and *Google Scholar* from our scope of databases. Lastly, we removed *Ei Compendex* as it was not accessible.

Database (DB#)	**Name**	**Included in SLR?**	**Date of Access**	**Papers Included in SLR**
DB1	**Web of Science**	Yes	01.06.2022	7
DB2	**IEEExplore**	Yes	01.06.2022	14
DB3	**ACM Digital Library**	Yes	01.06.2022	12
DB4	**ScienceDirect**	Yes	01.06.2022	1
DB5	SpringerLink	No	26.05.2022	-
DB6	**Scopus**	Yes	01.06.2022	24
DB7	CiteSeerx	No	26.05.2022	-
DB8	**Wiley Inter-Science**	Yes	01.06.2022	4
DB9	Ei Compendex	-	-	-
DB10	IET Digital Library	No	26.05.2022	-
DB11	**dblp**	Yes	01.06.2022	1
DB12	Google Scholar	No	26.05.2022	-

Table 4.4.: Database and Literature Selection for the SLR

Refined Searches

Following the trial searches, we engaged in refined searches across the 7 scientific databases that we eventually considered in the SLR (highlighted in **bold** in 4.4). Refined searching included application of the selection criteria and the exclusion criteria onto the search results in a systematic manner. This led to listing a total of 666 research papers right after searching. Subsequently, we executed the first literature screening phase where we discarded duplicate publications. This resulted in 249 papers after removing the duplicates.

Taking inspiration from [Kee+07] and in order to decide upon the inclusion of specific articles in our research based on our requirements and scope, we executed the second literature screening phase where we reviewed the titles, abstracts and conclusions of the 249 research articles for relevance. The relevance was centered around indications of best practices, design patterns, and design principles of constructing MSA in the papers. In this phase, we also removed inaccessible, unavailable or malicious literature sources. This activity yielded a total of 86 papers while discarding 163 papers. Finally, we carried out the third literature screening phase by cautiously reading through the full-text of the 86 publications and identifying concrete occurrences of best practices, design patterns, and design principles of building MSA. We also verified whether the text potentially answers the research question and whether it demonstrates clarity in the purpose of our investigation. This screening led to a total of 63 papers while discarding 23 papers. The evolution of the SLR through the described screening phases is visually represented in Figure 4.3.

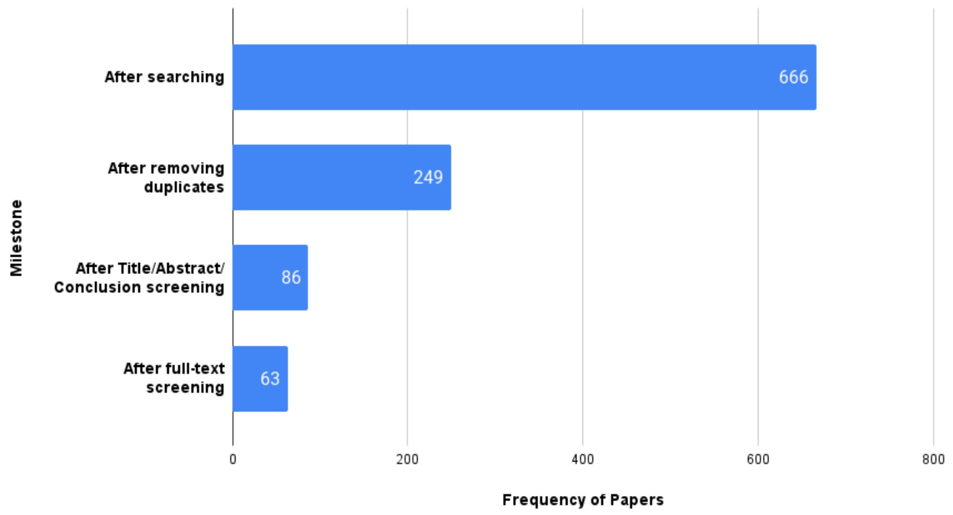

Figure 4.3.: Evolution of the SLR

4.2.6. SLR Corpus

The refined searches and the literature screenings yielded the SLR corpus containing a total of 63 academic publications that were included in this SLR. All of our research questions majorly benefited from the SLR corpus either directly or indirectly, thus it is an essential artifact of the thesis. The information offered by the papers of the SLR corpus was important in deriving the results of the thesis. Hence, we referred to it multiple times throughout the course of the thesis. The meta-information of the SLR corpus is available via GitLab[1]. The reference numbers, titles, and bibliographic references of the papers of the SLR corpus are available in Tables 4.5, 4.6, 4.7, 4.8, 4.9. The stated reference numbers were used to refer to the respective publications of the SLR corpus throughout this thesis.

[1]https://git.rwth-aachen.de/faizan.zafar/master-thesis/-/tree/main/SLR

Reference Number ([#])	**Title**	**Reference**
[1]	Performance Analysis of Microservice Design Patterns	[AP19]
[2]	Composition of heterogeneous web services: A systematic review	[HS19]
[3]	Design Quality Measurement for Service Oriented Software on Service Computing System: a Systematic Literature Review	[HW+18]
[4]	Architectural smells detected by tools: A catalogue proposal	[AFT19]
[5]	A Microservice Architecture for the Intranet of Things and Energy in Smart Buildings: Research Paper	[Bao+16]
[6]	Architecting Microservices: Practical Opportunities and Challenges	[BNK20]
[7]	Containerized Development and Microservices for Self-Driving Vehicles: Experiences & Best Practices	[BNB17]
[8]	Using microservices in educational applications of IT-company	[Ber+17]
[9]	Industry practices and challenges for the evolvability assurance of microservices: An interview study and systematic grey literature review	[Bog+21]
[10]	Designing software architecture to support continuous delivery and DevOps: A systematic literature review	[BD19]
[11]	Freshening the Air in Microservices: Resolving Architectural Smells via Refactoring	[BNS19]
[12]	Microservices approach for the internet of things	[BGT16]
[13]	Towards a Methodology for creating Internet of Things (IoT) Applications based on Microservices	[Cab+20]

Table 4.5.: SLR Corpus

Reference Number ([#])	Title	Reference
[14]	A systematic gray literature review: The technologies and concerns of microservice application programming interfaces	[CZL21]
[15]	Architectural Support for DevOps in a Neo-Metropolis BDaaS Platform	[Che+15]
[16]	Tapis API Development with Python: Best Practices in Scientific REST API Implementation: Experience implementing a distributed Stream API	[Cle+20]
[17]	Synergies of System-of-Systems and Microservices Architectures	[CNZ16]
[18]	Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption	[DML17]
[19]	Microservices: A performance tester's dream or nightmare?	[Eis+20]
[20]	Evaluation of API Request Bundling and its Impact on Performance of Microservice Architectures	[EZ21]
[21]	Following Domain Driven Design principles for Microservices decomposition: is it enough?	[Far+21]
[22]	Towards a Method for Monitoring the Coupling Evolution of Microservice-Based Architectures	[FF20]
[23]	An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems	[Gan+19]
[24]	"Functional-First" Recommendations for Beneficial Microservices Migration and Integration Lessons Learned from an Industrial Experience	[GT19]
[25]	Decision Guidance Models for Microservices: Service Discovery and Fault Tolerance	[HWB17]
[26]	Microservice transition and its granularity problem: A systematic mapping study	[HBK20]

Table 4.6.: SLR Corpus (continued)

Reference Number ([#])	Title	Reference
[27]	Microservice design for container based multi-cloud deployment	[Jam17]
[28]	Straddling the crevasse: A review of microservice software architecture foundations and recent advancements	[JC19]
[29]	ToLambda-Automatic Path to Serverless Architectures	[Kap19]
[30]	Migrating Legacy Software to Microservices Architecture	[KM19]
[31]	Data management in microservices: State of the practice, challenges, and research directions	[Lai+21]
[32]	Microservice Patterns for the Life Cycle of Industrial Edge Software	[Li+18]
[33]	Microservice Architectures for Advanced Driver Assistance Systems: A Case-Study	[Lot+19]
[34]	Interface Evolution Patterns: Balancing Compatibility and Extensibility across Service Life Cycles	[Lüb+19]
[35]	Actual Use of Architectural Patterns in Microservices-Based Open Source Projects	[MA18]
[36]	Semi-automatic Feedback for Improving Architecture Conformance to Microservice Patterns and Practices	[Nte+21]
[37]	Assessing Architecture Conformance to Coupling-Related Patterns and Practices in Microservices	[Nte+20]
[38]	Supporting Architectural Decision Making on Data Management in Microservice Architectures	[Nte+19]
[39]	A Method for Architectural Trade-off Analysis Based on Patterns: Evaluating Microservices Structural Attributes	[Oli+20]

Table 4.7.: SLR Corpus (continued)

Reference Number ([#])	Title	Reference
[40]	Architectural principles for cloud software	[PJZ18]
[41]	Security of Microservice Applications: A Practitioners' Perspective on Challenges and Best Practices	[Bil+22]
[42]	Microns: Commands for Building Bubble Microservices	[Ras18]
[43]	Content Management System Architecture	[Shi17]
[44]	Dimensions of Software Configuration: On the Configuration Context in Modern Software Development	[SRS20]
[45]	Patterns on Deriving APIs and Their Endpoints from Domain Models	[Sin+21]
[46]	The TOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures	[Sol+21]
[47]	Engineering Software for the Cloud: Messaging Systems and Logging	[Sou+17]
[48]	Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages	[Tai+17]
[49]	Patterns Related to Microservice Architecture: a Multivocal Literature Review	[Val+20]
[50]	MAMS: Multi-Agent MicroServices	[W C+19]
[51]	Promises and challenges of microservices: an exploratory study	[WKR21]
[52]	Developing, deploying, and operating twelve-factor applications with TOSCA	[Wur+17]

Table 4.8.: SLR Corpus (continued)

Reference Number ([#])	**Title**	**Reference**
[53]	Reflections on SOA and Microservices	[XWQ16]
[54]	Interface Responsibility Patterns: Processing Resources and Operation Responsibilities	[Zim+20]
[55]	Migrating Web Applications from Monolithic Structure to Microservices Architecture	[Ren+18]
[56]	Applying Microservice Refactoring to Object-2riented Legacy System	[ZZ21]
[57]	Best Practices and Strategy for the Migration of Service-Oriented Architecture-Based Applications to Microservices Architecture	[RS22]
[58]	State of the Practice in Service Identification for SOA Migration in Industry	[Abd+18]
[59]	Pattern-based Multi-Cloud Architecture Migration	[JPM17]
[60]	Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices	[Col+21]
[61]	Monolithic to Microservices Migration Strategy in Public Safety Secretariat of Mato Grosso	[Pre+21]
[62]	Monoliths to microservices-Migration Problems and Challenges: A SMS	[VF21]
[63]	A taxonomy of service identification approaches for legacy software systems modernization	[Abd+21]

Table 4.9.: SLR Corpus (continued)

4.2.7. Data Synthesis

We describe the data synthesis procedure in detail in Section 4.3, including how we extracted the data from the SLR corpus and synthesised the extracted data. During the extraction procedure, we identified the design guidelines, including best practices, design patterns, and design principles, for constructing MSA. During the synthesis procedure, we structured and categorised this information in the form of a catalogue.

4.2.8. Conflict of Interest

There were no conflicts of interest, nor secondary interests of the author of the thesis. All opinions presented were that of the author alone, and not of any institution to which they were affiliated.

4.2.9. Excluded Steps

As part of our SLR method, we omitted two steps from the sophisticated SLR method proposed by [Kee+07].

Study Quality Assessment

The purpose of study quality assessment was to take into account the systematic error of the papers, and their internal and external validity. Since the SLR method served as a basis for exploring the multiple research avenues of the thesis, it was difficult to assess the internal and external validity of the papers. Moreover, the research results of the papers seemed to have no bias as the authors did not focus on altering their research process between what they considered and what they presented.

Dissemination Strategy

As the SLR was a central method of the thesis, this work will be under the evaluation of other researchers and practitioners at some point. However, to date we have not assessed the SLR via a formal peer review due to time and resource constraints. The thesis itself was examined by one supervisor holding a Masters degree and two professors at RWTH Aachen University in order to grade the work of the thesis student who is the author of this thesis. All of the research papers cited and referenced in this SLR were peer reviewed as they were published in well-known scientific databases. The thesis document is made publicly available to members of microservices communities in industry and academia.

4.3. Developing a Catalogue of Design Guidelines

After obtaining the SLR corpus, we performed an in-depth analysis on all of its included papers. In this analysis, we focused on identifying and extracting the required data i.e. the design guidelines, including best practices, design patterns, and design principles, that are proposed and employed in industry and academia for constructing MSA. Subsequently, we structured this information as a collection of 239 design guidelines in total. Figure 4.4 presents a state chart which outlines the steps involved in this process.

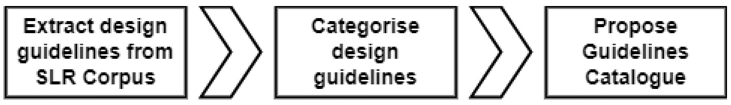
Extract design guidelines from SLR Corpus
Categorise design guidelines
Propose Guidelines Catalogue

Figure 4.4.: State Chart displaying the steps performed in developing a catalogue of design guidelines

Regarding the format of our collection, we were initially interested in utilising a pattern language which could serve as a basis for explaining our work. We were inclined towards employing well-known and established frameworks to organise the collected design guidelines, hence we referred to various notable sources for the same. However, we realised that our diverse collection encompassed a varying mix of patterns, practices and principles. Additionally, not all information that is required for common design pattern templates was available at our disposal. As a result, we decided upon a tabular format for our collection and referred to this artifact as a *Guidelines Catalogue* or *catalogue* throughout the thesis. We present the catalogue later in Chapter 6.2.

4.3.1. Categorisation of the Guidelines Catalogue

We developed the catalogue iteratively, improving it with each iteration. Due to the complex nature of the structure of the catalogue, we decided to classify the design guidelines in order to retrieve and refer the required information as efficiently as possible. We decided to utilise the core principles of the grounded theory methodology for deriving categories for the design guidelines reported in the catalogue. Qualitative data analysis in general, and the grounded theory methodology in particular, aims to add structure to unstructured data. Such structure is introduced in stages and forms the basis of any resulting frameworks, theories and explanations [SC98]. The grounded theory methodology helped us in identifying key themes, factors and concepts across the catalogue. Due to the qualitative nature of our research, the list of categories and dimensions that we report on is not intended to be complete. These constructs only emerged from performing the coding procedures of the grounded theory methodology on the data we collected, and represented homogeneous topics.

We followed three sequential stages: *open coding*, *axial coding*, and *selective coding*, as outlined in Figure 4.5.

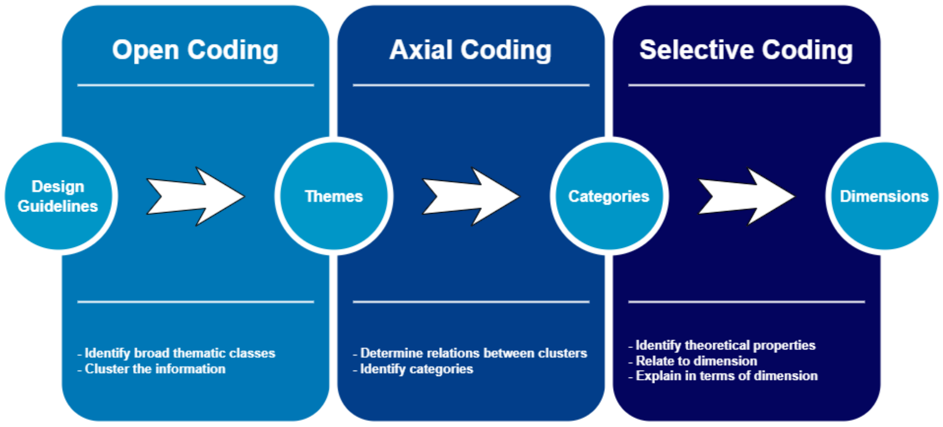
Open Coding
Axial Coding
Selective Coding
Design Guidelines
Themes
Categories
Dimensions
- Identify broad thematic classes
- Cluster the information
- Determine relations between clusters
- Identify categories
- Identify theoretical properties
- Relate to dimension
- Explain in terms of dimension

Figure 4.5.: The steps in the Grounded Theory Methodology for categorising the Guidelines Catalogue

First, we performed open coding which segments text into meaningful expressions and describes them as concisely as possible. Further, relevant annotations and concepts are then attached to these expressions [Mul14]. During open coding, we broke down the collected guidelines based on their semantic grouping, emerging concepts, functional domains, scope coverage, and known categories. Next, we performed axial coding which is used to group several open codes under a broader abstract concept [Mul14]. During axial coding, we explored and explained the relationships between the discovered clusters as distinct categories. Finally, we performed selective coding where we connect all of our discovered categories together around one or more *core* categories [GSS68]. During selective coding, we identified the core categories, related previously created codes and categories to the core, and explained them in terms of the core. Basically, we grouped the identified categories into *dimensions* which represent high-level ideas that we found in the collected data. We iteratively performed the coding procedures and reviewed them internally. Following each review, any coding-related disagreements were discussed and rectified.

In order to make the coding processes more structural, we utilised the *concept discovery* and *level of abstraction* concepts provided by [MT86]. Concept discovery refers to the process of converting raw data into abstract concepts that relate to the phenomenon that is being studied. The idea is to create abstract concepts or categories and group relevant sets of data under them [MT86]. The level of abstraction provides a structural approach towards creating these concepts. The main aim is to create a concept abstract enough so that each identified open code does not end up with its own unique concept. The other criterion is to make sure that the concept is explicitly related to the phenomenon under study [MT86]. For the purposes of our research, we did not fully adopt the grounded theory methodology as our main objective was to formulate categories and dimensions for the design guidelines, rather than establish a new theory. This led to the proposal of a categorisation scheme which we present later in Chapter 6.1.

4.4. Formulating a Quality Model For Microservices Architectures

After obtaining the Guidelines Catalogue, we analysed all of its design guidelines, along with examining the SLR corpus. In these analyses, we focused on identifying and extracting the required data i.e. the quality characteristics which pertain to microservices applications and architectures, and meaningfully express their quality. Considering these characteristics and the ISO/IEC 25010 standard as a basis, we proposed a Quality Model tailored to MSA. We evaluated this QM to produce a revised version as part of this thesis. Figure 4.6 presents a state chart which outlines the steps involved in this process.

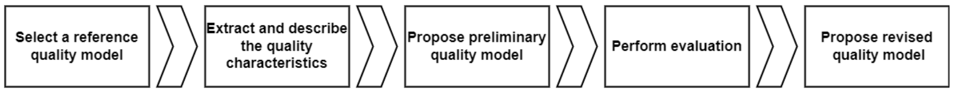
Select a reference quality model
Extract and describe the quality characteristics
Propose preliminary quality model
Perform evaluation
Propose revised quality model

Figure 4.6.: State Chart displaying the steps performed in developing a Quality Model for MSA

4.4.1. Important Terminologies

The overall quality of MSA is encompassed and explained by various quality characteristics. In our research, we split quality characteristics into *Quality Attributes (QAs)* and *Quality Factors (QFs)*:

Definition 3. *A* Quality Attribute *is a high-level quality characteristic that has been recognised and explored as part of a standard Quality Model.*

Definition 4. *A* Quality Factor *is a more specific and granular quality characteristic which pertains to quality in a certain context.*

During our research, we observed that each QA is influenced by a range of QFs which, in turn, informatively express the QA. The QFs which are related to a certain QA have a contributing affect on it. For example, one of the factors explaining the well-known *maintainability* attribute in MSA is the *modifiability* factor. The higher the degree of *modifiability*, the higher the *maintainability*.

4.4.2. Selection of a Reference Quality Model

Several QMs exist in academia and industry. The leading QM for systems and software is the ISO/IEC 25010 [ISO11]. It is part of the systems and software quality requirements and evaluation (SQuaRE) ISO/IEC 25000 series. This series of standards is a framework to specify and evaluate system and software quality. ISO/IEC 25010 evolved from ISO/IEC 9126 which is considered a standard decomposition of quality characteristics and suggests a small number of measures for measuring them [ISO01].

For assessing the different dimensions of quality traits and properties relevant to MSA in terms of an established quality framework, we were inclined to use a meaningful QM as a reference model. We decided that our reference QM should be able to determine which quality characteristics would be taken into account when evaluating certain properties of a software system. Hence, we employed the Software Product Quality Model defined in the ISO/IEC 25010 standard. The system and software product quality model relates to static properties of the software and dynamic properties of the computer system [ISO11]. It represents a normative perspective on a target entity and is usually considered from the standpoint of developers. The Quality Model decomposes into eight Quality Attributes: *functional suitability*, *performance efficiency*, *compatibility*, *usability*, *reliability*, *security*, *maintainability*, and *portability* with 23 corresponding Quality Factors, as shown in Figure 4.7.

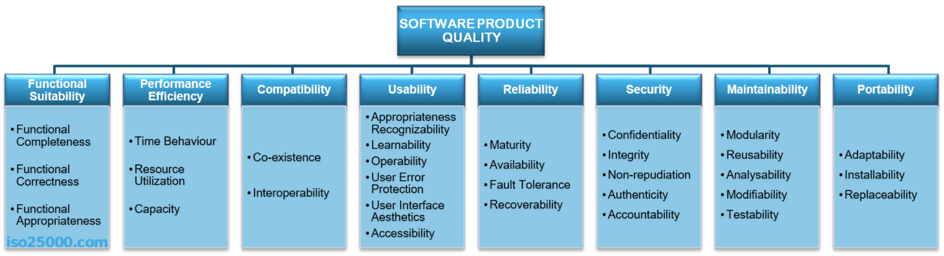

Figure 4.7.: The ISO/IEC 25010 Software Product Quality Model

The ISO/IEC 25010 Quality Model provided us with a solid starting point for investigating quality in MSA. We realised that this model, in its current form, is not sufficient for describing the quality of MSA very precisely, as highlighted in Chapter 2.1. Hence, we were inspired to pursue this research for deriving and proposing a particular QM for MSA.

4.4.3. Extraction and Description of Quality Characteristics

As part of our research, we analysed the Guidelines Catalogue and the SLR corpus with respect to which quality characteristics they were tackling. In these analyses, we noticed that these artefacts encompassed some quality characteristics (both attributes and factors) which have already been covered in the ISO/IEC 25010 Quality Model. We also observed that these artefacts encompassed several novel quality characteristics which are particularly relevant for MSA and have not been covered by the ISO/IEC 25010 QM. We considered these novel quality characteristics as Quality Factors on the same level as the ISO/IEC Quality Factors. All such factors collectively describe the quality of MSA to a great extent since they were derived from artefacts centered around MSA. For example, *evolvability* is a novel QF, among several others, which influences the *maintainability* attribute. Similarly, for services to be discoverable in a microservices application, developers needs to improve the *discoverability* factor, resulting in a more reliable architecture.

As a result of our analysis, we identified and extracted a total of 82 quality characteristics. As the next step, we sharply described each of the quality characteristics by proposing concrete and clear descriptions, so that our proposed Quality Model is as holistic as possible and considered as a complete taxonomy. We report the results of the extraction and description procedures later in Chapter 5.1.

4.4.4. Building the Model

Considering the complete ISO/IEC 25010 QM and the novel QFs as a basis, we vertically extended the ISO/IEC 25010 QM by describing the novel Quality Factors in terms of the Quality Attributes of the model. We adopted the *Concept Discovery* methodology from [Mul14] for extending the model as our objective was to realise the association of the novel QFs with the respect to the QAs of the model. Concept discovery is the process of grouping similar incidents from the raw data under abstract codes or categories. In our research, this translated to analysing the novel Quality Factors, and coding and categorising them under the broad Quality Attributes. Given the scope of our research, we adopted a one-to-one association scheme where each QF is considered to describe only one QA of the ISO/IEC 25010 QM. We depict the structure of the resulting model later in Chapter 5.1.

During our research, we discovered several novel quality characteristics that were not available in the ISO/IEC 25010 QM. This implied that the ISO/IEC 25010 QM is more of a universal and architecture-agnostic framework that does not differentiate in software architecture patterns, and is applicable across a variety of software architectures. Although we realised that the quality characteristics defined by our proposed QM are not specific to MSA, we ended up with the implication that the extracted quality characteristics are rather important for taking advantage of the MSA pattern than is the case for other architecture patterns. Moreover, they seem to describe the quality in MSA to a great degree. This proposition provides a strong basis for comparing MSA for which we are inclined to use an architecture-specific QM rather than a universal one.

4.4.5. Evaluation of the Quality Model

We understand that a Quality Model is a conceptualisation where a system's quality is abstracted in a model. We also comprehend the reliance on such a model and its significance in industry and academia. This led to the realisation that our proposed model needed to be validated through detailed feedback from relevant stakeholders. For this purpose, we were inclined towards adopting the evaluation approach. An evaluation describes a judgement based on a specific set of criteria. It is essentially an activity performed by the evaluator based on some predefined criteria. Hence, we labelled the proposed Quality Model as a preliminary one and worked towards producing a revised version as discussed later in Chapter 5.2.

4.5. Mapping Design Guidelines to Quality Characteristics

After obtaining the revised Quality Model, we analysed all of its elements, along with analysing the Guidelines Catalogue and the SLR corpus. In these analyses, we focused on recognising certain information i.e. the respective quality characteristic(s), including Quality Attributes and Quality Factors, which are related to the design guidelines around MSA. The relation comprises an influence or affect of a design guideline on its corresponding quality characteristic(s), irrespective of an improvement or deterioration. Figure 4.8 presents a state chart which outlines the steps involved in this process.

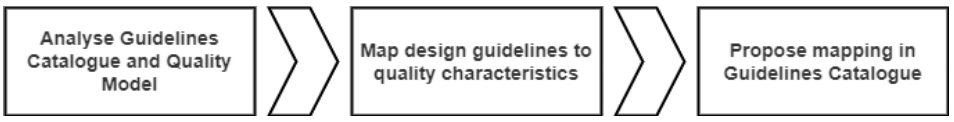
Analyse Guidelines Catalogue and Quality Model
Map design guidelines to quality characteristics
Propose mapping in Guidelines Catalogue

Figure 4.8.: State Chart displaying the steps performed in mapping design guidelines to quality characteristics

In the first step, we analysed the Guidelines Catalogue. We reviewed each design guideline and revisited the literature sources of the SLR corpus in which a particular guideline was referenced. From there, we identified the quality characteristic(s) that were in relation to the design guideline under scrutiny. Next, we immediately checked for the presence of our newly identified quality characteristic(s) in the revised QM, and finally included them in the catalogue against the guideline. We repeated the procedure for all the 239 guidelines that our Guidelines Catalogue encompassed.

For this mapping, we employed a one-to-many mapping scheme where each design guideline mapped onto one or more quality characteristics which it affects. For the few cases where a design guideline was not known to influence any quality characteristic, we inspected the description and properties of that guideline, cited it against multiple peer-reviewed publications outside of the SLR corpus, and discussed it internally. This provided us with the knowledge and expertise to map such guidelines to the corresponding quality characteristics which they influence.

In the following chapters, we present the results that were obtained from executing the research approach of this thesis. Due to clarity purposes, we decided to present the results in a different order as compared to how they are outlined in the current chapter. In the very next chapter, we present the Quality Model, including the preliminary results, the evaluation method, and the post-evaluation results. In the chapter following, we present the structured catalogue of design guidelines, including the proposed categorisation scheme and the mapping of the design guidelines to the related quality characteristics.

5. A Quality Model for Microservices Architectures

This chapter presents our contribution in the form of a Quality Model for MSA. We formally introduce our preliminary results and discuss in detail how we evaluate them as part of this thesis. We also elaborate the final post-evaluation results in this chapter.

5.1. Preliminary Quality Model

The resulting preliminary QM followed the same hierarchical structure as that of the ISO/IEC 25010 QM shown in Chapter 4.4.2. Moreover, since we extended the model and did not add or remove any of its attributes, the 8 Quality Attributes of the ISO/IEC 25010 QM remained unchanged in the proposed model and throughout our research. The preliminary QM encompassed 74 Quality Factors, including (51/74) novel QFs and (23/74) QFs which were the same as the existing QFs of the ISO/IEC 25010 QM. All quality characteristics (attributes and factors) of the model were derived and explained from the extraction and description procedures described in Chapter 4.4.3. The method described in Chapter 4.4.4 yielded the structure of the Quality Model, as visualised in Figure 5.1, with the novel Quality Factors marked in **bold**.

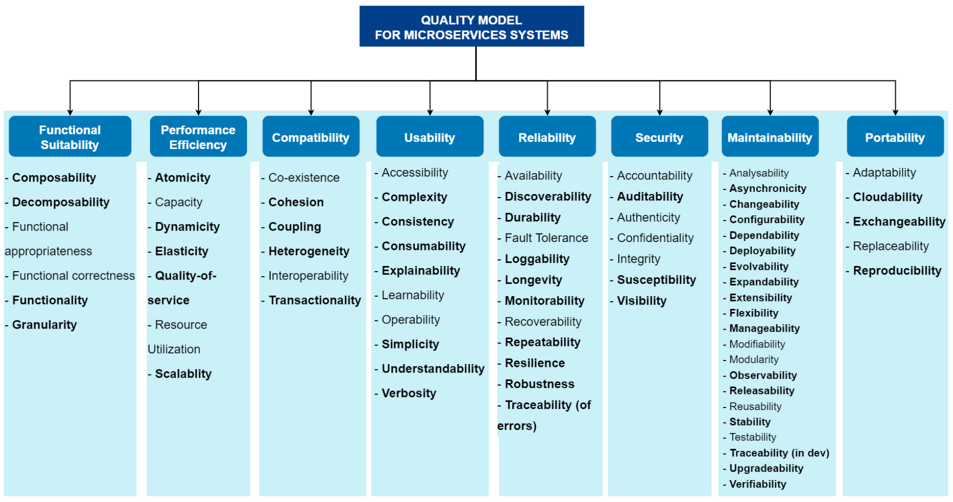

Figure 5.1.: Structure of the Preliminary Quality Model

Figure 5.2 visualises the ratio of the Quality Factors by their novelty type.

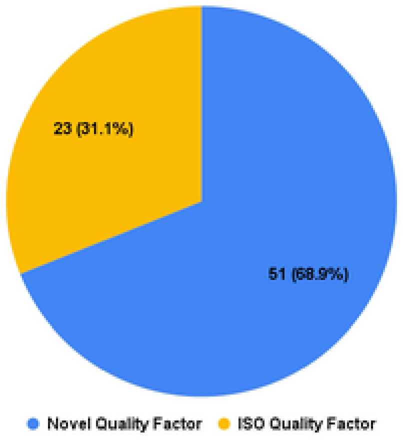

Figure 5.2.: Ratio of Quality Factors by Type

5.2. Evaluation of Preliminary Quality Model

As motivated in Chapter 4.4.5 and in order to appropriately determine the applicability and validity of our proposed Quality Model in industry and academia, we decided to evaluate our work as holistically as possible. We conducted semi-structured interviews with 13 microservices experts from industry and academia. We presented the participants our findings and particularly inquired about their feedback on our results with the help of an interview protocol. We collected and structured this feedback for further use in the thesis. Figure 5.3 outlines the steps involved in the evaluation process.

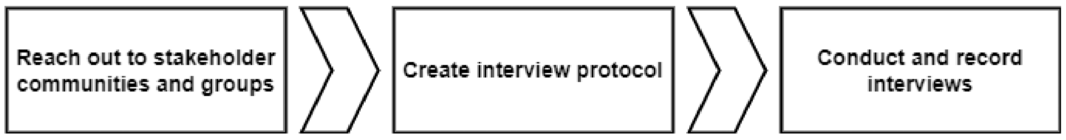
Reach out to stakeholder communities and groups
Create interview protocol
Conduct and record interviews

Figure 5.3.: State Chart of the Evaluation Approach

5.2.1. Evaluation Objective

The evaluation was solely aimed at the complete preliminary Quality Model and we intended to answer the following question with this evaluation:

- *To what extent does the proposed Quality Model, including all its elements, adequately describe the quality of Microservices Architectures?*

We were mainly interested in assessing the validity of the quality characteristics of the model in relation to MSA. We also intended to examine the rationality of the associations between the Quality Factors and the Quality Attributes in the model. Additionally, we aimed to evaluate our demonstrated understanding of the meaning of the quality characteristics based on the experience of microservices experts. We also aspired to uncover missing information in the proposed descriptions of the quality characteristics. Another goal of the evaluation was to assess the overall appropriateness of the proposed Quality Model for industrial and academic settings.

5.2.2. Evaluation Method

We encountered the situation of selecting an appropriate method for the evaluation of our work. We had to ensure that the selected method would do justice to evaluating the QM, while adhering to the scope and time constraints of the thesis. We were mostly divided between two methods. The first method was a survey which could target a wider audience and gather a variety of information quickly, however, it would require expertise and resources on survey preparation and could limit the scope of the responses. The second method was an interview which could obtain detailed answers from a limited number of interview participants and provide a basis for in-depth assessment, however, it would consume substantial time and management resources.

Since our research was exploratory in nature and we aimed to collect as much open-ended information as possible, we adopted an interview-based evaluation method which harnessed the strengths and opportunities of *individual semi-structured interviews*. Such interviews allow the researcher to understand the interviewees' perspective in a deeper manner and answer a wider range of questions [BC06], which further made it a preferable evaluation approach for this thesis. The semi-structured approach offers the flexibility to refocus questions, or prompt for more information, when an interesting or novel topic emerged [BNK20]. This method included preparing a set of open-ended questions for individual interviews to be conducted with stakeholders working with MSA such as industry practitioners and academic researchers. The interview itself was meant to be a guided discussion between the thesis student (the interviewer) and the respective stakeholder (the participant), along with some integrated conversational aspects.

5.2.3. Interview Preparation

First, we identified the different stakeholder communities that actively contribute to the microservices domain. We found the *Microservices Community (MC)* which is a European-based international community interested in the microservices paradigm, the *Gesellschaft für Informatik e. V. (GI)* which is a large Germany-based specialist society for computer science, and public microservices forums on *LinkedIn*. Next, we determined the different groups of stakeholders who would be interested in taking up an interview in this space. We decided that industry professionals and academic researchers who were part of the selected communities and possessed suitable and sufficient experiences around microservices would be ideal candidates for such an interview. We did not set any additional restrictions on the eligibility of the potential interview participants. We reached out to members of the selected communities by broadcasting a formal invitation community-wide. Alongside, we directly contacted microservices experts from the listed communities as well as from our social circles via formal personalised invites.

These invitations served as the interview preamble that explained the interview process, highlighted the relevant topics, and introduced the ethical considerations.

We decided to conduct one-on-one interview sessions with each of the participants involved. In order to eliminate the constraints of physical presence and owing to Covid-19 restrictions, we decided to conduct the interviews online via Zoom. Additionally, we decided to record the interview sessions with the participants' consent, so that the recordings were preserved and could be viewed later for extracting information. Given a limited time period, which was established to be 40 minutes, we decided to keep the interview on a more general level, rather than a very granular one which was not feasible in the limited duration. The preparatory information on the different aspects of the evaluation interview is summarised in Table 5.1.

Aspect	**Information**
Stakeholder Communities	Microservices Community
	Gesellschaft für Informatik e.V.
	Forums on LinkedIn
Stakeholder Groups	Industry Professionals
	Academic Researchers
Format	One-on-one
Time Limit	40 minutes per participant
Medium	Online via Zoom

Table 5.1.: Preparatory Information for the Evaluation Interview

Given the breadth and depth of the proposed QM and our limited time constraint, it was not possible for each participant to evaluate the complete QM in a single sitting. Hence, we decided to segment the model and discuss **one** Quality Attribute (e.g. *portability*) and a few of its selected Quality Factors (e.g. *adaptability*, *cloudability* etc.) with each participant. In this way, all segments of the model were evaluated by different participants and the model was completely covered as part of the evaluation.

In order to support the participants in evaluating the Quality Model, we prepared a slide deck. This slide deck outlined our research objectives, the premise of our research, and the research outcomes i.e. the structure of the preliminary Quality Model and its quality characteristics including their proposed descriptions. Since, we decided to split the model and present only one QA and a few of its selected QFs to each participant, we ended up with one slide deck per participant with the research objectives and the premise being common in all, but the presented quality characteristics being different.

Additionally, we authored an interview protocol that helped us to scope and organise the semi-structured interviews. We carefully designed the interview questions based on the evaluation objectives defined earlier. The protocol also served as a template for drafting the individual questions around our research outcomes. Since each participant was expected to evaluate a subset of the complete QM, we framed the interview questions according to the quality characteristics which were presented to the participant, and modified the questions accordingly. We structured the protocol in Table 5.2 taking the attribute *portability* and its factor *adaptability* as an example. We grouped the protocol in thematic sections, namely *Section A* which inquired on the participants' demographic information, and *Section B* which presented questions on the preliminary Quality Model itself. For the protocol, we defined the Likert scale as 1 corresponding to *Very Poor*, 2 corresponding to *Poor*, 3 corresponding to *Neutral*, 4 corresponding to *Acceptable*, and 5 corresponding to *Perfectly Acceptable.*

Section A: Questions on Demographic Information
Full Name
Qualification
Current Role and Affiliation
Years of Experience
Question 1: How long have you been working with microservices applications or architectures? What benefits or advantages have you experienced with such technologies?
Section B: Questions on Preliminary Quality Model
Premise 1: Ask Questions 2-5 for **each** of the presented Quality Factors.
Question 2: On a scale from 1 to 5, how would you assess *adaptability* as a Quality Factor for MSA? Why?
Question 3: On a scale from 1 to 5, how would you assess *adaptability* as a Quality Factor of the Quality Attribute *portability*? Why?
Question 4: Would you associate *adaptability* with any other Quality Attribute (from the proposed Quality Model or otherwise) for MSA? If so, which one?
Question 5: On a scale from 1 to 5, how would you assess the proposed description of *adaptability*? If poor, why?
Premise 2: Ask Question 6 for **only** the presented Quality Attribute.
Question 6: Do you feel that we have missed any Quality Factors for the quality attribute *portability*?
Question 7: On a scale from 1 to 5, how would you assess the overall appropriateness of the proposed Quality Model for MSA?

Table 5.2.: Interview Protocol for the Evaluation Interview

5.2.4. Interview Procedure

We started the interview with Section A of the protocol. We requested from the participant their demographic information and also asked about their background and experience with microservices applications and architectures. Next, we presented them their individual slide deck so that they get familiarised with our work. From there, we moved on to Section B of the protocol. We asked Questions 2 to 5 for **each** of the Quality Factors that were presented earlier to the participant. Next, we asked Question 6 which concerned the completeness of the Quality Attribute under discussion. Finally, our last question focused on the appropriateness of the Quality Models in a general sense.

During the interview, we engaged in *Note taking* which refers to the activity of making concise and relatable notes while the interview is being conducted [Mul14]. These quick notes supported us in asking for more detailed insights with regards to the concurrent response and feedback presented by the participant.

5.2.5. Participants' Demographic Information

Our 13 interview participants comprised knowledgeable professionals in the microservices domain, either scientifically or through experience gathered in the industry. This ensured high quality of the feedback obtained when evaluating the Quality Model. We collected the participants' demographic information in Tables 5.3, 5.4, particularly the aspects which we considered helpful in determining their credibility for assessing the QM.

Participant (P#)	Qualification	Current Role	Organization Size	Years of MSA Experience	Field	Expertise
P1	PhD Computer Science	Research Associate	Large	6	Academia	Research
P2	PhD Informatics Engineering	Professor	Large	6	Academia	Research
P3	M.Sc. Computer Science	Software Engineer	Large	5	Industry	Software Architecture
P4	BS Information Technology	Senior Data Engineer	Large	4	Industry	Data Architecture
P5	B.Sc. Computer Science	Masters Student	Large	1	Academia	Research
P6	PhD Computer Science	Researcher / Postdoc	Large	6	Academia	Research
P7	M.Sc. Services Computing	Research Associate / Lecturer	Large	4	Academia	Research
P8	M.Sc. Business Informatics	Research Associate	Large	4	Academia	Research
P9	B.Sc. Computer Science	Senior Engineering Manager	Large	6	Industry	Software Architecture

Table 5.3.: Participants for the Evaluation Interview

Participant (P#)	**Qualification**	**Current Role**	**Organization Size**	**Years of MSA Experience**	**Field**	**Expertise**
P10	B.Sc. Computer Science	Software Engineer	Medium	1	Industry	DevOps
P11	B.Sc. Computer Science	Software Engineer	Large	5	Industry	Software Architecture
P12	B.Sc. Electronics and Communications Engineering	Advanced CI/CD DevOps Engineer	Medium	3	Industry	DevOps
P13	M.Sc. Computer Science	Principal Architect	Small	5	Industry	Software Architecture

Table 5.4.: Participants for the Evaluation Interview (continued)

The majority i.e. (10/13) participants were affiliated with large scale organisations harbouring more than 500 employees. (2/13) participants were affiliated with medium scale organisations harbouring 300 to 500 employees. Lastly, (1/13) participant was affiliated with a small scale organisation harbouring less than 100 employees. Figure 5.4 visualises the number of participants by organisation size.

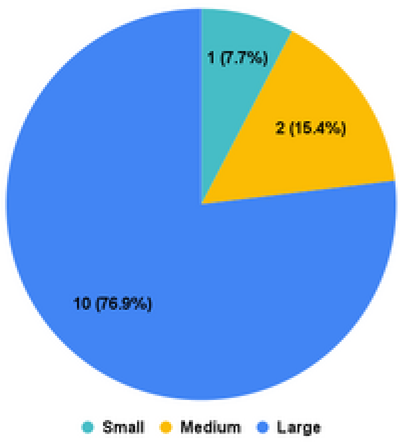

Figure 5.4.: Interview Participants by Organisation Size

Most of our interview participants were substantially well-versed with MSA. The majority i.e. (4/13) participants possessed 6 years of experience with MSA. (3/13) participants possessed 5 years of experience with MSA. (3/13) participants possessed 4 years of experience with MSA. (1/13) participant possessed 3 years of experience with MSA. Lastly, (2/13) participants possessed 1 year of experience with MSA. Figure 5.5 visualises the number of participants by years of experience with MSA.

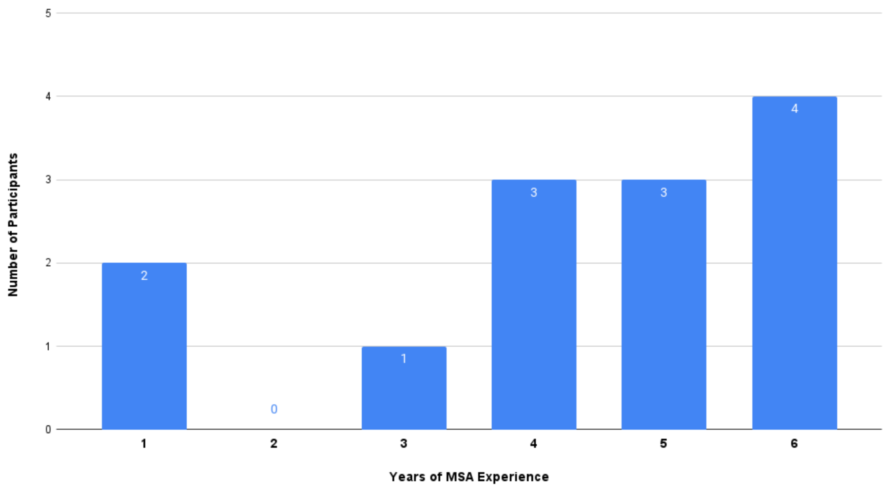

Figure 5.5.: Interview Participants by Years of MSA Experience

Out of the 13 interview participants, (7/13) belonged to industry while (6/13) belonged to academia, thus ensuring a considerably heterogeneous sample for the purposes of our evaluation. Figure 5.6 visualises the number of participants by field.

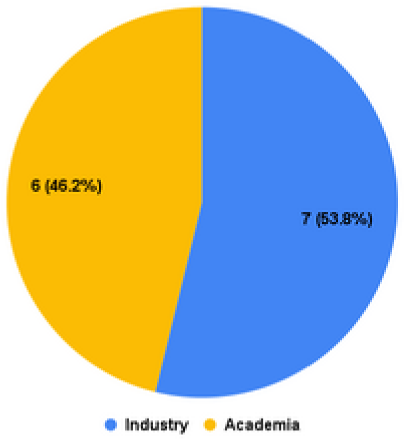

Figure 5.6.: Interview Participants by Field

The interview participants possessed different domain expertise within MSA. (6/13) participants possessed expertise in academic research. (4/13) participants possessed expertise in software architectures. (2/13) participants possessed expertise in DevOps. Lastly, (1/13) participant possessed expertise in data architectures. Figure 5.7 visualises the number of participants by expertise.

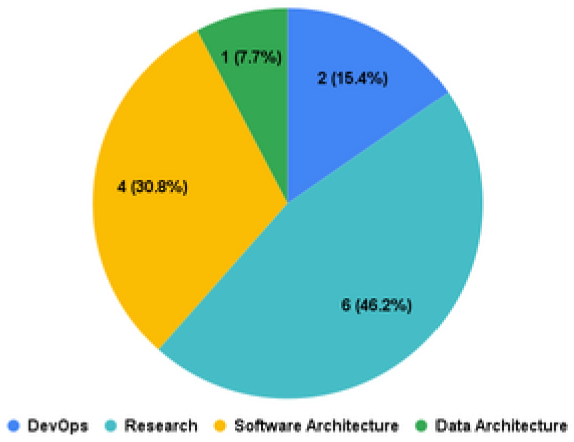

Figure 5.7.: Interview Participants by Expertise

5.2.6. Feedback Document

With the interview recordings of the (11/13) participants who agreed to it at our disposal, we engaged in *Note writing* which is the process of writing detailed notes from the interview recordings no later than a day after the interview has been conducted [Mul14]. We transcribed the answers to the posed questions in a systematic manner such that the transcription followed the same format in which the interview questions were structured. This led to the creation of the feedback document which served as the main evaluation artifact of the thesis and is available via GitLab[1]. The evaluation method was structured in such a way that we did not encounter conflicting opinions among the researchers and practitioners since each microservices professional assessed a distinct segment of the QM.

From the feedback document, we analysed all responses to Question 7, uncovering that (11/13) participants found the QM acceptable for their use cases. (1/13) participant found it perfectly acceptable in all aspects. Lastly, (1/13) participant remained neutral towards it. This strengthened our position in putting forward the QM as a research contribution in the right direction with room for improvements in its structure and descriptions. Figure 5.8 visualises the responses of Question 7 of the interview.

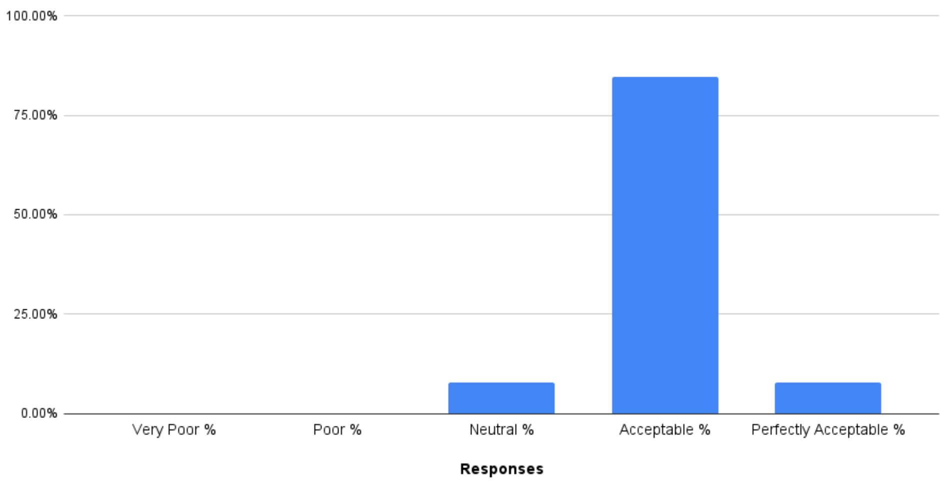

Figure 5.8.: Overall Appropriateness of the Preliminary Quality Model

[1] https://git.rwth-aachen.de/faizan.zafar/master-thesis/-/tree/main/Evaluation

5.3. Revision of Preliminary Quality Model

The evaluation feedback obtained from MSA experts allowed us to identify the important adjustments to consider in the QM. Hence, we revisited our preliminary QM and integrated the collected feedback into its structure as well as into the descriptions of its elements. This revision was meant to develop a simpler and better Quality Model than before as it included the opinions of seasoned researchers and practitioners.

After examining the feedback document, we proposed some modifications to the preliminary QM. We revised the QM in three sequential stages: the addition and removal of Quality Factors, changes in the positions of Quality Factors, and changes in the wordings of the proposed descriptions of the Quality Factors.

In the first stage, we added 2 and removed 14 Quality Factors in the structure of the QM. The descriptions of the 14 discarded QFs are available in Appendix A.1, with the novel Quality Factors marked in **bold**. Besides this, we marked 4 Quality Factors which were not considered appropriate at their current positions in the QM. Figure 5.9 visualises the execution of the first stage. The factors which were appended to the model are highlighted in green. The factors which were removed from the model are highlighted in red. The factors which were not considered appropriate at their current positions in the model are highlighted in yellow.

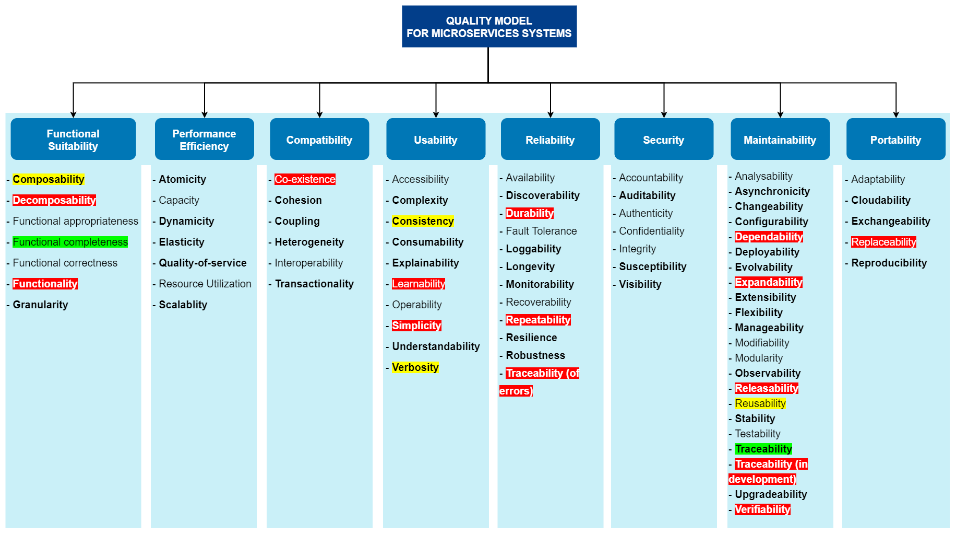

Figure 5.9.: Revision of the Preliminary Quality Model - First Stage

In the second stage, we changed the positions of the 4 Quality Factors which were not considered appropriate at their current positions in the model. A change in position meant that the QF under consideration was now being associated with another Quality Attribute than before as it describes the newly associated QA in a better way. Figure 5.10 visualises the execution of the second stage. The factors which were migrated to their updated positions in the model are highlighted in yellow.

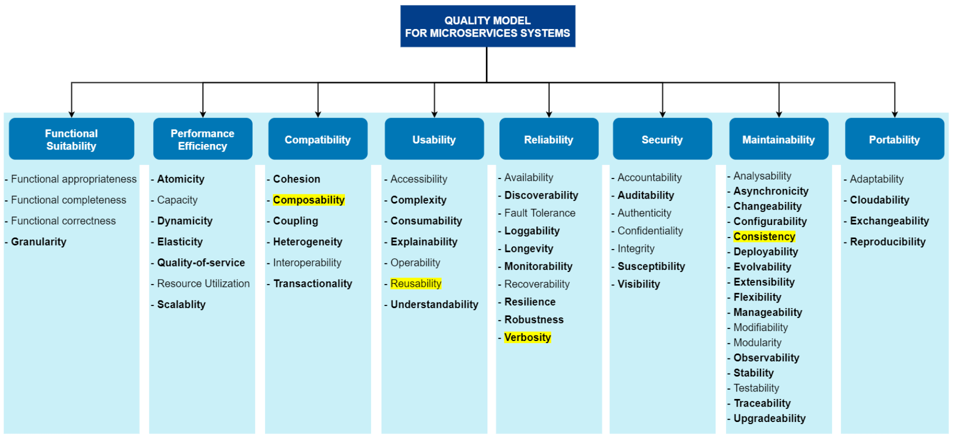

Figure 5.10.: Revision of the Preliminary Quality Model - Second Stage

Finally, in the third stage, we edited and proofread the proposed descriptions of the Quality Factors based on the collected feedback with the objective of having as polished, sharp and relevant descriptions as possible.

5.4. Revised Quality Model

After the revision procedure, we finally obtained the revised Quality Model that we proposed as a novel contribution of the thesis. The revised QM followed the same hierarchical structure as that of the preliminary model shown in Section 5.1. Moreover, since we did not add or remove any of its attributes, the 8 Quality Attributes of the preliminary model remained unchanged in the revised model. The revised QM encompassed 62 Quality Factors, including (41/62) novel Quality Factors and (21/62) Quality Factors which were the same as the existing QFs of the original ISO/IEC 25010 QM. The structure of the model is visualised in Figure 5.11, with the novel Quality Factors marked in **bold**.

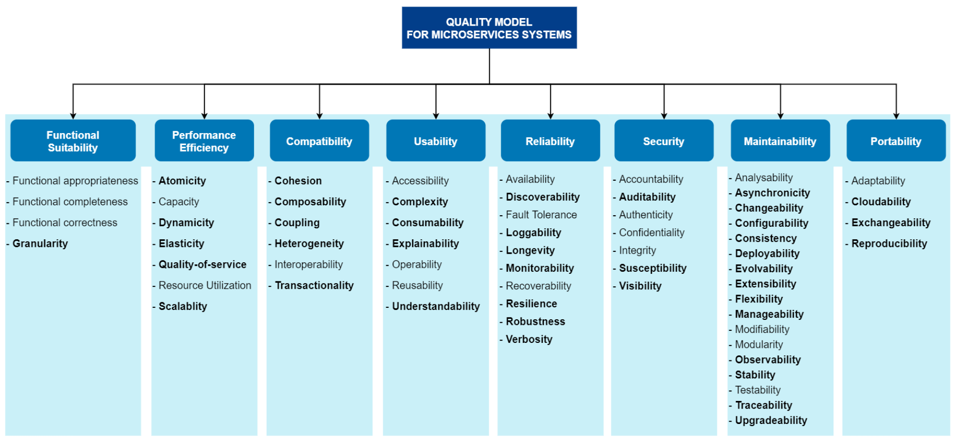

Figure 5.11.: A Quality Model for MSA

Figure 5.12 visualises the ratio of the Quality Factors by their novelty type.

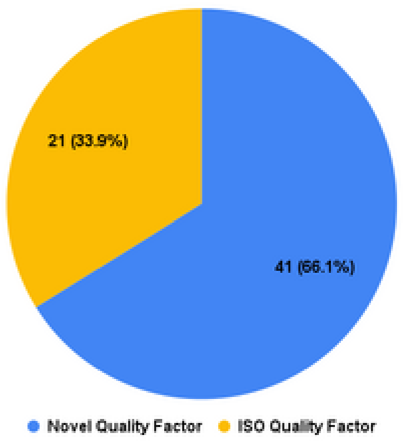

Figure 5.12.: Ratio of Quality Factors by Type

The descriptions of the Quality Attributes of the revised QM are available in Tables 5.5, 5.6.

The descriptions of the Quality Factors of the revised QM can be found in Tables 5.7, 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15, with the novel Quality Factors marked in **bold**. These tables should be read from left to right. For example, the first row in Table 5.7 presents the Quality Factor "Cohesion" which explains the Quality Attribute "Compatibility". Next, we can find the description of the "Cohesion" factor. Lastly, we see the reference numbers of the literature source(s) of the SLR corpus which mention or discuss this factor. The method explained to read this row can be applied to all rows of this and subsequent tables.

Quality Attribute	Description	Literature Source(s)
Compatibility	Degree to which a microservices system exchanges information with other systems (or components) and performs required functions while sharing the same environment.	[34],[49],[51],[54],[5],[7],[14]
Functional Suitability	Degree to which a microservices system provides services that meet the stated and implied needs when used under specified conditions.	[10],[26],[28],[42],[49],[59]
Maintainability	Degree of effectiveness and efficiency with which a microservices system can be modified according to changes in its environment and requirements.	[25],[26],[28],[32],[33],[34],[35],[42],[43],[45],[49],[50],[51],[58],[60],[63],[5],[6],[9],[12],[16],[21],[22],[46],[52]
Performance Efficiency	The performance relative to the amount of resources used in a microservices system under stated conditions.	[10],[15],[18],[23],[24],[25],[26],[28],[30],[31],[32],[34],[35],[37],[38],[39],[40],[41],[42],[43],[49],[50],[51],[54],[55],[57],[58],[59],[60],[62],[6],[7],[12],[14],[19],[20],[29],[44],[46],[53],[61]

Table 5.5.: Quality Attributes with Descriptions

Quality Attribute	**Description**	**Literature Source(s)**
Portability	Degree of effectiveness and efficiency with which a microservices system can be moved from one environment to another with minimal modifications.	[18],[27],[28],[32],[40],[49],[59],[62],[13],[16],[52]
Reliability	Degree to which a microservice operates independently for a specified period of time, regardless of whether other microservices in the system crash, or are attacked or destroyed.	[17],[26],[27],[28],[30],[32],[33],[35],[37],[38],[41],[42],[43],[49],[51],[54],[58],[59],[62],[1],[2],[5],[9],[14],[20],[29],[44],[53]
Security	Degree to which a microservices system guarantees data protection and resource access for stakeholders according to their type and level of authorization.	[10],[15],[18],[24],[28],[32],[33],[34],[35],[38],[39],[40],[41],[42],[43],[45],[47],[49],[50],[51],[54],[57],[59],[62],[5],[6],[9],[14],[44],[53]
Usability	Degree to which a microservices system is useful for specified users to achieve specified goals with effectiveness, efficiency and satisfaction, in a specified context of use.	[34],[43],[45],[49],[59],[11]

Table 5.6.: Quality Attributes with Descriptions (continued)

Quality Attribute	Quality Factor	Description of Quality Factor	Literature Source(s)
Compatibility	**Cohesion**	Measure of the strength of the relationships among the programming entities implementing a microservice and the functionality provided by the microservice.	[3],[17],[26],[28],[31],[37],[39],[40],[51],[54],[55],[58],[60],[62],[63],[9],[53]
Compatibility	**Composability**	Degree to which microservices are able to be combined with other microservices to form composite services, without high coupling on the interface layer.	[3],[10],[40],[42],[58],[63]
Compatibility	**Coupling**	Measure of the power of interactions and dependencies between microservices in a microservices system.	[3],[10],[17],[18],[26],[27],[28],[31],[33],[34],[35],[37],[38],[39],[40],[42],[47],[49],[50],[51],[54],[55],[57],[58],[60],[62],[63],[2],[4],[5],[9],[12],[19],[22],[36],[53]
Compatibility	**Heterogeneity**	Degree of diversity in programming languages and development frameworks in a microservices system.	[17],[23],[26],[39],[40],[51],[1],[2],[5],[9],[12],[22],[44]
Compatibility	Interoperability	Degree to which a microservice can communicate with other microservices in a microservices system, such that it supports the inclusion of microservices from other systems, including any exchange of information.	[15],[24],[25],[31],[34],[43],[49],[54],[58],[63],[2],[5],[8],[12],[48],[53]
Compatibility	**Transactionality**	Degree to which a distributed transaction can be achieved across different microservices in a microservices system.	[34],[54],[62]

Table 5.7.: Quality Factors with Descriptions

Quality Attribute	**Quality Factor**	**Description of Quality Factor**	**Literature Source(s)**
Functional Suitability	Functional appropriateness	Degree to which microservices, including interactions between the services, are collectively focused on capturing the complete set of business goals and requirements in order to implement the required features of a microservices system.	[43],[8]
Functional Suitability	Functional completeness	Degree to which the set of functions encapsulates a well-defined set of functionality that covers the specified task and user objective of a single microservice in a microservices system.	[15],[17],[23],[24],[25],[26],[27],[28],[30],[31],[32],[33],[34],[43],[50],[51],[58],[60],[62],[63],[2],[9],[14],[16],[19],[21],[52],[53],[56]
Functional Suitability	Functional correctness	Degree to which a microservices system delivers results with the needed degree of accuracy.	[31],[32],[7],[9]
Functional Suitability	Granularity	The number of fine-grained functionalities encapsulated in a single microservice.	[3],[24],[26],[30],[38],[39],[40],[43],[49],[51],[54],[58],[60],[62],[63],[9],[11],[19],[29],[46],[53],[56],[61]
Maintainability	Analysability	Degree to which it is possible to be completely certain about the bounded context for which a microservice is responsible for by reading and understanding its codebase and documentation.	[9]

Table 5.8.: Quality Factors with Descriptions (continued)

Quality Attribute	Quality Factor	Description of Quality Factor	Literature Source(s)
Maintainability	**Asynchronicity**	Degree to which microservices are able to publish and consume their messages to and from the communication channel, in a queue-based environment.	[24],[37],[47]
Maintainability	**Changeability**	Degree to which making enhancements in a microservices system is generally supported.	[26]
Maintainability	**Configurability**	Degree to which system modifications are easily accommodated in a microservices system, with minimal changes to its existing configuration.	[8]
Maintainability	**Consistency**	Degree to which the changes made to the data in one microservice are available to all other microservices in a microservices system.	[27],[31],[38],[40],[42], [54],[62],[9],[53]
Maintainability	**Deployability**	Degree to which microservices can be deployed independently without downtime, and without restarting the entire microservices system.	[10],[15],[26],[27],[28],[30], [33],[57],[60],[7],[11],[12] ,[46]
Maintainability	**Evolvability**	Degree of effectiveness and efficiency with which a microservice can evolve without hindering the evolution of other microservices in a microservices system.	[17],[24],[26],[30],[34],[35], [42],[45],[54],[60],[5],[9] ,[12]
Maintainability	**Extensibility**	Degree of effectiveness and efficiency with which a microservices system can be extended in additional use cases or resources.	[34],[42],[43],[49],[50], [55],[59]

Table 5.9.: Quality Factors with Descriptions (continued)

Quality Attribute	**Quality Factor**	**Description of Quality Factor**	**Literature Source(s)**
Maintainability	**Flexibility**	Degree to which it is possible to build a microservices system for distributed environments including hybrid cloud.	[15],[18],[27],[28],[30],[31],[32],[35],[39],[40],[41],[43],[47],[49],[51],[54],[58],[59],[60],[1],[6],[9],[16],[19],[22],[52]
Maintainability	**Manageability**	Degree of effectiveness and efficiency with which the affairs of a microservices system can be governed automatically.	[27],[34],[43],[54],[59]
Maintainability	Modifiability	Measure of the speed at which a new change can be pushed to a microservice, without degrading quality.	[10],[15],[30],[34],[37],[45]
Maintainability	Modularity	Degree to which a microservices system is composed of components, such that a change to one component has minimal impact on other components.	[10],[17],[23],[26],[28],[32],[33],[40],[4],[8],[9],[13]
Maintainability	**Observability**	Degree of effectiveness and efficiency with which the internal states of a microservices system can be inferred from its external outputs.	[35],[39],[45],[11]
Maintainability	**Stability**	Degree to which a microservices system can avoid the unexpected effects of technical or behavioural modifications.	[34],[37],[45],[51],[57],[4],[5],[9]
Maintainability	Testability	Degree to which it is possible to establish test criteria and perform tests in a microservices system.	[10],[15],[18],[28],[32],[33],[39],[49],[58],[60],[12]

Table 5.10.: Quality Factors with Descriptions (continued)

Quality Attribute	Quality Factor	Description of Quality Factor	Literature Source(s)
Maintainability	**Traceability**	Degree to which it is possible to locate where in the microservices system an error originated, and how it travelled through several microservices.	[10],[41],[62]
Maintainability	**Upgradability**	Degree to which a microservices system can be improved in technical or behavioural aspects.	[17],[40],[43],[8]
Performance Efficiency	**Atomicity**	Degree to which each microservice in a microservices system is treated as a single unit and has a single responsibility.	[24],[31],[38],[39],[55]
Performance Efficiency	Capacity	Degree to which the maximum limit of a parameter relevant to a microservices system meets the specified requirements.	[28],[32],[54],[55],[59],[53]
Performance Efficiency	**Dynamicity**	Degree to which the physical or logical structure of a microservices system evolves with time.	[26],[28],[31],[47],[2]
Performance Efficiency	**Elasticity**	Degree to which resources can be increased or decreased automatically or dynamically in a microservices system.	[18],[23],[28],[30],[40],[50], [54],[59],[62],[8],[19],[48]
Performance Efficiency	**Quality-of-service**	Measure of the communication requirements of a microservices system, usually expressed in terms of bandwidth, latency or data transfer rate.	[23],[26],[28],[32],[34],[47], [57],[58],[59],[5],[7]
Performance Efficiency	Resource Utilization	Degree to which the amount and type of resources used by a microservices system meets the specified requirements.	[19],[44]

Table 5.11.: Quality Factors with Descriptions (continued)

Quality Attribute	Quality Factor	Description of Quality Factor	Literature Source(s)
Performance Efficiency	**Scalability**	Degree to which new instances of microservices can be provisioned and deployed in a microservices system, while preventing the system from slowing down once the load gets higher.	[10],[17],[18],[23],[25],[26],[27],[28],[30],[31],[32],[33],[34],[35],[37],[38],[39],[40],[41],[42],[43],[45],[47],[49],[50],[51],[54],[55],[57],[59],[60],[62],[1],[6],[7],[8],[12],[16],[19],[21],[22],[29],[48],[52],[53],[61]
Portability	Adaptability	Degree to which a microservices system can be dynamically adapted for different environments.	[26],[40],[58],[46]
Portability	**Cloudability**	Degree to which the deployment characteristics of a microservices system make it a great match for the elasticity of the cloud.	[30]
Portability	**Exchangeability**	Degree to which a microservices system can be built independent of any cloud provider-specific API, domain-specific language, or deployment technology.	[52]
Portability	**Reproducibility**	Degree to which the behaviour of a microservices system can be achieved again in the same or different environments.	[47],[54],[16]
Reliability	Availability	Degree to which a microservices system is operational and usable when accessed by an authorized entity.	[10],[15],[17],[23],[25],[26],[27],[28],[31],[32],[34],[35],[38],[40],[41],[43],[47],[51],[54],[58],[59],[60],[62],[1],[5],[13],[20],[29],[44],[61]

Table 5.12.: Quality Factors with Descriptions (continued)

Quality Attribute	Quality Factor	Description of Quality Factor	Literature Source(s)
Reliability	**Discoverability**	Degree to which a microservice is able to locate other microservices or microservices instances currently running in the microservices system.	[28],[42]
Reliability	Fault Tolerance	Degree to which the presence of faults in one microservice does not break the operation of the entire microservices system.	[18],[25],[26],[28],[30],[31],[33],[39],[49],[51],[55],[62],[6],[9],[29]
Reliability	**Loggability**	Degree to which the activities occurring in a microservices system can be recorded in a log.	[10]
Reliability	**Longevity**	Degree to which a microservice continues to fulfill its purpose for a certain time span or if a defined set of conditions holds, without requiring excessive maintenance.	[34],[42]
Reliability	**Monitorability**	Degree to which the behavioural aspects (information, requests and errors) and performance aspects (system behaviour, CPU usage and response times) of a microservices system can be followed, when they travel through several microservices.	[10]
Reliability	Recoverability	Degree to which all data and microservices can be restored to a certain state in operation, from an event which has interrupted or taken down the microservices system.	[62],[6]
Reliability	**Resilience**	Degree to which a microservices system can continue an acceptable level of operations in case of system or service failures.	[26],[33],[35],[39],[41],[47],[54],[59],[62],[8],[9],[13],[16],[29]

Table 5.13.: Quality Factors with Descriptions (continued)

Quality Attribute	**Quality Factor**	**Description of Quality Factor**	**Literature Source(s)**
Reliability	**Robustness**	Degree to which a microservices system can tolerate entire microservices' failures, including communication failures, such that the failures don't propagate through the system.	[26],[28],[33],[40],[43], [51],[9]
Reliability	**Verbosity**	Degree to which all possibly relevant information can be preemptively captured in a microservices system for tracing or debugging.	[47]
Security	Accountability	Degree to which all actions performed by an entity (or resource) in a microservices system can be traced uniquely to that entity (or resource).	[2],[9]
Security	**Auditability**	Degree to which a microservices system keeps sufficient records to support specified financial or legal audits.	[18],[24],[45],[54]
Security	Authenticity	Degree to which the identity of an entity (or resource) in a microservices system can be proved as the identity it claims.	[8]
Security	Confidentiality	Degree to which a microservices system ensures that certain data is accessible only to authorized stakeholders.	[28],[41]
Security	Integrity	Degree to which consistency across data and resources can be maintained in a microservices system, in the event of a security breach.	[28],[31],[32],[38],[41],[51], [55],[59],[22],[53]
Security	**Susceptibility**	Degree to which a microservices system is exposed to the possibility of security incidents.	[41]
Security	**Visibility**	Degree to which all events that compromise business operations and information security in a microservices system can be closely monitored.	[41],[61]

Table 5.14.: Quality Factors with Descriptions (continued)

Quality Attribute	Quality Factor	Description of Quality Factor	Literature Source(s)
Usability	Accessibility	Degree to which a microservices system is convenient to use by users having diversity in capabilities and characteristics.	[43],[59],[29],[46]
Usability	**Complexity**	Degree of effort that is required to realise and implement the technical and behavioural aspects of a microservices system.	[3],[10],[17],[18],[23],[24], [25],[26],[27],[28],[30],[31], [33],[35],[37],[38],[39],[41], [42],[45],[47],[49],[50],[51], [54],[57],[58],[59],[60],[62], [5],[6],[8],[12],[20],[21], [22],[36],[44],[48],[52],[53]
Usability	**Consumability**	Degree to which the artefacts of a microservices system are perceivable by users having diverse backgrounds.	[34]
Usability	**Explainability**	Degree to which a microservices system is able to describe its own structure as well as behaviour.	[58],[44]
Usability	Operability	Degree to which a microservices system is easy to run and control.	[8]
Usability	Reusability	Degree to which the functionality or source code of components in a microservices system can be reused in the future.	[3],[10],[25],[26],[28],[32], [35],[37],[39],[42],[43],[58], [59],[60],[62],[63],[1],[5], [9],[16]
Usability	**Understand-ability**	Degree to which a microservices system enables its users to comprehend its suitability for particular tasks and specified context of use.	[28],[32],[45],[51],[44]

Table 5.15.: Quality Factors with Descriptions (continued)

6. A Catalogue of Design Guidelines for Microservices Architectures

This chapter presents our contribution in the form of a structured catalogue of design guidelines that are centered around constructing MSA. We formally introduce our results and analyse them in this chapter.

6.1. Catalogue Categorisation Scheme

The research approach described in Chapter 4.3.1 yielded a three-dimensional categorisation scheme for the Guidelines Catalogue, which introduced the dimensions *Type*, *Scope* and *Design*:

Definition 5. Type *represents the semantic group of the design guideline.*

Definition 6. Scope *describes the area which the design guideline is responsible for.*

Definition 7. Design *reports the type of activity that the design guideline undertakes with respect to designing the system architecture.*

The defined dimensions encompassed a total of 23 categories which are outlined in Table 6.1.

Type	**Scope**	**Design**
Design Pattern	Architecture	Migration
Design Principle	Code Management	Universal
Best Practice	Communication	
Context-sensitive	Data Consistency	
Best Practice	Data Management	
	Data Persistence	
	Decomposition	
	Deployment	
	Development	
	Distribution	
	Entry Point	
	Fault Tolerance	
	Infrastructure	
	Monitoring	
	Security	
	Supplementals	
	Testing	

Table 6.1.: Three-Dimensional Categorisation Scheme

First, the *Type* dimension encompasses the *Design Pattern*, *Design Principle*, *Best Practice*, and *Context-sensitive Best Practice* categories:

Definition 8. *A* Design Pattern *explains the solution to a well-defined problem, in such a way that we are able to use the solution repeatedly. We used nouns as a naming schema for the design patterns in the catalogue (e.g. API Gateway). This suggests that the pattern is an entity to be used, employed or observed.*

Definition 9. *A* Design Principle *constitutes a set of prescribed considerations that support consistency in design decisions. Design principles can be realised in a variety of ways, given the system constraints, implementation language, existing framework or component support, and cost/benefit trade-offs. We used a mix of nouns and verbs as a naming schema for the design principles in the catalogue, depending on the context (e.g. Single Responsibility Principle, Evolutionary design etc.)*

Definition 10. *A* Best Practice *serves as a means for appropriate adherence to established rules, along with detecting deviations from their adherence. It is meant to be used by any means and it always holds. We used imperatives as a naming schema for the best practices in the catalogue in order to portray them as concrete directives to be followed (e.g. Avoid direct synchronous calls between services). However, in some cases, a best practice takes advantage of a design pattern and its naming convention implies what should be practically done with the design pattern. As an example, we propose that* *the Asynchronous messaging pattern leads to application of the Avoid direct synchronous calls between services practice.*

Definition 11. *A* Context-sensitive Best Practice *is a best practice that only proves to be optimal in certain practical contexts or complex situations. It is also highly dependent on the specific use case which the microservices system is responsible for, hence it should not necessarily always be used. We chose to use imperatives as a naming schema for the context-sensitive best practices in the catalogue and composed them as advice (e.g. Limit language diversity).*

The *Context-sensitive Best Practice* category is a specialisation of *Best Practice* due to pragmatic reasons and dependency on the use case. For example, if the software architect wishes to actively control all elements and interactions in a microservices system, they ought to adopt the *Orchestration* pattern. While this pattern promotes tight coupling, it is the current defacto standard in microservices systems and not using it might come as a surprise to developers working with MSA. On the other hand, if the software architect wants to establish a routine that microservices follow without requiring supervision, they ought to employ the *Choreography* pattern. This makes it clear that in one situation adopting *Choreography* is presumed to be the better choice, while in the other employing an *Orchestration* should be chosen.

Next, the *Scope* dimension encompasses 17 categories as described in Table 6.2:

Scope	**Description**
Architecture	Related to the architectural properties of a microservices system.
Code Management	Related to source code management and versioning in a microservices system.
Communication	Related to the interaction and coordination between microservices in a microservices system.
Data Consistency	Related to the consistency of data across microservices in a microservices system.
Data Management	Related to the management and utility of data resources in a microservices system.
Data Persistence	Related to improving the management of databases in a microservices system.
Decomposition	Related to the service decomposition principles.
Deployment	Related to the deployment mechanisms and practices in a microservices system.
Development	Related to the coding and implementation aspects of a microservices system.
Distribution	Related to improving the distribution of microservices in a logical way.
Entry Point	Related to supplying the control access and entry point to services or types of backends.
Fault Tolerance	Related to solving problems around tolerating faults and error handling.
Infrastructure	Related to the infrastructure-level properties of a microservices system.
Monitoring	Related to the monitoring and post-deployment activities in a microservices system.
Security	Related to the security principles of a microservices system.
Supplementals	Related to complementing and improving an existing design guideline of a microservices system.
Testing	Related to quality assurance and testability in a microservices system.

Table 6.2.: Categories of *Scope* dimension

Lastly, the *Design* dimension encompasses the *Migration* and *Universal* categories:

Definition 12. *A* Migration *guideline is exclusively meant to be used for the migration from monolithic architectures to MSA.*

Definition 13. *A* Universal *guideline is generally applicable and always holds in microservices design and development. It is explicitly used in designing microservices from* *scratch or building cloud native architectures.*

We understand that, in reality, an individual design guideline may portray multiple ideas, may be responsible for multiple areas across a microservices system, and may design the system using multiple strategies. However, considering the required level of detail with respect to the categorisation of the catalogue for the purposes of this thesis, we proposed a unitary categorisation scheme where only one category per dimension was assigned to each design guideline.

6.2. Catalogue of MSA Design Guidelines

As a result of the research methods described in Chapter 4.3 and 4.5, and applying the categorisation scheme from Section 6.1, we obtained a structured catalogue of 239 design guidelines, including best practices, design patterns, and design principles, that serve to build meaningful MSA. The catalogue comprises the names and descriptions of the design guidelines as well as the information of their mapped quality characteristics. The complete catalogue is available via GitLab[1]. A preview of the first two rows of the catalogue is shown in Table 6.3. The catalogue should be read from left to right. The first row presents a design guideline with the serial number ”G1”. This guideline is of the ”Best Practice” type, having the ”Architecture” scope, and pertaining to the ”Migration” design. Then, we have the name of the guideline i.e. ”Administer rehosting” followed by its concrete and concise description. Next, we observe the mapped (affecting) quality characteristics of this design guideline i.e. ”Accessibility, Scalability”. Lastly, we see the reference numbers of the literature source(s) of the SLR corpus which mention or discuss this guideline. The method explained to read this row can be applied to all rows of the catalogue.

[1]https://git.rwth-aachen.de/faizan.zafar/master-thesis/-/tree/main/Catalogue

Guideline (G#)	Type	Scope	Design	Name	Description	Affecting Quality Characteristic(s)	Literature Source(s)
G1	Best Practice	Architecture	Migration	Administer rehosting	Move a legacy system from one platform to a more modern alternative, with minimal changes.	Accessibility, Scalability	[58]
G2	Best Practice	Architecture	Universal	Avoid vendor lock-in	Choose vendor-neutral interfaces and technologies to avoid vendor lock-in.	Evolvability, Performance Efficiency, Deployability, Portability, Resource Utilization	[5],[27],[51], [52]

Table 6.3.: Guidelines Catalogue (Preview)

6.3. Analysis

In order to get a high level overview, we analysed the catalogue with respect to the number of design guidelines within each category. In the first level of categorisation by the *Type* categories, we reported (126/239) design patterns, (40/239) design principles, (49/239) best practices, and (24/239) context-sensitive best practices, as shown in Figure 6.1.

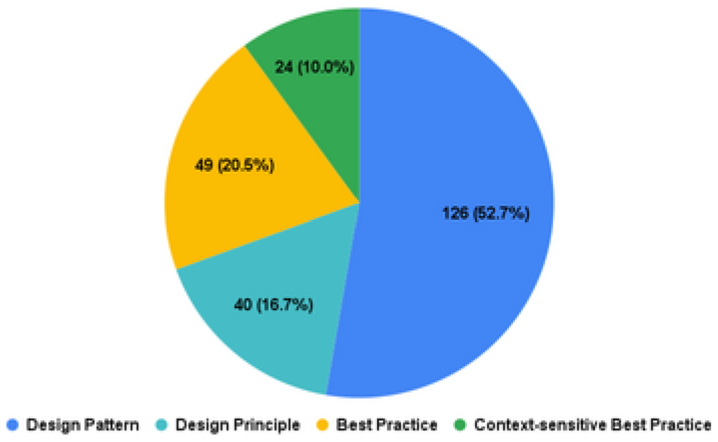

Figure 6.1.: Frequency of Design Guidelines by *Type*

Next, in the second level of categorisation by the *Scope* categories, we reported (33/239) Architecture, (8/239) Code Management, (30/239) Communication, (4/239) Data Consistency, (7/239) Data Management, (15/239) Data Persistence, (12/239) Decomposition, (9/239) Deployment, (23/239) Development, (18/239) Distribution, (5/239) Entry Point, (15/239) Fault Tolerance, (17/239) Infrastructure, (9/239) Monitoring, (9/239) Security, (20/239) Supplementals, and (5/239) Testing design guidelines, as shown in Figure 6.2.

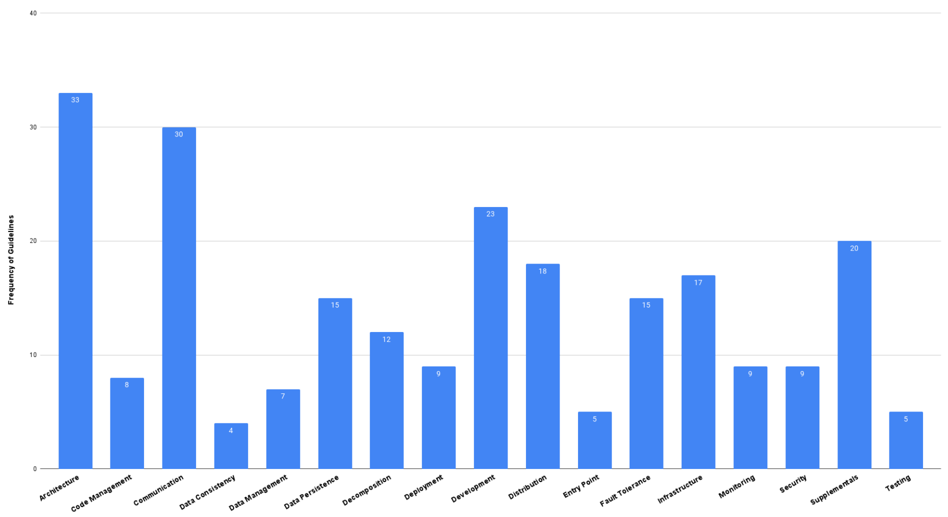

Figure 6.2.: Frequency of Design Guidelines by *Scope*

Finally, in the third level of categorisation by the *Design* categories, we reported (28/239) Migration and (211/239) Universal design guidelines, as shown in Figure 6.3.

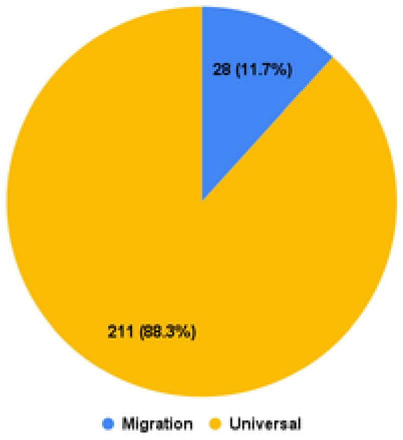

Figure 6.3.: Frequency of Design Guidelines by *Design*

With respect to the mapping of the design guidelines to the affecting quality characteristics, we realised that a significant number of design guidelines mapped to multiple quality characteristics. This occurred because a design guideline may be targeted towards improving a certain facet of quality at a given time, with the possibility to cater to other aspects of quality at a later point in time. This also signifies how much each quality characteristic is relevant considering its trade-offs at a given time. We analyse further aspects of the catalogue in detail in Chapter 7.1.

7. Discussion

In this chapter, we highlight and review the key findings from the results of the thesis. Next, we answer all the research questions of the thesis. We also consider the implications of the results of this work. Lastly, we discuss the threats to validity concerning this research.

7.1. Results Findings

We analysed the meta-information of the SLR corpus which we obtained in Chapter 4.2.6 in order to portray a valid understanding of the state of academic research. Figure 7.1 illustrates the number of publications in the SLR corpus broken down by year of publication. From 2015 onwards, we observed a steep rise in the annual number of publications, reaching an all-time high in 2021. These phenomena seem to closely coincide with the increasing adoption of microservices for various use cases across the industry and the rising trend in the publication of academic literature relevant to MSA in the past years.

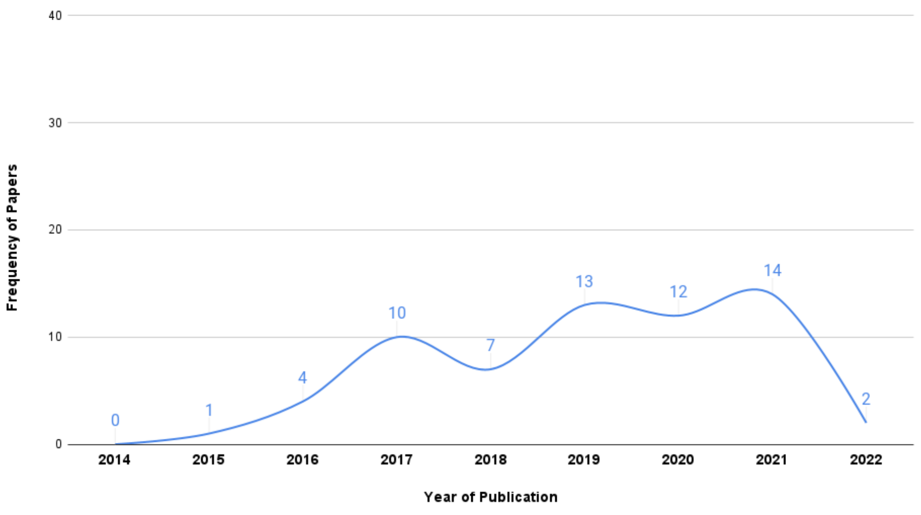

Figure 7.1.: Frequency of Scholarly Papers by Year in the SLR corpus

During the data extraction process in Chapter 4.3, we came across discussions on a variety of topics that we deemed meaningful and important to refer to when studying the related work of this thesis. These topics could also be beneficial when pursuing the future work of the thesis. We listed such topics along with their literature sources in Table 7.1.

Topic	**Literature Source(s)**
Monolith to MSA Migration	[6],[9],[10],[17],[24],[25],[26],[29],[30], [35],[48],[51],[55],[56],[57],[58],[59], [60],[61],[62],[63]
Quality in Microservices Applications and/or Architectures	[3],[5],[9],[10],[15],[17],[18],[23],[24], [25],[26],[27],[28],[30],[31],[32],[33], [34],[35],[37],[38],[39],[40],[41],[42], [43],[45],[47],[49],[50],[51],[54],[55], [57],[58],[60],[62],[63]
SLR/GLR	[2],[3],[9],[10],[14],[31],[35],[41],[49], [59],[62],[63]

Table 7.1.: Important Topics found in the SLR Corpus

Despite the high number of guidelines reported in the Guidelines Catalogue, some guidelines represented specialised implementations (or variations) of other known guidelines. For clarity purposes, we specified this property of the variants in their descriptions within the catalogue itself. For example, the *Gateway Aggregation* and *Gateway Offloading* patterns are specialised versions of the *API Gateway* pattern. Additionally, we came across a few MSA design guidelines which were in common with SOA such as *Asynchronous Messaging*, *Service Discovery*, and *Circuit Breaker*. This makes sense since the MSA pattern is a subset of the SOA paradigm [Neu15].

Design Patterns dominated the catalogue while *Best Practices* and *Design Principles* were significantly less in number respectively. This can be attributed to the extensive academic research that has been conducted on microservices design patterns and their rising interest in industry over the years, similar to the trend observed in Figure 7.1. *Context-sensitive Best Practices* turned out to be the least frequent guidelines, and rightfully so, since we regard them as specialised practices pertaining to certain use cases and subject to conditions. We also observed that the most frequently reported design guidelines were related to the architectural properties followed by the ones related to the interaction and coordination between microservices. Overall, some of the most frequently occurring guidelines were *Independent and automated deployment (including independent development)*, *Isolation of failures* and *Lightweight containerization* in the catalogue.

During our research, we noticed that a significant amount of publications use the terms *best practice*, *design pattern*, and *design principle* interchangeably. Although widely differing from a theoretical perspective, these terms are not clearly distinguishable among microservices professionals at all times. As a result, we witnessed that sometimes a particular guideline was either classified in multiple categories or incorrectly classified in some category. For example, in multiple papers, the guideline *Continuous Integration/-Continuous Delivery* was designated as a design pattern. We analysed CI/CD and found that it serves as a collection of well-defined activities related to agile software development. It proposes a set of actions and tools for code integration, automating processes, running test suites, and building deployment artefacts. This definition and description implied that this guideline does not explain the solution to a well-defined problem, and hence should not be classified as a design pattern. Moreover, it also demonstrates the lack of a clear naming and definition convention for this guideline. Based on this analysis, we were inclined to consider CI/CD as a best practice, reframing it according to the appropriate naming convention. This was an example of how our research classified the design guidelines as well as proposed standardised naming schema for the reported best practices, design patterns, and design principles respectively. As a result, the proposed classification and the naming schema majorly helped in establishing structure in the catalogue and served as novel contributions of this thesis.

We realised that the categorisation of the design guidelines by categories of *Type* dimension is not strictly sharp. We marked in the catalogue with an asterisk (*) such guidelines whose category assignment requires further investigation. For example, the catalogue mentions *Choreography* as a design pattern and *Employ Choreography* as a context-sensitive best practice. Both of these guidelines refer to the same underlying strategy with the difference of context. A similar observation we made was that for some of the design guidelines, we did not anticipate their strict categorisation as explored in the literature. For example, we would not expect *Bounded Context* as a design pattern for microservice systems. The fact that *Bounded Context* is referred to as a design pattern for Microservices Architectures in the literature shows that a meaningful classification is needed in order to differentiate between design patterns such as *Bounded Context* and design patterns such as *API Gateway* for MSA.

The mapping of the design guidelines to the quality characteristics is a direct consequence of the research work performed for answering RQ1 and RQ2. Tables 7.2, 7.3 list the frequencies of the occurrences of the quality characteristics in the Guidelines Catalogue, along with the percentage of coverage they entail.

Quality Characteristic	**Frequency of Occurrence**	**Coverage (%)**
Complexity	103	8.64%
Maintainability	82	6.88%
Coupling	64	5.37%
Security	62	5.20%
Reliability	58	4.87%
Performance Efficiency	56	4.70%
Scalability	55	4.61%
Evolvability	53	4.45%
Resource Utilization	52	4.36%
Availability	43	3.61%
Granularity	31	2.60%
Functional completeness	29	2.43%
Consistency	28	2.35%
Deployability	26	2.18%
Understandability	23	1.93%
Flexibility	22	1.85%
Quality-of-service	22	1.85%
Modifiability	21	1.76%
Compatibility	18	1.51%
Fault Tolerance	17	1.43%
Testability	17	1.43%
Interoperability	16	1.34%
Manageability	16	1.34%
Resilience	15	1.26%
Auditability	14	1.17%
Modularity	14	1.17%
Reusability	14	1.17%
Observability	13	1.09%
Stability	13	1.09%
Transactionality	12	1.01%
Usability	12	1.01%
Monitorability	10	0.84%
Traceability	10	0.84%
Cohesion	9	0.76%
Asynchronicity	8	0.67%
Loggability	8	0.67%

Table 7.2.: Frequencies and Coverage of Quality Characteristics in the Guidelines Catalogue

Quality Characteristic	**Frequency of Occurrence**	**Coverage (%)**
Upgradability	8	0.67%
Elasticity	7	0.59%
Extensibility	7	0.59%
Changeability	6	0.50%
Functional correctness	6	0.50%
Functional Suitability	6	0.50%
Heterogeneity	6	0.50%
Longevity	6	0.50%
Portability	6	0.50%
Authenticity	5	0.42%
Configurability	5	0.42%
Accessibility	4	0.34%
Composability	4	0.34%
Discoverability	4	0.34%
Integrity	4	0.34%
Reproducibility	4	0.34%
Robustness	4	0.34%
Adaptability	3	0.25%
Cloudability	3	0.25%
Dynamicity	3	0.25%
Explainability	3	0.25%
Functional appropriateness	3	0.25%
Recoverability	3	0.25%
Susceptibility	3	0.25%
Analysability	2	0.17%
Operability	2	0.17%
Verbosity	2	0.17%
Accountability	1	0.08%
Atomicity	1	0.08%
Capacity	1	0.08%
Confidentiality	1	0.08%
Consumability	1	0.08%
Exchangeability	1	0.08%
Visibility	1	0.08%

Table 7.3.: Frequencies and Coverage of Quality Characteristics in the Guidelines Catalogue (continued)

Interestingly, we observed in the catalogue that *Complexity*, *Coupling*, *Scalability*, *Evolvability*, *Resource Utilization*, *Availability*, were the most frequently mapped quality characteristics. We attribute this high frequency to the easy and fast deployment principle of MSA, and to the objective of MSA to deliver high quality satisfaction. On the other hand, *Accountability*, *Atomicity*, *Capacity*, *Confidentiality*, *Consumability*, *Exchangeability*, *Visibility* were among the least mapped quality characteristics. Our results proved to be in line with prior research on the topic. According to [BCK12], the starting point for the quality attributes of an architecture can be defined in terms of *performance*, *availability* and *maintainability.* According to the literature study in [Di 17], *performance* and *maintainability* are the most investigated Quality Attributes for Microservices Architectures. Lastly, [Li17] reveals a significance on *performance*, *maintainability* and *reliability* as the most emphasised Quality Attributes for MSA in research.

7.2. Research Questions Findings

In our research, we forged three research questions around which the whole thesis was framed and that were defined towards the beginning of the thesis. The first research question revolved around design guidelines of MSA and was structured as follows:

RQ1: Which design guidelines exist for constructing meaningful Microservices Architectures?

To answer this question, we performed a Systematic Literature Review to gather the existing design guidelines associated with constructing meaningful MSA (refer to Chapter 4.2). This led to the creation of a Guidelines Catalogue comprising 239 design guidelines, including best practices, design patterns, and design principles (refer to Chapter 6). This research allowed us to gain perspective concerning the well-rounded interest in microservices design guidelines.

The second research question was centered around the quality of MSA and was framed as follows:

RQ2: Which quality characteristics adequately describe the quality of Microservices Architectures and how can they be meaningfully represented in the form of a Quality Model?

To answer this question, we proposed a Quality Model for MSA which explained 70 quality characteristics that collectively describe the quality of MSA meaningfully, and illustrated how they they are interrelated (refer to Chapter 5).

Finally, the third research question focused on consolidating the previous research and was devised as follows:

RQ3: Which quality characteristics are affected by the existing design guidelines of Microservices Architectures?

To answer this question, we performed a mapping to identify which of the 70 quality characteristics of the proposed QM are related to and affected by each of the 239 design guidelines collected in the catalogue. We only used academic literature for the mapping. The mapping was done in the Guidelines Catalogue in deriving its final version. In this way, the third research question holistically unites the main research artefacts of the thesis.

7.3. Implications of Results

Our research relies on the notion that although the microservices technology possesses great potential, it has not reached its anticipated success. From this, we deduced that knowledge on how to design and develop microservices applications and architectures optimally and feasibly has not been adequately structured, defined or standardised. In addition, the problem lies with the inability to harness the guidelines collectively (e.g. using design guidelines A and B together in order to utilise the maximum potential of design guideline C), and the context-sensitivity i.e. some guidelines can only be observed in certain circumstances. As a result, it becomes difficult for microservices developers to realise the stakeholders' requirements, and hence they are unable to deliver required value. This can often result in financial loss for the stakeholder, and can negatively impact the relationship between the developer and the stakeholder in the long run. To mitigate this, our SLR approach unified knowledge in providing a diverse and broad range of design guidelines targeted towards MSA. We regard the Guidelines Catalogue as an important first step towards the standardisation of building MSA while catering to specific quality requirements of stakeholders. We did not find such an in-depth catalogue for the MSA domain scientifically devised in prior research. Thus, this work is a first of its kind.

It is to be noted that not all reported design guidelines were proven to improve the quality of MSA at a given time under a given constraint. For example, *Database per service* was reported as a best practice in literature, however it might not be pragmatic to implement in most organisations due to resource constraints. As an alternative, *Shared Database* was proposed as a recommended solution because, although being an anti-pattern, it improves the performance of the architecture. Hence, this work provides a strong foundation for determining the precise affect (improvement or deterioration) of the MSA design guidelines on the corresponding quality characteristics.

A holistic framework on the quality of MSA has been missing in academia and industry. By investigating the quality characteristics relevant to MSA and proposing the QM, we complement the existing research that has focused predominantly on a variety of software QMs. With our work, we provide a credible background to gain a broader picture of the quality characteristics present in the context of Microservices Architectures. Our research approach is methodologically innovative as we extended the ISO/IEC 25010 Quality Model from [ISO11] using a variety of findings on quality characteristics relevant to MSA. At the time of this research, we were not aware of any other Quality Model that was built around MSA. Hence, the presented work is, to the best of our knowledge, a novel research contribution and the only one of its kind.

During the evaluation phase, we obtained valuable feedback both with respect to the structure of the Quality Model and the descriptions it offers. We converted this feedback into a proposition of an improved model to enhance its usability and quality. Since our evaluation interviews were semi-structured, we harnessed the freedom to dive into details with the participants when needed. Due to this freedom and owing to our heterogeneous sample, we believe that the evaluation results in the finally proposed Quality Model accurately reflect the views of the practitioners and researchers.

Our research led us to suspect a considerable gap in the knowledge of academia and industry. We felt that the current research paradigm does not adequately cover all facets of MSA to the required degree of depth. We attribute this to the novelty of the field and its only recent adoption in industry and academia. Additionally, we acknowledged that none of the interview participants who were industry practitioners mentioned the use of research papers or academic literature for their work. From this, we deduced that the real industrial problems and challenges are not sufficiently and appropriately addressed in academic research. The lack of discussion in academia on the quality of MSA and the lack of quality validation on real enterprise-scale systems is a problem that hinders knowledge transfer between academia and industry.

7.4. Threats to Validity

To determine the threats to validity, we followed the definitions from [Woh+12] and considered the internal and external threats to validity in relation to the entirety of the results of this thesis.

7.4.1. Internal Validity

The internal validity studies the trustworthiness of the relationship established between the research study and its results. One aspect to consider is our own possible biases in the results of the SLR. We perceived a threat regarding selection bias towards the inclusion, exclusion and selection criteria of the SLR. To mitigate this, we reported and consulted all criteria internally. For the initially proposed Quality Model, we performed an evaluation during which we reviewed and consulted the results with experts in the field. This approach was meant to minimise the impact of subjectivity in the model.

7.4.2. External Validity

The external validity considers the generalisability of the research results to other situations. Since we comprehensively described and documented the SLR method, we believe that the complete study could be reproduced as is. Another aspect to consider is the limited scope and amount of resources used for the SLR. To mitigate this, we considered as many possible papers that resulted from the application of the SLR method.

We anticipated personal biases in the participants' statements and answers during the evaluation interviews as a threat. Participants might comment in affirmation of our proposed results, which could lead to confirmation bias. Participants might also be scared to give an incorrect answer. We assumed these threats to be low since the participants did not seem to be worried about highlighting negative aspects of the Quality Model and boldly offered differing opinions.

8. Conclusion and Future Work

This chapter summarises the work done in this thesis and presents the future work.

8.1. Summary

In this work, we ascertained that it is important for software architects and developers to observe a set of design guidelines in order to build purposeful MSA. Such guidelines direct and advise on how to fundamentally structure, develop and maintain the microservices system. In addition, we realised the limited understanding and lack of standardised concepts in industry and academia around the quality of microservices applications and architectures, including how it can be conceptualised and explained. We also argued that following a particular design guideline seeks to solve a problem while affecting specific quality characteristics related to the stakeholder requirements.

In this thesis, we conducted a SLR to collect existing design guidelines i.e. best practices, design patterns, and design principles that are centered around constructing meaningful MSA. Subsequently, we proposed a categorisation and naming schema to structure the collected information in the form of a catalogue of 239 design guidelines. We also performed a mapping in the catalogue in order to determine a set of quality characteristics that are affected by each of the discovered design guidelines. Besides this, we developed a Quality Model tailored to microservices applications and architectures by extending the ISO/IEC 25010 QM. We proposed an initial version of the model, and evaluated it with 13 microservices professionals from industry and academia in order to finally put forward a revised Quality Model. This model reported and explained 70 quality characteristics that collectively and meaningfully describe the quality of microservices applications and architectures.

8.2. Future Work

Since the catalogue is derived entirely from the SLR method, the method can be adjusted to include more academic literature, and consequently discover new design guidelines. A similar direction is to consider the inclusion of grey literature, and subsequently perform a Grey Literature Review to determine particularly industry-driven design guidelines. For the catalogue, a tool could be developed to store all the information in one place and retrieve it conveniently when required.

While the search criteria for the SLR in this research was majorly centered around *best practices* and *design principles* of MSA, the catalogue includes *design patterns* of MSA that were naturally encountered during our investigation procedure as described in Chapter 4.2.2. In order to bridge the gap, future work in this direction could enhance the catalogue specifically for microservices design patterns.

In order to bridge the knowledge gap between industry and academia, we realised that assessing whether the design guidelines observed in industry for constructing MSA are in coherence with the discovered design guidelines of the catalogue is a very important issue and actually a study of its own. Since the Guidelines Catalogue was derived via a systematic method which is consistently backed by academic research, we did not consider a thorough evaluation of this artifact as part of this thesis and regard the same for future work.

Regarding the proposed Quality Model, we believe it is imperative to further study the model, especially its applicability in a given context, and refine it. As future work, it is worth considering the quality characteristics with the point of interest being the applicability of the model in particular use cases such as migration, service identification etc. We imagine that additional evaluations of the proposed QM could be conducted in terms of its clarity under the changes applied. The re-evaluations could include more questions that were not put forward as part of this thesis. Additionally, the responses given to the initially proposed QM could be compared to those given to the QM proposed after the evaluation. This would make the results more reliable, allowing to use them with higher confidence.

It is to be noted that a limitation of the ISO/IEC 25010 QM is that it caters only to product quality. We found that the model does not describe process quality that is utilised while designing and developing MSA. During our research, we discovered a few properties, such as *agility*, that are rather fit to software process quality than to software product quality, however we did not have enough input and resources to formulate a holistic Quality Model around them. This opens up a future research direction in formulating a relevant QM.

A. Appendix

A.1. Descriptions of the Discarded Quality Factors

Quality Attribute	**Quality Factor**	**Description of Quality Factor**	**Literature Source(s)**
Compatibility	Co-existence	Degree to which a microservice can perform its required functions efficiently while sharing a common environment and resources with other microservices, without impacting other microservices.	[34],[25],[12],[25],[51]
Functional Suitability	**Decomposability**	Degree to which an application can be partitioned by dividing it into a subset of business-driven microservices.	[10]
Functional Suitability	**Functionality**	Microservices should be designed to encapsulate well-defined set of functionality that is meaningful to the business.	[15],[17],[23],[24],[25],[26], [27],[28],[30],[31],[32],[33], [34],[43],[50],[51],[58],[60], [62],[63],[2],[9],[14],[16], [19],[21],[52],[53],[56]
Maintainability	**Dependability**	Degree to which a microservice is dependent on (or independent of) other microservices in a microservices system.	[32]

Table A.1.: Discarded Quality Factors with Descriptions

Quality Attribute	Quality Factor	Description of Quality Factor	Literature Source(s)
Maintainability	**Expandability**	Microservices applications should be flexible enough to meet the future growth needs.	[26]
Maintainability	**Releasability**	Ability to rely on the rapid and independent release of individual components in a microservices system.	[37]
Maintainability	**Traceability (in development)**	Degree to which information is available to track changes in the artefacts produced in a microservices system.	[10],[24],[33],[34],[35],[47],[60],[7]
Maintainability	**Verifiability**	Ability to show that a microservices system is consistent with its specification with respect to all the specified quality requirements.	[44]
Portability	Replaceability	Degree to which a microservices system or its underlying components (including technology stacks) can replace other systems or components for the same purpose in the same environment.	[17],[25],[26],[40]
Reliability	**Durability**	Services should be designed for long-term use, without requiring excessive maintenance.	[24],[28],[53]
Reliability	**Repeatability**	Ability to add and remove replicas of a microservice.	[16],[61]

Table A.2.: Discarded Quality Factors with Descriptions (continued)

Quality Attribute	**Quality Factor**	**Description of Quality Factor**	**Literature Source(s)**
Reliability	**Traceability (of errors)**	Ability to locate where in the microservices system an error originated, and how it travelled through several microservices.	[10],[41],[62]
Usability	Learnability	Degree to which a microservices system can be used by specified users to achieve specified goals of learning to use the system.	[34],[54],[48]
Usability	**Simplicity**	Ease with which a microservices system can be developed and used.	[34],[42],[62],[5],[6],[9]

Table A.3.: Discarded Quality Factors with Descriptions (continued)

Bibliography

[Abd+18] M. Abdellatif et al. "State of the practice in service identification for soa migration in industry." In: *International Conference on Service-Oriented Computing*. Springer. 2018, pp. 634–650 (cit. on p. 22).

[Abd+21] M. Abdellatif et al. "A taxonomy of service identification approaches for legacy software systems modernization." In: *Journal of Systems and Software* 173 (2021), p. 110868 (cit. on pp. 7, 8, 22).

[AFT19] U. Azadi, F. A. Fontana, and D. Taibi. "Architectural smells detected by tools: a catalogue proposal." In: *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE. 2019, pp. 88–97 (cit. on p. 18).

[AP19] A. Akbulut and H. G. Perros. "Performance analysis of microservice design patterns." In: *IEEE Internet Computing* 23.6 (2019), pp. 19–27 (cit. on p. 18).

[Bao+16] K. Bao et al. "A microservice architecture for the intranet of things and energy in smart buildings." In: *Proceedings of the 1st International Workshop on Mashups of Things and APIs*. 2016, pp. 1–6 (cit. on p. 18).

[BC06] B. Bloom and B. Crabtree. "Making sense of qualitative research: the qualitative research interview." In: *Medical Education* 40 (2006), pp. 314–321 (cit. on p. 33).

[BCK12] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice: Software Architect Practice_c3*. Addison-Wesley, 2012 (cit. on pp. 1, 3, 70).

[BD19] R. Bolscher and M. Daneva. "Designing Software Architecture to Support Continuous Delivery and DevOps: A Systematic Literature Review." In: *ICSOFT* (2019), pp. 27–39 (cit. on pp. 6–8, 13, 18).

[Ber+17] Y. Berkunskyi et al. "Using microservices in educational applications of IT-company." In: *2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON)*. IEEE. 2017, pp. 1208–1211 (cit. on p. 18).

[Bev95] N. Bevan. "Measuring usability as quality of use." In: *Software Quality Journal* 4.2 (1995), pp. 115–130 (cit. on p. 2).

[BGT16] B. Butzin, F. Golatowski, and D. Timmermann. "Microservices approach for the internet of things." In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2016, pp. 1–6 (cit. on p. 18).

[Bil+22] P. Billawa et al. "Security of Microservice Applications: A Practitioners' Perspective on Challenges and Best Practices." In: *arXiv preprint arXiv:2202.01612* (2022) (cit. on pp. 7, 21).

[BNB17] C. Berger, B. Nguyen, and O. Benderius. "Containerized development and microservices for self-driving vehicles: Experiences & best practices." In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE. 2017, pp. 7–12 (cit. on p. 18).

[BNK20] S. Baškarada, V. Nguyen, and A. Koronios. "Architecting microservices: Practical opportunities and challenges." In: *Journal of Computer Information Systems* 60.5 (2020), pp. 428–436 (cit. on pp. 7, 18, 33).

[BNS19] A. Brogi, D. Neri, and J. Soldani. "Freshening the air in microservices: resolving architectural smells via refactoring." In: *International Conference on Service-Oriented Computing*. Springer. 2019, pp. 17–29 (cit. on p. 18).

[Bog+21] J. Bogner et al. "Industry practices and challenges for the evolvability assurance of microservices." In: *Empirical Software Engineering* 26.5 (2021), pp. 1–39 (cit. on pp. 7, 8, 18).

[Cab+20] E. Cabrera et al. "Towards a Methodology for creating Internet of Things (IoT) Applications based on Microservices." In: *2020 IEEE International Conference on Services Computing (SCC)*. IEEE. 2020, pp. 472–474 (cit. on p. 18).

[Che+15] H.-M. Chen et al. "Architectural support for DevOps in a neo-metropolis BDaaS platform." In: *2015 IEEE 34th symposium on reliable distributed systems workshop (SRDSW)*. IEEE. 2015, pp. 25–30 (cit. on pp. 7, 19).

[Cle+20] S. B. Cleveland et al. "Tapis api development with python: best practices in scientific rest api implementation: experience implementing a distributed stream api." In: *Practice and Experience in Advanced Research Computing*. 2020, pp. 181–187 (cit. on p. 19).

[CNZ16] C. E. Cuesta, E. Navarro, and U. Zdun. "Synergies of system-of-systems and microservices architectures." In: *Proceedings of the International Colloquium on Software-Intensive Systems-of-Systems at 10th European Conference on Software Architecture*. 2016, pp. 1–7 (cit. on p. 19).

[Col+21] T. Colanzi et al. "Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices." In: *15th Brazilian Symposium on Software Components, Architectures, and Reuse*. 2021, pp. 61–70 (cit. on p. 22).

[CZL21] F. Chen, L. Zhang, and X. Lian. "A systematic gray literature review: The technologies and concerns of microservice application programming interfaces." In: *Software: Practice and Experience* 51.7 (2021), pp. 1483–1508 (cit. on pp. 5, 7, 19).

[Di 17] P. Di Francesco. “Architecting microservices.” In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE. 2017, pp. 224–229 (cit. on pp. 3, 70).

[DML17] P. Di Francesco, I. Malavolta, and P. Lago. “Research on architecting microservices: Trends, focus, and potential for industrial adoption.” In: *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE. 2017, pp. 21–30 (cit. on pp. 7, 19).

[Eis+20] S. Eismann et al. “Microservices: A Performance Tester’s Dream or Nightmare?” In: *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 2020, pp. 138–149 (cit. on p. 19).

[EZ21] A. El Malki and U. Zdun. “Evaluation of API Request Bundling and its Impact on Performance of Microservice Architectures.” In: *2021 IEEE International Conference on Services Computing (SCC)*. IEEE. 2021, pp. 419–424 (cit. on p. 19).

[Far+21] H. Farsi et al. “Following Domain Driven Design principles for Microservices decomposition: is it enough?” In: *2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA)*. IEEE. 2021, pp. 1–8 (cit. on p. 19).

[FF20] D. R. de Freitas Apolinário and B. B. N. de França. “Towards a method for monitoring the coupling evolution of microservice-based architectures.” In: *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse*. 2020, pp. 71–80 (cit. on p. 19).

[Gan+19] Y. Gan et al. “An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems.” In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019, pp. 3–18 (cit. on p. 19).

[GQ84] D. A. Garvin and W. D.-P. Quality. “Really mean.” In: *Sloan management review* 25 (1984), pp. 25–43 (cit. on p. 2).

[GSS68] B. G. Glaser, A. L. Strauss, and E. Strutzel. “The discovery of grounded theory; strategies for qualitative research.” In: *Nursing research* 17.4 (1968), p. 364 (cit. on p. 25).

[GT19] J.-P. Gouigoux and D. Tamzalit. ““Functional-First” Recommendations for Beneficial Microservices Migration and Integration Lessons Learned from an Industrial Experience.” In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2019, pp. 182–186 (cit. on p. 19).

[HBK20] S. Hassan, R. Bahsoon, and R. Kazman. “Microservice transition and its granularity problem: A systematic mapping study.” In: *Software: Practice and Experience* 50.9 (2020), pp. 1651–1681 (cit. on pp. 7, 19).

[HS19] A. Huf and F. Siqueira. "Composition of heterogeneous web services: A systematic review." In: *Journal of Network and Computer Applications* 143 (2019), pp. 89–110 (cit. on p. 18).

[HW+18] R. C. A. Hutapea, A. P. Wahyudi, et al. "Design Quality Measurement for Service Oriented Software on Service Computing System: a Systematic Literature Review." In: *2018 International Conference on Information Technology Systems and Innovation (ICITSI)*. IEEE. 2018, pp. 375–380 (cit. on pp. 7, 8, 13, 18).

[HWB17] S. Haselböck, R. Weinreich, and G. Buchgeher. "Decision guidance models for microservices: service discovery and fault tolerance." In: *Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems*. 2017, pp. 1–10 (cit. on p. 19).

[ISO01] ISO/IEC. *Software engineering — Product quality — Part 1: Quality model (ISO/IEC 9126)*. 2001 (cit. on p. 26).

[ISO11] ISO/IEC. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models (ISO/IEC 25010)*. 2011 (cit. on pp. 2, 26, 27, 71).

[Jam17] K. Jambunathan. "Microservice design for container based multicloud deployment microservice design for container based multi-cloud deployment." In: *Jour Adv Res. Dyn. Control Syst* (2017) (cit. on p. 20).

[JC19] C. T. Joseph and K. Chandrasekaran. "Straddling the crevasse: A review of microservice software architecture foundations and recent advancements." In: *Software: Practice and Experience* 49.10 (2019), pp. 1448–1484 (cit. on pp. 6, 7, 20).

[JPM17] P. Jamshidi, C. Pahl, and N. C. Mendonça. "Pattern-based multi-cloud architecture migration." In: *Software: Practice and Experience* 47.9 (2017), pp. 1159–1184 (cit. on p. 22).

[Kap19] A. Kaplunovich. "ToLambda–Automatic Path to Serverless Architectures." In: *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*. IEEE. 2019, pp. 1–8 (cit. on p. 20).

[Kas17] H. Kashfi. "Software engineering challenges in cloud environment: Software development lifecycle perspective." In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 2.3 (2017), pp. 251–256 (cit. on p. 2).

[Kee+07] S. Keele et al. *Guidelines for performing systematic literature reviews in software engineering*. Tech. rep. Technical report, ver. 2.3 ebse technical report. ebse, 2007 (cit. on pp. 10, 13–16, 23).

[KM19] J. Kazanavičius and 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 pp. 6, 20).

[Lai+21] R. Laigner et al. “Data management in microservices: State of the practice, challenges, and research directions.” In: *arXiv preprint arXiv:2103.00170* (2021) (cit. on pp. 7, 20).

[Li+18] F. Li et al. “Microservice patterns for the life cycle of industrial edge software.” In: *Proceedings of the 23rd European Conference on Pattern Languages of Programs.* 2018, pp. 1–11 (cit. on p. 20).

[Li17] S. Li. “Understanding quality attributes in microservice architecture.” In: *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW).* IEEE. 2017, pp. 9–10 (cit. on pp. 3, 70).

[Lot+19] J. Lotz et al. “Microservice architectures for advanced driver assistance systems: A case-study.” In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C).* IEEE. 2019, pp. 45–52 (cit. on p. 20).

[Lüb+19] D. Lübke et al. “Interface evolution patterns: balancing compatibility and extensibility across service life cycles.” In: *Proceedings of the 24th European Conference on Pattern Languages of Programs.* 2019, pp. 1–24 (cit. on p. 20).

[MA18] G. Márquez and 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. 7, 20).

[MSM16] N. Mehta, D. Steinman, and L. Murphy. *Customer success: How innovative companies are reducing churn and growing recurring revenue.* John Wiley & Sons, 2016 (cit. on p. 2).

[MT10] N. Medvidovic and R. N. Taylor. “Software architecture: foundations, theory, and practice.” In: *2010 ACM/IEEE 32nd International Conference on Software Engineering.* Vol. 2. IEEE. 2010, pp. 471–472 (cit. on p. 1).

[MT86] P. Y. Martin and B. A. Turner. “Grounded theory and organizational research.” In: *The journal of applied behavioral science* 22.2 (1986), pp. 141–157 (cit. on p. 25).

[Mul14] M. Muller. “Curiosity, creativity, and surprise as analytic tools: Grounded theory method.” In: *Ways of Knowing in HCI.* Springer, 2014, pp. 25–48 (cit. on pp. 25, 28, 36, 41).

[Neu15] S. Neuman. “Building microservices: Designing fine-grained systems.” In: *Oreilly & Associates Inc* (2015) (cit. on pp. 1, 12, 66).

[Nte+19] E. Ntentos et al. “Supporting architectural decision making on data management in microservice architectures.” In: *European Conference on Software Architecture.* Springer. 2019, pp. 20–36 (cit. on p. 20).

[Nte+20] E. Ntentos et al. “Assessing architecture conformance to coupling-related patterns and practices in microservices.” In: *European Conference on Software Architecture.* Springer. 2020, pp. 3–20 (cit. on p. 20).

[Nte+21] E. Ntentos et al. “Semi-automatic feedback for improving architecture conformance to microservice patterns and practices.” In: *2021 IEEE 18th International Conference on Software Architecture (ICSA)*. IEEE. 2021, pp. 36–46 (cit. on p. 20).

[Oli+20] T. de Oliveira Rosa et al. “A method for architectural trade-off analysis based on patterns: Evaluating microservices structural attributes.” In: *Proceedings of the European Conference on Pattern Languages of Programs 2020*. 2020, pp. 1–8 (cit. on pp. 7, 20).

[OMA18] F. Osses, G. Márquez, and H. Astudillo. “An exploratory study of academic architectural tactics and patterns in microservices: A systematic literature review.” In: *Avances en Ingenieria de Software a Nivel Iberoamericano, CIbSE* 2018 (2018), pp. 71–84 (cit. on p. 1).

[PJZ18] C. Pahl, P. Jamshidi, and O. Zimmermann. “Architectural principles for cloud software.” In: *ACM Transactions on Internet Technology (TOIT)* 18.2 (2018), pp. 1–23 (cit. on pp. 8, 21).

[Pre+21] J. P. D. Preti et al. “Monolithic to microservices migration strategy in public safety secretariat of Mato Grosso.” In: *2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*. IEEE. 2021, pp. 1–5 (cit. on p. 22).

[Ras18] S. Rasmy. “Microns: Commands for Building Bubble Microservices.” In: *2018 Sixth International Conference on Enterprise Systems (ES)*. IEEE. 2018, pp. 158–165 (cit. on p. 21).

[Ren+18] Z. Ren et al. “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 p. 22).

[Ric17] C. Richardson. “Microservices architecture (2014).” In: *URL https://microservices.io* (2017) (cit. on p. 1).

[RS22] V. Raj and K. Srinivasa Reddy. “Best Practices and Strategy for the Migration of Service-Oriented Architecture-Based Applications to Microservices Architecture.” In: *Proceedings of Second International Conference on Advances in Computer Engineering and Communication Systems*. Springer. 2022, pp. 439–449 (cit. on p. 22).

[SC98] A. Strauss and J. Corbin. *Basics of qualitative research, techniques and procedures for developing grounded theory., 2nd edn.(Sage: Thousand Oaks, CA)*. 1998 (cit. on p. 24).

[Shi17] S. K. Shivakumar. “Content Management System Architecture.” In: (2017) (cit. on p. 21).

[Sin+21] A. Singjai et al. “Patterns on deriving apis and their endpoints from domain models.” In: *26th European Conference on Pattern Languages of Programs*. 2021, pp. 1–15 (cit. on p. 21).

[Sol+21] J. Soldani et al. “The μTOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures.” In: *Software: Practice and Experience* 51.7 (2021), pp. 1591–1621 (cit. on p. 21).

[Sou+17] T. B. Sousa et al. “Engineering software for the cloud: Messaging systems and logging.” In: *Proceedings of the 22Nd European Conference on Pattern Languages of Programs.* 2017, pp. 1–14 (cit. on pp. 7, 21).

[SRS20] N. Siegmund, N. Ruckel, and J. Siegmund. “Dimensions of software configuration: on the configuration context in modern software development.” In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 2020, pp. 338–349 (cit. on p. 21).

[Sta15] I. O. for Standardization. *Quality Management Systems-Fundamentals and Vocabulary (ISO 9000: 2015).* ISO Copyright office, 2015 (cit. on p. 2).

[Tai+17] D. Taibi et al. “Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages.” In: *Proceedings of the XP2017 Scientific Workshops.* 2017, pp. 1–5 (cit. on p. 21).

[Val+20] J. A. Valdivia et al. “Patterns related to microservice architecture: a multivocal literature review.” In: *Programming and Computer Software* 46.8 (2020), pp. 594–608 (cit. on pp. 7, 8, 21).

[VF21] V. Velepucha and P. Flores. “Monoliths to microservices-Migration Problems and Challenges: A SMS.” In: *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST).* IEEE. 2021, pp. 135–142 (cit. on pp. 7, 22).

[W C+19] R. W. Collier et al. “MAMS: Multi-Agent MicroServices .” In: *Companion Proceedings of The 2019 World Wide Web Conference.* 2019, pp. 655–662 (cit. on p. 21).

[Wag13] S. Wagner. “Software product quality control.” In: (2013) (cit. on p. 3).

[WKR21] Y. Wang, H. Kadiyala, and J. Rubin. “Promises and challenges of microservices: an exploratory study.” In: *Empirical Software Engineering* 26.4 (2021), pp. 1–44 (cit. on pp. 7, 13, 21).

[Woh+12] C. Wohlin et al. *Experimentation in software engineering.* Springer Science & Business Media, 2012 (cit. on p. 72).

[Wur+17] M. Wurster et al. “Developing, deploying, and operating twelve-factor applications with TOSCA.” In: *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services.* 2017, pp. 519–525 (cit. on p. 21).

[XWQ16] Z. Xiao, I. Wijegunaratne, and X. Qiang. “Reflections on SOA and Microservices.” In: *2016 4th International Conference on Enterprise Systems (ES).* IEEE. 2016, pp. 60–67 (cit. on p. 22).

[Zim+20] O. Zimmermann et al. “Interface responsibility patterns: processing resources and operation responsibilities.” In: *Proceedings of the European Conference on Pattern Languages of Programs 2020*. 2020, pp. 1–24 (cit. on p. 22).

[ZZ21] J. Zhao and K. Zhao. “Applying Microservice Refactoring to Object-2riented Legacy System.” In: *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE. 2021, pp. 467–473 (cit. on p. 22).

Glossary

API Application Programming Interface

CI/CD Continuous Integration/Continuous Delivery

DevOps DevOps is a set of practices that combines software development (Dev) and IT operations (Ops).

GI Gesellschaft für Informatik e.V.

GLR Grey Literature Review

MC Microservices Community

MSA Microservices Architectures

QA Quality Attribute

QF Quality Factor

QM Quality Model

RESTful A RESTful API is an architectural style for an API that uses HTTP requests to access and use data.

RQ Research Question

SLR Systematic Literature Review

SOA Service-oriented Architectures